Multi-Path Routing combined with Resource Reservation

Israel Cidon
Department of Electrical Engineering
Technion, Haifa, Israel
e-mail: cidon@ee.technion.ac.il
and
Sun Microsystems
Mountain View, CA 94043-1100

Raphael Rom
Sun Microsystems
Mountain View, CA 94043-1100
e-mail: raphi@eng.sun.com
and
Department of Electrical Engineering
Technion, Haifa, Israel

*Yuval Shavitt**
Department of Computer Science
Johns Hopkins University, Baltimore, MD 21218-2691
e-mail: shavitt@cs.jhu.edu

Abstract

In high-speed networks it is desirable to interleave routing and resource (such as bandwidth) reservation. The PNNI standard for private ATM networks is a recent example for an algorithm that does this using a sequential crank-back mechanism. In this work, we suggest to do resource reservation along several routes in parallel. We present an analytical model that demonstrates that when there are several routes to the destination it pays to attempt reservation along more than a single route. Following this analytic observation, we present a family of algorithms that route and reserve resources along parallel subroutes. The algorithms of the family represent different trade-offs between the speed and the quality of the established route. The presented algorithms are simulated against several legacy algorithm, including PNNI crank-back, and exhibit higher network utilization and faster connection set-up time.

1 Introduction

Broadband integrated services digital networks (B-ISDN) are aimed to transport all electronic communication formats from e-mail to phone calls to home video. Communication in such high-speed networks is connection-oriented, i.e., before data can be transferred a connection should be established.

*Work done in its entirety while at the Department of Electrical Engineering, Technion, Haifa, Israel

In order to use the network resources efficiently, bandwidth reservations are made to ensure high probability of data arrival to its destinations. Traditionally, this task is done in two steps. First, a route must be selected, then, it is set-up and the resources are reserved along the route. However, in B-ISDN, resource availability may change rapidly or alternatively routing information may be outdated. In this environment a route that was computed in the first step, may lack the required resources in the second step. Combining the two steps was suggested in the ATM PNNI standard [5] to overcome this difficulty. When a link is considered for routing, resources are reserved along this link. If the route cannot be fully established the reservation process crank-back to an intermediate node, the resources downstream of this node are released, the rest of the route is recomputed, and the set-up is resumed. If the setup reaches the destination, the route is established with the required resources. In this work, we demonstrate that attempting to reserve resources along several sub-routes in parallel is better that attempting to do so along a single route, as in the PNNI case.

To quantify the effect of reserving resources along several routes in parallel, we analyze a model where several disjoint routes connect two nodes. We also assume that the source lacks reliable information regarding the availability of resources over these paths (which might model remote paths or a fast changing network environment). Under these assumptions, we show that for the analyzed model it always pays to use more than a single route, since the reservation success probability and the network utilization are the lowest when only a single route is considered. The penalty for using more than a single route is one of the analysis parameters. The advantage to multi-path routing is apparent even when this penalty is maximal, i.e., when the redundant resources in the sub-paths not selected to be part of the established route are held for the entire connection duration. There are two other advantages of multi-path routing, not captured by our analysis. First, since several routes are reserved in parallel the destination has the ability to chose the best. The second advantage is the speed of the search. When a single path routing and reservation algorithm fails to capture resources crank-back mechanisms [5] are used to try and move forward in another path. The move backwards wastes time and slows down the routing process. When multi-path algorithms are used several paths are searched in parallel, thus reservation failures in some paths do not influence the speed of the search on other paths.

We present a family of algorithms that route and reserve resources along parallel sub-paths. The algorithms (some of them presented in [11]) represent different trade-offs between the speed the search advances and the quality of the resulted route. The paths, along which the search is performed, are not necessarily disjoint, and can share nodes and links. In fact, any subgraph of the network graph, termed here a *diroute*, can be used to confine the search area. We present simulation results that demonstrate the advantage of the presented algorithms both in reservation success probability and in set-up time.

The idea of routing along several routes in parallel, was suggested in the context of WANs only in few works [8, 1]. The only work that considers reservation along multiple routes in parallel [6] analyzes and compares several parallel reservation algorithms for the special case where routing is performed along predefined routing trees, a model tailored for telephony networks. In LANs, multi-path routing was suggested in the context of flooding [4, 9, 10], since there the cost of packets going in many directions does not seem problematic.

The rest of the paper is organized as follows. In the next section, we first present the system model for our analysis. We then present a Markovian analysis, closed form expressions for two special cases, and a recurrence that improves the complexity of the numerical analysis. In section 3, we present the multi-path algorithms and their asymptotic complexity. In section 4, simulation results are presented. We present our concluding remarks in section 5.

