A Fast Bypass Algorithm for High-Speed Networks

Israel Cidon[*]
Sun Microsystems
Mountain View, CA 94043-1100

Raphael Rom[†] *Yuval Shavitt*
Department of Electrical Engineering
Technion, Haifa, Israel

Abstract

In this work we suggest an algorithm that increases the reservation success probability for bursty traffic in high speed networks by adding flexibility to the construction of the routes. The algorithm is simple enough to be implemented by cheap hardware. It causes no additional delay to packets that use the original route, and a very small delay to the packets that are rerouted. In addition, the presented algorithm has a minimal communication overhead due to the local nature of its work. Two high-speed network models are considered: source routing and ATM.

1 Introduction

High speed networks are intended to support applications with widely varying traffic characteristics: from short database queries to long video streams. In order to use the network resources efficiently, bandwidth reservations are made to ensure high probability of data arrival to their destinations. For applications such as constant bit rate video or voice conversations this is the right approach. However, for bursty traffic, i.e., traffic whose intensity varies in time, reservation itself introduces non-negligible overhead. Moreover, the widely varying nature of bursty traffic indicates that a simplistic burst reservation mechanism would not suffice. The scheme must consider the burst size and the timing constraints in its operation, as we briefly explain below.

Short bursts are those whose transmission time is not more than a few round trip delays. For such bursts waiting for a reservation, that itself takes a few round trip delays, is clearly not acceptable. The best method for this type of bursts is to make an initial zero-bandwidth reservation and subsequently to send the data without reservation and use time-outs (possibly at a higher layer) to detect failures. Turner [12] suggested an on-the-fly reservation scheme. In his scheme a burst that arrives to an ATM switch and finds sufficient bandwidth for its cells, reserves the required bandwidth (to prevent new bursts from disturbing this one) and proceeds to the next switch towards its destination. This scheme does not guarantee that a burst that succeeds in reserving enough bandwidth in one switch will also succeed in the next one along the route. Hence the choice of the route is crucial in the success of the on-the-fly reservation.

The same solution does not fit longer bursts. Here, the overhead of reservation is not as bothersome so traditional reservation algorithms can be used. Note, however, that this approach is valid only if there is enough storage at the source to hold the burst data until positive acknowledgment is received for the reservation signaling [2, 6]. Thus, such an approach would be useful for bursty data applications such as FTP in which the data can be easily kept in the source. This approach would not be useful for bursty real-time application, e.g., variable bit rate video, which (for storage reasons) cannot tolerate long waiting times for a reservation process to complete.

In this work we suggest an algorithm that increases the probability to successfully transfer bursty traffic by adding flexibility to the burst routing. We assume that bursty applications reserve no bandwidth during their set-up process. Instead, bandwidth is requested for each burst separately (with either on-the-fly or traditional fast reservation algorithms) and is freed immediately after the burst transmission. The suggested bypass algorithm is simple enough to be implemented by cheap hardware. Before proceeding, we describe two routing approaches for high speed networks with

[*] Also with the Department of Electrical Engineering, Technion, Haifa, Israel.

[†] This work was supported in part by the Technion V.P.R. fund and the Fund for the Promotion of Research at the Technion. Also with Sun Microsystems, Mountain View, CA 94043-1100.

which our algorithm can be used: source routing and ATM.

Source routing, or *Automatic Network Routing (ANR)* [5], is a routing method where each packet carries in it the entire route it should traverse. In our discussion, we will assume that the route is placed in the header as a list of port-IDs (or link-IDs), and each node along the packet route strips the ID it uses from the head of the list (in practice, there are other methods for handling the source route that only differ in technicalities and can be integrated with our algorithm [5]). In networks that employ source routing, the route for the session is computed at the source node using data that is distributed by a *topology update* algorithm. It is therefore plausible that routes thus computed are not optimal (and may not even be feasible). Changing the route on-the-fly amounts to modifying the source route in the packet's header.

In ATM networks, cells travel along Virtual Circuits (VCs) that are constructed by a concatenation of Virtual Paths (VPs). The VC and VP identifiers are written in the cell header and possibly swapped in every switch. Tables in the switches are used to determine the route based on local identifiers [1]. For our purpose it is important to note that the routing information is distributed in the switches along the path the cells traverse. Modifying a cell's route on the fly requires changing the routing information in several switches – an operation that is neither simple nor fast [9]. In particular, buffering requirements for the cells while a new route is created makes on-the-fly rerouting look impractical.