2 Analytical Results

2.1 The system model

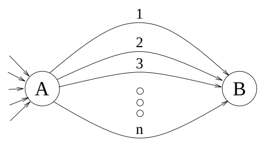

Figure 1: The analyzed system

Consider a source-destination pair of nodes that are connected by n disjoint routes, each of which can support a single connection at a time. Let the source node be A and the destination node be B (see figure 1). The connection-request arrival process is Poisson with intensity λ. When a connection-request arrives at node A, there is no knowledge about the availability of the routes. Thus, the reservation algorithm selects randomly and with equal probability k ($1 \le k \le n$) of the routes and tries to capture (reserve bandwidth) them. If more than one route is captured, only a single one is used for the connection while the others are released. The overall period of the reservation and the connection duration time is exponentially distributed with mean $1/\mu$. The period until an unused route is released is exponentially distributed with mean $1/\theta$ ($1/\theta \le 1/\mu$). Note that, $\theta = \mu$ models the case where the connection is short with regard to the reservation process. Thus, if several reservations succeed all the routes appear to be used (resources are reserved), while the destination is using only one of them, ignoring the rest. This happens, for example, when a short burst is sent preceded by a reservation request that tries to reserve sufficient resources on-the-fly [14, 2].

2.2 Analysis

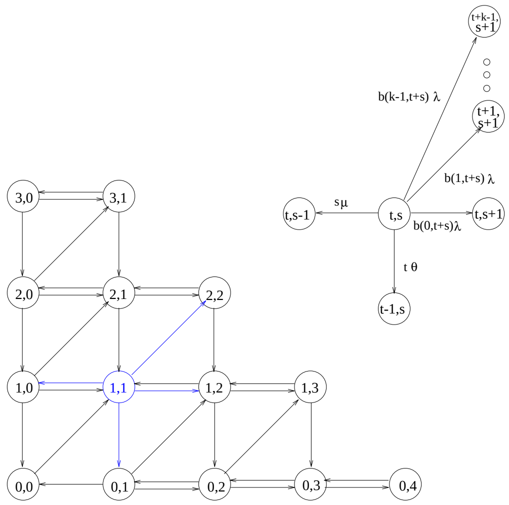

Figure 2: A Markov chain for a system with $n = 4$ and $k = 2$

The above system can be modeled by a continuous-time Markov chain with $n(n+3)/2$ states as illustrated in figure 2. Each state is represented by the ordered pair (t, s), where s is the number of routes that are used for transmission, and t is the number of redundant routes that were captured and are not used for transmission. The infinitesimal transition rates from state (t, s) to state (t', s'), $q_{t,s,t',s'}$ are

$$\begin{aligned} q_{t,s,t-1,s} &= t\theta \\ q_{t,s,t,s-1} &= s\mu \\ q_{t,s,t+i,s+1} &= b(i, t+s)\lambda \end{aligned} \tag{1}$$

where $0 \leq i \leq \min\{k-1, n-(s+t+1)\}$ and

$$b(i, m) = \binom{n-m}{i+1}\binom{m}{k-(i+1)} \Big/ \binom{n}{k}$$

We are interested in the connection reservation success probability, P_{suc}, that is proportional to the system throughput. To this end, the system steady state probabilities, $\pi_{t,s}$, should be found by solving the system equilibrium equations together with the probability conservation relation. P_{suc} is then given by

$$P_{suc} = \frac{\lambda_{out}}{\lambda_{in}} = \rho^{-1} \sum_{t=0}^{n-1} \sum_{s=1}^{n} s \cdot \pi_{t,s} \tag{2}$$

where $\rho = \lambda/\mu$.

The above straight-forward numerical solution requires $O(n^6)$ basic operations (or $\approx O(n^5)$ operations if sophisticated matrix inversion procedures are used). In the following sections, we shall describe less complex solution methods.

2.3 Analysis of special cases

When $k = 1$, t always equal to 0, thus the system can be modeled by an $n+1$-state birth-death process with transition rates $\lambda_s = (n-s)\lambda/n$ and $\mu_s = s\mu$.

When $\theta = \mu$, the system is represented by a single number signifying the total amount of occupied routes. For the case where $k = n$, we can write the equilibrium equations $(\lambda + s\mu)\pi_s = (s+1)\mu\pi_{s+1}$, where $0 \leq s < n$.

Calculating the success probabilities for both cases gives the same result [13]:

$$P_{suc} = \frac{n}{n+\rho} \tag{3}$$

I.e., when the penalty for capturing more than one link is maximal, a system where a connection attempts to capture one link performs identically to a system where a connection attempts to capture all the links.

2.4 Reducing the analysis complexity using recurrence

For the general case, we can reduce the computation complexity of section 2.2 by using recursion. Our aim is to write the steady state probabilities of all the system states, $\pi_{t,s}$, as functions of $\pi_{0,s}$, $0 \leq s \leq n+1$. Then, we can write n equilibrium equations and together with the probability conservation equation we obtain $n+1$ linear equations that can be solved with $O(n^3)$ operations. We shall demonstrate each step in the computation process for the case where $k = 2$, a general solution can be found in [13].

Using the Markov chain of Figure 2 and the transition rates of Eq. 1, we can write the following $n(n+1)/2 - 1$ equilibrium equations

$$\begin{aligned} q_{t,s,t,s}\pi_{t,s} &= q_{t-1,s-1,t,s}\pi_{t-1,s-1} + q_{t,s-1,t,s}\pi_{t,s-1} \\ &\quad + q_{t+1,s,t,s}\pi_{t+1,s} + q_{t,s+1,t,s}\pi_{t,s+1} \quad (4) \\ &\quad 2 \leq t \leq n-2, \ 0 \leq s \leq n-(t+1) \\ q_{0,s,0,s}\pi_{0,s} &= q_{0,s-1,0,s}\pi_{0,s-1} + q_{1,s,0,s}\pi_{1,s} \\ &\quad + q_{0,s+1,0,s}\pi_{0,s+1} \quad 1 \leq s \leq n-1 \\ q_{0,0,0,0}\pi_{0,0} &= q_{1,0,0,0}\pi_{1,0} + q_{0,1,0,0}\pi_{0,1} \end{aligned}$$

where $q_{t,s,t,s} = t\theta + s\mu + \sum_{l=0}^{\min\{n-(t+s+1),1\}} b(l, t+s)\lambda$ is the transition rate out of state (t, s).

Now, we can write the following recursion relations for $\pi_{t,s}$, $t > 0$:

$$\begin{aligned} \pi_{1,0} &= (q_{0,0,0,0}\pi_{0,0} - q_{0,1,0,0}\pi_{0,1})/q_{1,0,0,0} \\ \pi_{1,s} &= (q_{0,s,0,s}\pi_{0,s} - q_{0,s-1,0,s}\pi_{0,s-1} - \\ &\quad q_{0,s+1,0,s}\pi_{0,s+1})/q_{1,s,0,s} \quad s = 1, 2, \ldots, n-1 \\ \pi_{t,s} &= (q_{t-1,s,t-1,s}\pi_{t-1,s} - q_{t-2,s-1,t-1,s}\pi_{t-2,s-1} - \\ &\quad q_{t-1,s-1,t-1,s}\pi_{t-1,s-1} - q_{t-1,s+1,t-1,s}\pi_{t-1,s+1}) \\ &\quad /q_{t,s,t-1,s} \quad t = 2, 3, \ldots, n \quad s = 1, 2, \ldots, n-t \end{aligned}$$

The above recurrence suggests that all $\pi_{t,s}$ can be written as functions of $\pi_{0,s}$, i.e.,

$$\pi_{t,s} = \sum_{l=0}^{n} C_{t,s}(l)\pi_{0,l},$$

It is easier to calculate the recurrence for the coefficients, $C_{t,s}(l)$, rather than directly for $\pi_{t,s}$. First, we calculate the coefficients of $\pi_{1,s}$ by

$$\begin{aligned} C_{1,s}(s) &= q_{0,s,0,s}/q_{1,s,0,s} \quad s = 0, 1, 2, \ldots, n-1 \\ C_{1,s}(s-1) &= -q_{0,s-1,0,s}/q_{1,s,0,s} \quad s = 1, 2, \ldots, n-1 \\ C_{1,s}(s+1) &= -q_{0,s+1,0,s}/q_{1,s,0,s} \quad s = 0, 1, \ldots, n-1 \\ C_{1,s}(l) &= 0 \quad |l-s| > 1 \end{aligned}$$

Next, we calculate the coefficients of $\pi_{t,s}$ for $t = 2, 3, \ldots, n-1$. For $k = 2$ the recurrent calculation is done by

$$C_{t,s}(m) = (q_{t-1,s,t-1,s}C_{t-1,s}(m) \\ -q_{t-1,s+1,t-1,s}C_{t-1,s+1}(m) - q_{t-1,s-1,t-1,s}C_{t-1,s-1}(m) \\ -q_{t-2,s-1,t-1,s}C_{t-2,s-1}(m))/q_{t,s,t-1,s}$$

The recurrence calculation requires $O(k \cdot n^3)$ operations ($1 \le k \le n$). $n+1$ equilibrium equations are not used to derive the recurrence, thus n of them can be used together with the probability conservation equation to achieve the following $n+1$ linear equation system.

$$\begin{aligned} q_{t,n-t,t,n-t}\pi_{t,n-t} &= q_{t,n-(t+1),t,n-t}\pi_{t,n-(t+1)} + \\ & \quad q_{t-1,n-(t+1),t,n-t}\pi_{t-1,n-(t+1)} \\ & \quad 1 \le t \le n-1 \qquad (5) \\ q_{0,n,0,n}\pi_{0,n} &= q_{0,n-1,0,n}\pi_{0,n-1} \\ \sum_{(t,s)} \pi_{t,s} &= 1 \end{aligned}$$

Using the recurrence on the coefficients we transform Eq. 5 to an $n+1$ linear equation system with the variables $\pi_{0,l}$ whose solution complexity is lower than $O(n^3)$.