The algorithm we suggest in this work increases the probability of a successful short burst transmission or the probability of a successful reservation for longer bursts by using local route-deflections. Because the route is determined based on somewhat inaccurate data, and because a proper reservation process is not undertaken, it is possible that the determined route may actually not be able to accommodate the bandwidth of the burst. To overcome this possible lack of bandwidth local route deflections are constructed. To use these deflections our algorithm uses load information from the immediate neighboring nodes. This does not require dissemination of large volumes of load-data across the network, keeps the information fairly up to date, and increases the probability of reservation success.

Most high speed networks are constructed as an interconnection of specially constructed packet switches. Unlike traditional switches these switches must support extremely fast streams of packets (or small cells in the case of ATM) meaning that switching speed is very high and implying that the use of a software operated general-purpose processor is out of the question. A typical switch is constructed as an interconnection of *link processors* (LPs) each supporting a single link [5, 4]. The routing of packets that arrive at the input links is done directly by these LPs. Only packets that require more complex processing (e.g., control packets) are forwarded to a more sophisticated control unit. Naturally, the suggested algorithm is designed to be performed by the LPs.

The rest of the paper is organized as follows. In section 2 we describe the fast bypass algorithm for networks that employ source routing. Next we describe in section 3 the fast bypass algorithm for ATM networks. In section 4 we analyze the performance of the algorithm in terms of reservation success probability, and in section 5 we give our concluding remarks.

2 A Fast Bypass Algorithm for Networks with Source Routing

The LPs in a switch share the routing tables that are used for the presented algorithm. Thus, it is convenient to treat the switch LPs (the hardware) and the controlling algorithms as a single abstract entity, the *node*. The algorithm we present in this section works by exchanging load information in a close locality of every node. To that end, we make use of the following definitions. The *2-neighborhood* of a node is the set of nodes that are at most two hops away. A *local segment* is a single link or a concatenation of two links leading from a node to another node in its 2-neighborhood. A local segment is typically a part of a longer path between source and destination nodes. Note that every local segment uniquely identifies a node, but several local segments may identify the same node. The *local segment group* is the collection of all the local segments that identify the same node. We use (l_1, l_2) to signify a two-hop local segment comprised of links l_1 and l_2 (in that order), and (l_1) for a one-hop local segment. The bypass algorithm will, in congested situations, replace one local segment with another.

The Bypass Algorithm, BA, is based on frequent load measurements (typically the load indicator is the sum of the total reserved bandwidth and the average number of buffers occupied by nonreserved traffic) of the outgoing links at every node, and on sharing this information with the immediate neighbors. This way every node has updated knowledge of the load on all the local segments emanating from it. This informa-

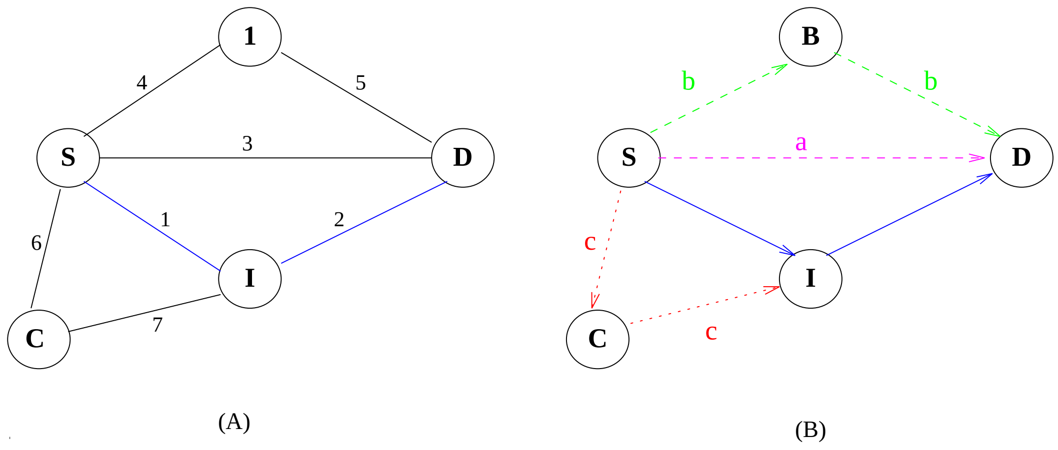


Figure 1: An example network with the three bypass types

tion is maintained in a local *RoutingTable* that has an entry for each local segment. Each entry in the *RoutingTable* contains a field that indicates the availability of the local segment, and the preferred alternate local segment should the original one be blocked (see below). *RoutingTable* size is quadratic in the output degree of the nodes, and in most practical networks is not expected to have more than a few tens of entries.