2.5 Numerical results

Figure 3 depicts the throughput, λP_{suc}, as a function of ρ for $n = 6$ and $\mu/\theta = 1$, results for other values of n and other μ/θ ratios can be found in [13]. For all the calculated parameters, $k = 1$ always achieves the lowest success probability when compared with other values of k (the solid line that is the lowest in the presented graph). $k = 1$ is the case where only one route is considered for routing as in PNNI and other classical routing algorithms, while $k > 1$ represents parallel routing. This serves as an incentive to reserve routes in parallel. For the case where the penalty is maximal, i.e., $\mu = \theta$, the best throughput is achieved for $k \cong n/2$. As the penalty decreases, larger values of k become optimal. Tables that compare the system throughput for fixed n values when $\mu = \theta$, for all the values of k can be found in [13].

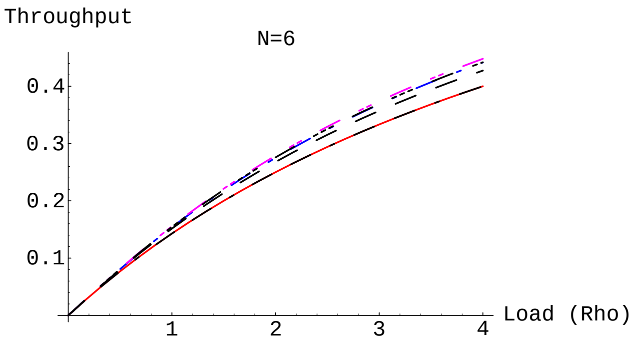

Figure 3: Normalized throughput for $n = 6$ when $\theta/\mu = 1$

3 Algorithms for Multi-Path Routing

In this section, we present a family of algorithms that route and reserve resources along parallel sub-routes. The algorithms offer a trade-off between the quality of the selected path and the set-up speed.

The algorithms are based on a flooding algorithm that attempt to reserve bandwidth along several possible routes. Generally, searching from scratch for a route between two nodes in the entire network is inefficient in terms of communication cost and set-up time. Thus, we assume that a topology-update algorithm informs the nodes about the (slow) changes in the network topology and about the cost of the links. When a node wishes to establish a connection, it searches for the best route in a subgraph of the network that contains links that lead to the destination and that have a “reasonable” cost. We call this restricted subgraph a *diroute*. The selection of the diroute can be made by the source node or, in a distributed manner by the nodes on the graph [13]. To avoid reserving resources in the entire diroute until the optimal route is chosen, the algorithms release resources from segments of the diroute as soon as they learn that these segments are inferior to another segment where reservation was made. The implementation of this *early release* of bandwidth is possible since a node in the diroute that receives two or more reservation messages from different links, can locally select the best one, and can locally decide to release the bandwidth from the other incoming paths.

Three sub-families of algorithms are presented:

Fast algorithms where the reservation message travels to the destination as fast as possible, but the selected path might be suboptimal.

Slow algorithms where the reservation message travels to the destination at the speed of the slowest path, but the selected path is optimal and the message complexity is linear in the number of diroute links.

Superfast algorithms where the reservation message from the source to the destination and the positive acknowledgment from the destination to the source, both travel as fast as possible. Similar to the fast algorithms, the selected path might be suboptimal. The superfast algorithms use initial multicast connections that are gradually reduced to a unicast connection.

3.1 Problem definition

The main objective of the routing algorithms presented here is to set-up connections in a network. The set-up process includes a reservation of network resources for the connection, if these resources are not available the set-up fails, or in other words, the connection is blocked. Our aim is to minimize the blocking probability for a connection.

Before a connection is set-up, a route must be selected. The routing decision influence the blocking probability. E.g., if two routing algorithms differ in the length of the selected routes, and thus in the consumption of network resources, usually the probability of successfully finding sufficient resources for more connections is lower for the algorithms that wastes resources. Thus, the quality of the routing decision also effect the blocking probability in the network.

We would like to route and reserve resources for a connection in a way that (1) minimizes the blocking probability by attempting several routes in parallel, (2) always considers the feasible route with the minimal cost (the shortest route), (3) selects a route with a cost that is close to the min-cost feasible route, and (4) limits the flow of information and the use of computing resources in that process. To this end we define the *diroute*, a collection of directed links where the route is searched for. The construction of the diroute remains an open problem. Shavitt [13] gives some insights to the diroute construction and discuss some heuristics.

Each link in the diroute is associated with a cost, available resources, and processing/transmission delays. While the cost of all the links might be known before the algorithm starts its operation, the rapidly changing availability of resources and the processing delay are unknown in advance. In addition, several connections might compete for resources and even slight timing differences among the reservation messages can dictate which connections succeed and which will fail. Under the above constraints, we wish to find (and reserve) a path in the diroute between the source and the destination nodes that has a favorable cost and sufficient resources.

3.2 Fast algorithms

In this section we present two algorithms: the *basic fast algorithm* which is the basic algorithm of this family, and the *logarithmic algorithm* which improves the message complexity of the first. Some variants of these algorithms, where the destination response is delayed are described in [13]. The basic fast algorithm is described in details since it serves as a building block for the logarithmic algorithm.

3.2.1 The fast algorithm

The main thrust of the algorithm is to reach the destination with a feasible path as fast as possible (using a flooding-like approach), altering the path if better alternatives are found in time, and releasing superfluous reserved bandwidth as soon as it is identified.

We describe here the algorithm's operation; an example and a pseudo-code formal description are given in [13]. Since the algorithm is executed on the diroute the term "all outgoing links" refers only to links of the diroute. A link is called *eligible* if it has enough available resources to potentially be part of the requested route.

The source node starts the algorithm by reserving the requested resources in all the eligible outgoing links and then sending a *Request* message along these links. The *Request* message carries a specification of the resources requested for the connection and the total cost of the path from the source to the current node. Other quantities, such as route delay, can be accumulated, as well [13].

A node that receives the first *Request* message on some incoming link l reserves the needed resources in all the eligible outgoing links and forwards the *Request* message on all these links. The node marks link l as the link to the father; this marking will be used to forward the positive acknowledgment message, *Accept*, or the negative acknowledgment message, *Reject*, back to the source. This establishes the fast flooding towards the destination node.

A node sends a *Reject* message if it has no outgoing link with sufficient resources. In this case a $Reject(0)$ message is sent to signal the lack of resources. Another scenario where a *Reject* message is sent is when a node receives two *Request* messages. In this case the sender of the *Request* with the higher cost is answered with $Reject(x)$, where x is the cost received in the other *Request*. This comes to save the transmission of *Request* messages that are bound to be answered by *Reject*. A third situation for sending *Reject* is when a node receives *Reject* for every *Request* it sent. If the source receives *Reject* messages from all its outgoing links the reservation fails and the connection can not be established.

A node that receives another *Request* message with a better path-cost, sends a *Reject* message to its father, and if the message arrives from a different incoming link the node changes the father marking. In addition, the node forwards the message on all the outgoing links where the previously sent *Request* was not yet answered and on all the outgoing links where *Reject* was received but the parameter carried by the received *Reject* indicates that the new *Request* might not be rejected. Note that a node is required to hold a counter on the number of unanswered *Request*s. Every *Reject* but the last one are ignored. The change of the father marking replaces the old route with a better one and causes the *early release* of reserved bandwidth on the previously marked link which is now known to be inferior. A *Request* message with a worse path-cost then the current one is answered with *Reject*. Thus, with the early release mechanism a node always keeps, at most, a single incoming path with resource reservation.

The destination node answers with *Accept* to the first *Request* it receives, and with *Reject* to all the other subsequent *Request* messages. This ensures the fastest establishment of a connection with the information known thus far. The *Accept* message is immediately forwarded by every node to its father until it reaches the source, at which time it can start transmitting data. A node that forwarded an *Accept* message will respond with *Reject*(0) to every subsequent *Request* it receives to signal the termination of the route establishment process.

When the source receives the *Accept* message, it knows that a reserved path exists to the destination. However, the source does not know when all the nodes in the diroute terminate their part of the algorithm (which, at this point, is irrelevant to the task of path establishment) and more important, when the algorithm message-terminates [7], i.e., when no more messages belonging to the algorithm exist in the network. A node that is not part of the selected path can never learn that the search ended successfully. The problem with inability to identify termination is that nodes cannot release the variables allocated to the current execiution of the algorithm. Note, that this has nothing to do with the resources required from the network which are guaranteed to be released. A termination mechanism is described in [13].

3.2.2 The Logarithmic Algorithm

As will be shown in section 3.5, the message complexity of the basic fast algorithm might be exponential in the number of nodes in the diroute. We wish to devise an algorithm that has the merits of the basic fast algorithm and has an acceptable message complexity.

The new algorithm is based on the basic fast algorithm which is supplemented with a counter for the arriving *Request* messages. When the first *Request* message arrives the counter is set to 1, and it is increased by one for every arrival of a *Request* message with a better path cost (counting all the *Request* messages does not increase the message complexity). *Request* messages are sent only if the arriving *Request* advances the counter to a value that is a natural power of two, i.e., only the first, the second, the forth, etc. arriving *Request* messages are forwarded. Essentially, the number of messages sent by a node is logarithmic compared to the number of messages that would be sent by the fast algorithm in the same network. Note that the early release of bandwidth is performed for all the arriving *Request* messages. For the message complexity analysis, see section 3.5.

3.3 Slow algorithms

The fast algorithms attempt to establish a connection fairly fast at the cost of path optimality. For those cases where path optimality is important the fast algorithms will therefore not do and we offer here slower algorithms that guarantee path optimality at the cost of connection establishment latency. We present two algorithms that differ only in their use of the early release mechanism. Thus most of the presentation and the discussion is done for the two together. A formal description of the algorithms appears in [13].