Figure 1A shows an example of a five node network. The routing table of node S has 10 entries for the following local segments: (4) and $(3,5)$ to node B; $(4,5)$, (3), and $(1,2)$ to node D; (1), $(3,2)$, and $(6,7)$ to node I; and (6) and $(1,7)$ to node C. Suppose a packet arrives to node S with the route $(1,2)$ written in its header and finds link 1 blocked. Three types of bypasses are possible for this packet (see figure 1B):

a) the direct link to D (a shortcut) that avoids both links 1 and 2,

b) a two-link bypass via node B (identical length) that again avoids links 1 and 2, or

c) a two-link bypass of link 1 to node I (a long bypass) that is followed by link 2.

These types are the only ones considered in this algorithm.

When a packet arrives at a switch, its entry in *RoutingTable* is examined to check if the local segment it should traverse is not blocked. If it is blocked *RoutingTable* is first searched for a bypass that does not increase the length (types a and b) and then for one that bypasses only this link (bypass type c). If the search succeeds the new local segment replaces the original one in the packet header and the packet is forwarded along the deflected route; otherwise, the packet is discarded.

To maintain *RoutingTable*, we keep for every entry the load of the preferred route. An entry in *RoutingTable* is updated when one of the following occurs:

- The load on the preferred route is changed.
- A bypass route with more residual bandwidth (lower load) than the preferred route is found.

Link failures are treated as a maximal decrease in the available bandwidth as will be explained in section 2.1.

To expedite the processing of regular packets *RoutingTable* is sorted according to the local segments. This, however, poses an update problem since there may be several local segments that identify the same node and it is natural to maintain their bypass information simultaneously. To allow simultaneous updates of all the entries in *RoutingTable* that refer to the same node, a second table, *HostTable*, is used. *HostTable* is sorted by node-IDs[1] with a single entry for every node in the 2-neighborhood. Each entry contains the node-ID and a list of all the local segments that identify this node, i.e., its local segment group. *HostTable* can be initialized either when the network is started or can be built by a topology update algorithm [11].

2.1 A Detailed Description

Two tables are maintained and used by the algorithm: *RoutingTable* has an entry for each local segment that comprises four fields:

[1] Node IDs can be global or locally assigned by higher level algorithms.

- The id of the node at the end of the local segment.
- The load of the local segment.
- The preferred alternative local segment.
- The load of the preferred alternative local segment.

The maintenance of this table is described below. *HostTable* has an entry for each node in the 2-neighborhood that lists all the local segments in the *local segment group* of the node.

Every node periodically measures the loads on the links that emanate from it. The way these measurements are made is out of the scope of this paper. For our purpose, it is enough to assume that the resulting load indicator is based on both reserved and non-reserved traffic. The load indicators are sent to the immediate neighbors and are locally used to update *RoutingTable* as follows. For every emanating link, the load entry of the one-hop local segment is updated. If the direct link is the preferred local segment then the preferred local segment load field in the entries of its l ocal segment group are updated. If its load is lighter than the load of the preferred local segment of its local segment group then the local segment is written in the preferred local segment field and its load is written in the preferred local segment load field in the entries of all the members of the local segment group. The members of the group are easily located with *HostTable*.

The measurements are sent to the neighbors as a list of number pairs, a link id and its load. For every link in the list, the load of the corresponding local segment is updated. If this local segment is the preferred local segment the load field of the preferred route in the entries of all the members in the local segment group is updated. If the local segment load is lighter than the one of the preferred local segment then the first becomes the preferred local segment and the entries of the local segment group are changed to reflect this, i.e., the local segment is written in the preferred local segment field and its load is written in the preferred local segment load field.

A failure in a link that is not directly connected to a node is treated as if the available bandwidth of this link dropped to zero, and can be reported by sending a measurement list. A failure in a link adjacent to a node requires a pass through the entire *RoutingTable* (typically, few tens of entries) to search for all the entries that have this link as part of their preferred local segment, and then to update their load to the maximum, so that every new measurement of a different local segment will update it. This process is not efficient, but is used only in the rare event of link failure and only in the two nodes at the ends of the failed link.