The main difference between the fast and slow algorithm families is the decision when to forward the *Request* message. In the fast algorithms, a node sends a *Request* immediately after it receives the first *Request*, and sends additional *Request*s when it learns about better paths. In the slow algorithms, a node sends a *Request* message only after it knows the optimal path from the source to itself, i.e., only after it receives *Request* messages from all the incoming links. To prevent deadlocks or livelocks the diroute must be acyclic. Heuristics are needed to transform the diroute, which is a general directed graph, to a directed acyclic graph without deleting any link that belongs to the minimal feasible path.

In the *S-1* algorithm, a node waits until it receives messages from all its incoming links before it sends *Reject* messages to the links which were not selected as the father. In *S-2*, for every *Request* that arrives, its path cost is compared to that of the father link and a *Reject* is sent to the inferior of the two. If needed the father marking is changed, too.

S-2 is more efficient from the network resources point of view, since it releases resources as soon as possible. S-1 is more resilient to failures. In addition, if the algorithm succeed to reserve a path when a failure occurs – this path is guaranteed to be optimal for the remaining links (see [13]).

As already stated, both algorithms delay the submission of *Request* messages until *Request* messages are received from all the incoming links. It is therefore clear that the slow algorithms progress according to the slowest path, which is typical to this type of flooding algorithms [12]. Furthermore, depending on implementation, it might be necessary to send messages even on non-eligible links to assure proper progress of the algorithms.

A node that receives messages from all its outgoing links can release all the algorithm's variables as it will not receive any more messages, thus, a separate termination phase is not needed here.

3.4 SuperFast algorithms

In some cases, the speed of the connection establishment is more important than the cost of the established path. In the fast algorithms (see section 3.2), the *Request* message reaches the destination as fast as possible, however, for the *Accept* message that travels back to the source the journey may be longer, for two reasons. First, the delay across a link in the two directions may not be the same. Second and more important, by the time the *Accept* message reaches a node on the selected path, this node may have chosen another incoming path with a better cost but longer delay. In the worst case (see example in [13]), the fast algorithm may select a path which is the slowest possible.

In this section, we present an algorithm which is the fastest possible. The algorithm is based on two floodings, the first of a *Request* from the source to the destination and the second of an *Ack* started by the destination immediately after the receipt of the first *Request*. Since it is proved [12] that flooding is the fastest data delivery algorithm and we use flooding both ways, with the superfast algorithms the source can start transmitting at the earliest possible time.

The superfast algorithm we present here, SF-1, is simple as it does not attempt to cost-optimize the selected path. The path that is selected is the one that reaches the source first. Due to the simplicity of its design, the message complexity of SF-1 is optimal. The algorithm is started by the source node that sends a *Request* message with the requested bandwidth on all the eligible links. A node that receives the first *Request* on link l forwards the message on all the outgoing eligible links, and marks l as the father link. In addition, the node inserts the requested connection to its routing table as a multicast connection. When the destination receives the first *Request*, it starts a flooding algorithm (such as PI [12]) with the *Ack* message. The flooding travels only on incoming links, i.e., every node that receives the first *Ack* forwards it to all the incoming links. When the destination receives the *Ack* message it can start transmitting along all the *active* links. The data packets travels along multiple routes until they either reach the destination or reach a node where the connection is not in the routing table; in the second case, the packets are discarded. To minimize the waste in network resources, a node that receives additional *Request*s replies with a *Reject*. A node that receives a *Reject* message deletes the rejected link from the routing table and releases the allocated resources. Consequently, data packets will not be transmitted on this link. A node that receives *Reject* messages from all its outgoing links sends a *Reject* on its father link, and deletes the connection from its routing table.

When SF-1 is executed as described above, the initially-set multicast connection is pruned until eventually a single acyclic path remains active, at which time no more control messages are sent by the algorithm. However, this situation when the algorithm message terminates cannot be identified by any of the nodes, and thus the algorithm variables cannot be released. To identify termination, we add an *Accept* message that travels from the destination to the source along the selected path. When the source receives either *Reject* or *Accept* messages from all its outgoing eligible links it starts a *Terminate* flooding in the same manner described for the fast algorithm (see section 3.2.1).

The problem with SF-1 is its inability to correct the selected path. We would like to have the flexibility in path selection of the fast algorithm with the super-fast reaction time. To this end Shavitt [13] introduces SF-2, an algorithm with super-fast reaction time but with (almost) the same flexibility of the fast algorithm.

3.5 Asymptotic complexity analysis

In this section, the message complexity of the presented algorithms is analyzed. We start with the trivial analysis of the slow algorithms, move to the more complex analysis of the basic fast algorithm, present bounds on the complexity of the logarithmic algorithm, and finally, we analyze the message complexity of the superfast algorithms. In all the algorithms, the number of *Reject* and *Accept* messages is at most equal to the number of *Request* messages. Thus, throughout the analysis we count only *Request* messages; for the superfast algorithms the *Ack* and *Commit* messages are counted, too.

In the **slow algorithms**, exactly one *Request* is sent on every link. This translates to a message complexity of $\Theta(|E|)$ which is optimal for this problem.

Intuitively, the message complexity of the **fast algorithm** is exponential since, in the worst case, a message can be sent for every path in the diroute. We shall show now that this is true even for the case where the diroute is acyclic.

Consider the acyclic diroute of figure 4 where node i has a link to node j if $i < j$. Node 1, receives a single *Request* message. Node i receives all the messages received by node $i-1$ directly and, in addition, the messages sent by node $i-1$ to node i. In the worst case, node $i-1$ forwards all the messages it receives to node i. Thus the number of messages received by node i is twice the number of messages received by node $i-1$, and the total amount of messages that are sent by the fast algorithm is $\sum_{i=1}^{|V|-1} 2^{i-1} = O(2^{|V|})$. Note that to achieve an exponential number of messages, as the upper bound suggests, the number of different path costs should be exponential. If the link costs can only take values from a discrete set, the number of possible path costs is bounded by a polynomial of $|V|$ whose degree is the number of members in the set.

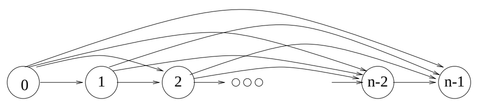

Figure 4: The worst case acyclic diroute

Since the number of messages transmitted on a link is bounded by the number of path costs the algorithm complexity is pseudo-polynomial for this case.

To analyze the **logarithmic algorithm**, we first observe that if node i receives R_i *Request* messages it transmits $\lfloor \log_2 R_i \rfloor + 1$ *Request* messages on every outgoing link. Denote by j the node that transmits the maximum number of *Request* messages during an execution of the algorithm. Using the above observation, we can write

$$S_j \leq \lfloor \log_2 \sum_{i \neq j} S_i \rfloor + 1 \leq \log_2 (n-1) S_j + 1 \qquad (6)$$

where $n = |V|$, and S_i is the number of messages transmitted by node i on each of its outgoing links.

For acyclic diroutes, we can derive a tight upper bound on the message complexity by counting the number of messages that are sent for the worst case diroute shown in figure 4. Since node i receives all the messages received by node $i-1$ plus the ones transmitted by node $i-1$ we can write

$$S_i = S_{i-1} + \log_2 S_{i-1} + 1 \qquad (7)$$

It can be shown by induction that $S_n = O(\log n)$, and thus the algorithm message complexity is $O(|E| \log |V|)$.

To calculate an upper bound on message complexity for a general diroute, assume that the number of messages that are sent by node j is proportional to $\log^\alpha n$ for some $\alpha > 1$ (i.e., $S_j = O(\log^\alpha n)$). Substituting in equation 6 we get

$$\begin{aligned} S_j = \log^\alpha n &\leq \log_2[(n-1) \cdot \log^\alpha n] + 1 \qquad (8) \\ &= \log(n-1) + \alpha \log\log n + 1 \end{aligned}$$

But, $\log^\alpha n$ is asymptotically greater than $\log(n-1) + \alpha \log\log n + 1$ for every $\alpha > 1$; which implies that the number of messages transmitted by node j on every link is bounded by $o(\log^\alpha n)$. The message complexity of the algorithm is, thus, in $o(|V|^2 \log^\alpha |V|)$ for every $\alpha > 1$. Since for an acyclic diroute we have found that the logarithmic algorithm sends $O(|E| \log |V|)$ messages, we have only a minor gap between the algorithm performance and the upper bound for the case where the number of links is in $O(|V|^2)$.

In **SF-1**, exactly one *Request*, one *Commit* and one *Ack* are sent on every link. This translates to a message complexity of $\Theta(|E|)$ which is optimal for this problem.

4 A Simulation Study

In this section, the fast algorithm is compared with several legacy algorithms that are described below. In all the simulations, a connection duration time is Exponentially distributed with parameter μ. When a connection request arrives at the network a reservation attempt is made. If the reservation process fails more attempts are made, each after a period that is Exponentially distributed with parameter η. If a threshold time, T_{thr}, passed since the first reservation attempt the connection is rejected.

Reservations are made only along *proper links*. The definition of the proper links changes from simulation to simulation. In the simplest case, proper links are only the ones that belong to some shortest path between the source and the destination. A path between the source and the destination that is comprised of proper links is called a *proper path*.

The simulated legacy algorithms are:

Source routing — The source selects a proper path and writes it in the header of the packets. In all the retries of the same connection request, the same path is selected. In case more than a single proper path exists, one is randomly selected for every arriving request.

Deflection routing — The source randomly selects an eligible proper link, and sends a reservation along this link. Each intermediate node, acts the same way as the source until the reservation request reaches the destination or until no eligible proper link is found.

Crank-back routing — Same as above, only if no eligible proper link is found, the algorithm cranks-back one hop and tries to continue from there.

Due to space limitation we describe here results from one simulation set. More results can be found in [13].