2.2 Avoiding Loops

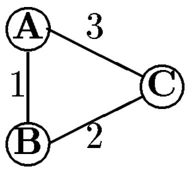


Figure 2: An example of a bypass loop

Deflection routing in general and BA in particular may cause a packet to cycle in loops. We next demonstrate how BA can cause a message to travel in a two link loop. Consider the network of Figure 2 and suppose a packet reaches node A and is trying to reach node C through link 3. At the time the packet arrives at node A, the buffers of link 3 are all full, there are free buffers in front of link 1, and *RoutingTable* indicates that the preferred bypass for the local segment (3) is the local segment $(1,2)$. The packet is, thus, sent via the bypass $(1,2)$. Suppose the packet is somewhat delayed in the queue and when it reaches node B link 2 has no free buffers but according to *RoutingTable* the preferred bypass to segment (2) is $(1,3)$ since links 1 and 3 are not totally full now. As a result the packet is deflected back to node A.

On the one hand, going in circles or busy waiting, can be considered as a good solution since instead of discarding the packet we use the network as storage. On the other hand, if the buffers of the switches are all almost full we might create a livelock where messages travel around and never reach their destinations, or do so after consuming too much network resources. To disable routing loops, a bit in the packet header can be set the first time the packet is deflected and if a second deflection is needed the packet is discarded. If more routing flexibility is needed a few bits can be allocated in the packet header to bound the number of deflections above one. Two or more allowed deflection may theoretically cause a packet to go in cycles, but in practice, the probability for this is small if the number of allowed bypasses is kept small. The analysis in section 4 shows that allowing only one deflection significantly decreases the rejection probability, while the residual contribution of additional deflections to the success probability of the packet decreases for every additional allowed bypass.

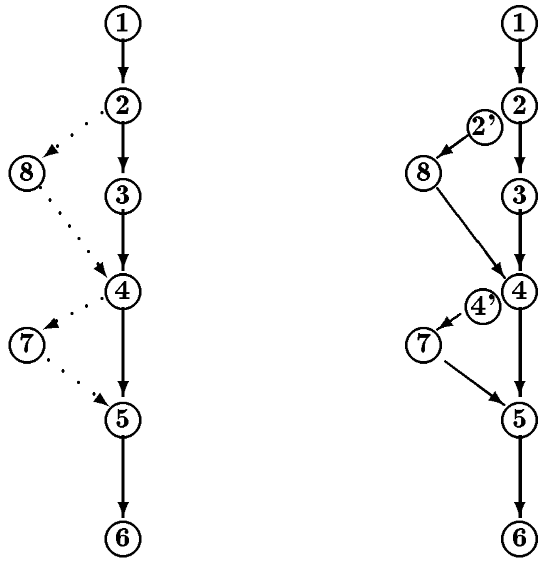
3 A Fast Bypass Algorithm for ATM Networks

In this section we adapt the bypass algorithm presented in the previous section to ATM networks. In networks that employ source routing, once we identify a blocked link and know about a local segment that bypasses it, we can deflect the message by changing the routing information in its header. In ATM networks the routing information is not carried in the cells but is scattered in the switches along the path it traverses. Changing the routing information in several switches to create a deflection route is therefore neither simple nor fast [9]. In particular, buffering requirements for storing the cells while the new route is created make on-the-fly deflection look impractical.

We suggest, instead, to preprepare for congestion when a VP is constructed. Upon a VP construction, loaded areas are identified and bypass routes are created to be used when the primary route is blocked. Each VP_i is assigned two VPIs: VPI_i for the primary route and VPI'_i for the bypass route. In figure 3A the primary VP is drawn with solid lines and the bypass routes with dotted lines. In the same way, bypass routes can be preprepared per VC in an ATM network that is constructed of VC switches without the use of VPs, but then the relative cost of the preparation for a single VC is higher.

As already stated, bursty VCs are brought up with no bandwidth reservation. We shall concentrate here on applications that use fast reservation algorithms whenever there is a burst to transmit [2], on-the-fly reservation [12] is shortly discussed in section 5. In ATM networks, fast reservation algorithms use monocell messages that traverse the VC route, reserve bandwidth in one direction, and acknowledge/reject a reservation request in the reverse direction. Since ATM VCs are unidirectional, the reverse direction does not necessarily use the same physical links.

We now show how reservation cells are deflected to create bypass VCs, and how the switches identify and route cells that belong to bypass VCs. We term the switches where a bypass starts (switches 2 and 4 in figure 3A) *junction switches.* A reservation cell starts its way on the VP with the primary VPI. Non-junction switches do not participate in the deflection process. When a reservation cell arrives at a junction switch that is unable to fulfill the burst request for bandwidth in the primary route it tries to reserve bandwidth in the first link of the bypass. Upon a success, the reservation cell is forwarded to the bypass link and its VPI is set to the secondary VPI. In addition, the junction switch registers the VC in a separate table, *BypassTable*. The values registered in *BypassTable* are the secondary VPI (cells of this burst will use it as their VPI), the VCI, and the rest of ATM switching data, i.e., the local VPI in the next hop and the output port-ID. A bandwidth release cell that follows the end of the burst causes the deletion of the corresponding entry from *BypassTable*.



A. A VP with bypass routes. **B**. A tree representation of A.

Figure 3: A VP with bypass routes and its tree representation

Usually, the arrival of a reservation cell to the switch at the end of the VP triggers the transmission of an *Ack* to the VC-switch at the beginning of the VP. Similarly, the successful arrival of a reservation cell with the secondary VPI to the end of the VP triggers the transmission of a similar message, Ack', signaling the local source that the cells of this burst should be switched through the secondary VPI (or, if the source is the origin of the cells, it should initiate the cells with the secondary VPI). When a data cell arrives with the secondary VPI at a junction switch, *BypassTable* is searched and if a match is found the cell is routed according to the data in the table. Otherwise, if no match is found in *BypassTable* the cell is routed according to the ATM switching table.

The suggested bypass algorithm offers 2^m potential routes (where m is the number of junction switches) for the price of only two VP identifiers and less than four switching table entries in every switch. The switching information to the next switch in the primary route is saved in two entries: one for the primary VPI and one for the secondary VPI. The two entries have identical switching information. If a VC reserves bandwidth for a burst in a bypass route, an entry in *BypassTable* keeps the switching information of the deflected route. This entry is created only in the junction switches where the VC is deflected, and is deleted when the bandwidth is freed. In the switches of the bypass routes one entry for the secondary VPI is kept in the regular switching tables. There is no need for an entry for the primary VPI.

In practice, a network manager might wish to bound the number of bypasses to keep the stretch factor, i.e., the ratio between the original VP length (in hops) and the length of the VC with the bypasses, low. A small counter in the reservation cell can be used to implement any practical bound, and in particular one bit can be used to allow only one bypass.

Examining the VP with the bypasses of figure 3A one can easily identify a tree rooted at the destination as depicted in figure 3B. This suggests an alternative way to look at the bypass scheme: instead of building a shoelace VP, we build a VP with a tree structure, such that every leaf except the VP entry point must also be an internal node of another branch of the tree. An interesting extension to this algorithm will enable to bypass a bypassed route, e.g., in figure 3 if the link between switches 7 and 5 is loaded one may wish to use a direct link between switches 7 and 6. Other uses of tree shaped VPs can be found in [8].

4 Analysis

In this section we compute the improvement in the reservation success probability when our algorithm is used in several networks with regular structure. Throughout the analysis we assume that the probability to succeed in reserving bandwidth on a link is p for all the links, and this probability is independent for every link. The independence is justified by the fact that a burst is mainly competing against other applications that have constant bit rate, and not against other bursts. We assume that the original routes (VPs in the case of ATM) are all shortest path routes.

The way the algorithm is implemented impacts its performance. For source routing, we suggested in section 2 that the availability of the local segment will be checked at every switch, and that if no local segment with sufficient bandwidth is found the burst (or reservation cell) will be discarded. For the case where only the second hop is blocked and especially in the case of fast reservation algorithms when a reservation cell is sent, it might be better to forward the reservation request in the hope that a bypass will be found. The implementation of this variant might increase the switch complexity and cost. Another variant is to check *RoutingTable* only if the burst can not be forwarded, which implys no extra handling for the bursts if the route in not loaded. The performance of this implementation is the worst since it does not allow a bypass from a bypass route. We choose this variant for the analysis of this section.