The results reported here are from a simulation of a grid network. Between every pair of nodes in the grid, two connections in each direction were simulated, all the connections requests arrive according to a Poisson process with the same intensity. The sum of all the intensities, λ is a simulation variable. The link cost for the fast algorithm is set to 1. The other parameters are: $\eta = 20$, $T_{thr} = 100$, and $\rho = \lambda L(m)/(4m(m-1)\mu)$, where $4m(m-1)$ is the number of directed links in an $m \times m$ grid, and $L(m) = 2m/3$ is the average[1] length of a connection in hops.

Figure 5 shows the reservation success probability for connections in a 5×5 grid. Figure 6 shows the success probability when the threshold value is decreased. Clearly, the Fast algorithm outperforms the other algorithms. Note, that for long connections, i.e., those

[1] This is a direct result of order statistics see [3, sec. 5.4].

whose hop count is $2(m-1)$ or $2(m-1)-1$, the Fast algorithm is much better than the other algorithms. This suggests that the Fast algorithm is fairer in the sense that long connection observe a success probability similar to the average.

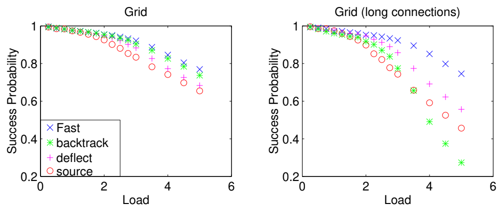

Figure 5: A comparison of the reservation success probability as a function of the load, ρ, in a 5x5 grid.

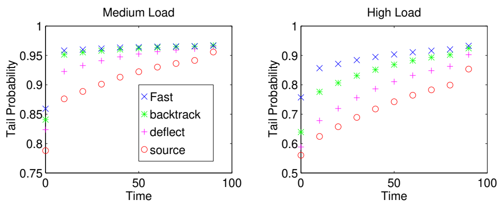

Figure 6: A comparison of the reservation success probability as a function of the threshold time for several loads in a 5x5 grid.

5 Concluding Remarks

We demonstrated that multi-path routing is an attractive solution in cases when there is a need to set up a route fast in a network where resource availability changes rapidly. A discussion on the connection establishment time and failure handling can be found in [13]. We believe that there is much more work to be done, and point to a few directions below.

The adaptation of the suggested algorithms to multicast connections seems natural. When routes to several destinations are searched, the search does not stop when a destination is found but continues along the diroute to look for more destinations. Several options that are discussed in [13] can serve as a starting point to construct efficient multicast routing algorithms.

Diroute construction is an area where there is much work to be done both for distributed methods, for which some directions were pointed in [13] and for centralized methods. It is interesting to try and quantify the quality of the diroute, and then look for algorithms or heuristics that try to achieve an optimal diroute. This question arises also in the context of multicast connections.

References

[1] Y. Azar, J. Naor, and R. Rom. Routing strategies for fast networks. In *IEEE INFOCOM'92*, pages 170 – 179. IEEE, 1992.

[2] P. E. Boyer and D. P. Trachier. A reservation principle with applications to the ATM traffic control. *Computer Networks and ISDN Systems*, 24:321 – 334, 1992.

[3] H. A. David. *Order Statistics*. Wiley, second edition, 1981.

[4] D.J. Farber and M. Parulkar. A closer look at NOAHNET. In *SIGCOM'86*, pages 205 – 213. ACM, August 1986.

[5] ATM Forum. Private network network interface (PNNI) v1.0 specifications, June 1996.

[6] R. Hwang, J. F. Kurose, and D. Towsley. On-call processing delay in high speed networks. *IEEE/ACM Transactions on Networking*, 3(6):628 – 639, December 1995.

[7] A. Itai and M. Rodeh. Symmetry breaking in distributed networks. In *FOCS'81*, pages 150 – 158. IEEE, 1981.

[8] O. Lesser and R. Rom. Routing by controlled flooding in communication networks. In *IEEE INFOCOM'90*, pages 910 – 917, 1990.

[9] C. Petitpierre. Meshed local computer networks. *IEEE Communications Magazine*, 22(8):36 – 40, August 1984.

[10] H. K. Pung. ARBnet+: an experimental mesh-like local area network. In *SICON'89*, pages 301 – 306, 1989. Singapore.

[11] R. Rom and Y. Shavitt. A combined fast-routing and bandwidth-reservation algorithm for ATM networks. In *International IFIP/IEEE Conference on Broadband Communication*, pages 76 – 87. Chapman & Hall, April 1996.

[12] A. Segall. Distributed network protocols. *IEEE Transaction on Information Theory*, IT-29(1):23 – 35, January 1983.

[13] Y. Shavitt. *Burst Control in High-Speed Networks*. PhD thesis, Technion – Israel Institute of Technology, Electrical Engineering Dept., Technion City, Haifa 32000, Israel, June 1996.

[14] J. S. Turner. Managing bandwidth in ATM networks with burtsy traffic. *IEEE Network*, 6(5):50 – 58, September 1992.