For ATM, we suggested in section 3 to check in every junction switch the local segment, and to deflect the burst if the local segment is blocked. In the analysis of this section we assume, as for the source routing model, the less efficient implementation where only if the first hop in the local segment is blocked a bypass is searched.

Hypercubes

In hypercubes every two-hop route has exactly one two-hop bypass, every h-hop route can be bypassed in $h-1$ points, and none of the bypass routes share links. It is clear that the success probability of a reservation along an h hop route is p^h. If we allow only one bypass for a burst route, allow every link to be bypassed, and build the VP-tree to contain $h-1$ bypass routes (in the ATM model) the success probability of the reservation grows to

$$p^h[1+(h-1)(1-p)] \tag{1}$$

in both network models. If we do not limit the number of bypass the success probability along an h-hop route, $S(h)$, is given by the recurrence

$$\begin{aligned} S(0) &= 1 && (2)\\ S(1) &= p && (3)\\ S(h) &= pS(h-1)+(1-p)p^2S(h-2) && (4) \end{aligned}$$

The solution of this recurrence (see [10]) gives the expression for $S(h)$, $h \geq 2$

$$\begin{aligned} S(h) &= \frac{1}{2}\left(1+\frac{1}{\sqrt{5-4p}}\right)\left[\frac{p(1+\sqrt{5-4p})}{2}\right]^h + \\ &\quad \frac{1}{2}\left(1-\frac{1}{\sqrt{5-4p}}\right)\left[\frac{p(1-\sqrt{5-4p})}{2}\right]^h \end{aligned} \tag{5}$$

Figure 4 compares the probability for a burst to succeed in reserving bandwidth along its route as a function of the probability to find enough bandwidth

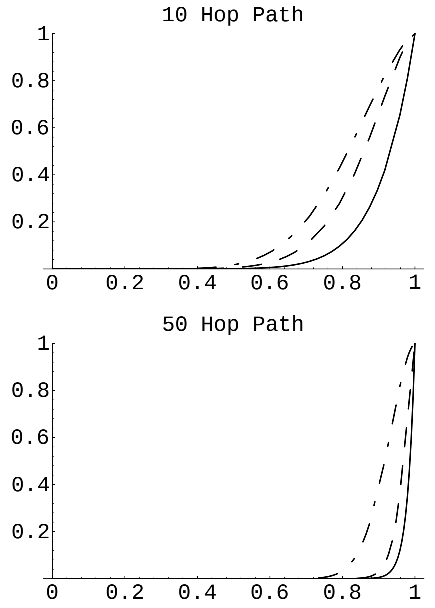


Figure 4: The effect of the Bypass Algorithm on the success probability of bursts as a function of the link load

on a link. The solid line depicts the success probability without bypasses, the dashed line depicts the case when only one bypass is allowed, and the dashed-dotted line depicts the case when there is no restriction on the number of bypasses. It is clear from these graphs (and others omitted due to space restrictions) that the effectiveness of the bypass algorithm grows with the length of the VP. Note also that even when only one bypass is allowed the improvement is significant. In fact, it is shown in [7] that the contribution of the first bypass is the most significant while the improvement of the third bypass and on is negligible. This may lead us to implement the algorithm with the one bypass option which is the easiest and most efficient implementation.

Triangulated Graphs

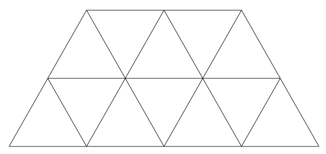


Figure 5: A lattice of triangles

A planar graph where every region is bounded by a circuit of three edges is said to be *triangulated* [3]. Consider, first, a lattice of triangles (figure 5). A shortest path route in a triangular lattice has no 60° turns. The probability that a two-hop segment is congested is $1 - p^2$.

We look, first, at networks that employ source routing. If there are no turns in a path, a congested link can be bypassed by one of two possible two-link type c bypasses with probability $1 - (1 - p^2)^2$. If there is a 120° turn in the path the bypass probability is higher since an additional type b bypass can be found. When only one bypass is allowed the success probability of reservation along an h hop route is at least

$$p^h \left(1 + h(1-p)[1 - (1-p^2)^2]/p\right) \tag{6}$$

We assume that in an ATM network only one bypass route is prepared per link. The success probability under this assumption when only one bypass is allowed is at least

$$p^h \left(1 + h(1-p)p\right) \tag{7}$$

If K bypasses are allowed the success probability is

$$p^h \sum_{k=0}^{K} \binom{h}{k} (1-p)^k p^k \tag{8}$$

For $K = h$ we get

$$p^h \left(1 + p - p^2\right)^h \tag{9}$$

Note that the success probability of the algorithm for source routing is higher than for ATM networks since in the later we prepare only one bypass per link.

For general triangulated networks, let h be the number of links in a path of length H that are part of a triangle. Note that since we consider only shortest path routes, no two links in a route belong to the same triangle. Based on our previous results, we can write the following lower bounds for the success probability when only one bypass is allowed (in the worst case there is only one type c bypass for each of the h links in both network models)

$$p^H \left(1 + h(1-p)p\right) \tag{10}$$

If K bypasses are allowed the success probability is

$$p^H \sum_{k=0}^{K} \binom{h}{k} (1-p)^k p^k \tag{11}$$

For $K = h$ we get

$$p^H \left(1 + p - p^2\right)^h \tag{12}$$

5 Concluding Remarks

The algorithm presented here can be used in both models, ATM and source routing, for short bursts that may use on-the-fly reservation and for longer bursts that use fast reservation algorithms. For bursts that use fast reservation, our algorithm adheres to the reservation principle of the network whether it is ATM based or source routing type. For non-reserved traffic, one should be cautious not to allow this traffic to disturb the reserved traffic. This can be achieved by setting a low priority bit of the non-reserved traffic cells (in ATM the CLP bit; in the ANR model, such a bit should be allocated in the packet header).

Deflection of short bursts that do not reserve bandwidth before transmission in ATM networks can be done in two distinct ways: with or without reservation on-the-fly. If on-the-fly reservation is used each burst is preceded by a reservation cell and succeeded by a release cell. The cells are sent with VPI', the secondary VPI, and are treated as described in section 3 for bursts that use reservation, i.e., only if no bandwidth is available in either route the burst is discarded. If no reservation is used, non-reserved cells should be identified and switched to the primary or the secondary route regardless of their VC. This might cause a burst to have its head go in one route while its tail goes in the other. However, the probability for this happening is low since the time between switching from route to route is long with respect to the short burst length. Another possibility is to switch the route only after an end-of-burst is detected. Further studies are needed to investigate the implementation of the last two methods.

One of the important merits of our algorithm is the ability to implement it with simple hardware without adversely effecting performance. In [7] we describe a possible implementation of our algorithm for a source routing based network. This implementation does not add delay to packets that are not deflected, and adds only a small delay (a few byte transmission time) to the ones that are deflected.

References

[1] Jean-Yves Le Boudec. The asynchronous transfer mode: a tutorial. *Computer Networks and ISDN Systems*, 24:279 – 309, 1992.

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

[3] Robert G. Busacker and Thomas L. Saaty. *Finite Graphs and Networks: an introduction with applications.* McGraw-Hill, 1965.

[4] I. Cidon, I. Gopal, P. M. Gopal, R. Guérin, J. Jannielo, and M. Kaplan. The plaNET/Orbit high speed network. Technical Report RC-18270, IBM, T. J. Watson Research Center, Yorktown Heights, NY, March 1993.

[5] Israel Cidon and Inder Gopal. PARIS: an approach to integrated high-speed private networks. *International Journal of Digital and Analog Cabled Systems*, 1(2):77 – 86, April-June 1988.

[6] Israel Cidon, Inder Gopal, and Adrian Segall. Fast connection establishment in high speed networks. In *ACM SIGCOM'90*, pages 287 – 296, 1990.

[7] Israel Cidon, Raphael Rom, and Yuval Shavitt. Fast bypass algorithms for high-speed networks. Technical Report EE PUB No. 924, Technion - Israel Institute of Technology, June 1994.

[8] Reuven Cohen, Baiju Patel, Frank Schaffa, and Marc Willebeek-LeMail. The sink tree paradigm: Connectionless traffic support on ATM LANs. In *INFOCOM'94*, pages 821 – 828, June 1994.

[9] Reuven Cohen and Adrian Segall. Connection management and rerouting in ATM networks. In *INFOCOM'94*, pages 184 – 191, June 1994.

[10] Daniel H. Greene and Donald E. Knuth. *Mathematics for the Analysis of Algorithms.* Birkhauser, second edition, 1982.

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

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