Bandwidth reservation for bursty traffic in the presence of resource availability uncertainty

Israel Cidon *Raphael Rom*
Dept. of Electrical Engineering
Technion, Haifa, Israel
e-mail: {cidon, rom}@ee.technion.ac.il

Yuval Shavitt[*]
Bell Laboratories, Lucent Technologies
Holmdel, NJ 07733-3030
e-mail: shavitt@ieee.org

Abstract

In this work we suggest algorithms that increase the reservation success probability for bursty traffic in high speed networks by adding flexibility to the construction of the routes. These algorithms are simple enough to be implemented by cheap hardware. They cause 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 algorithms have a minimal communication overhead, due to the local nature of their 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 [Tur92] 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

[*] Corresponding author. Part of this author work was done while he was with the Technion - I.I.T., Haifa, Israel.

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 [BT92, CGS90]. 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 algorithms that increase 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 algorithms are simple enough to be implemented by cheap hardware. Before proceeding we describe two routing approaches for high speed networks with which our algorithms can be used: source routing and ATM.

Source routing, or *Automatic Network Routing (ANR)* [CG88], 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 [CG88]). 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 therefor 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 [Bou92]. 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 [CS94]. In particular, buffering requirements for the cells while a new route is created makes on-the-fly rerouting look impractical.

The algorithms we suggest in this work increase 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 algorithms use 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.

Deflection routing was suggested almost since the beginning of the research on distributed computing. Baran [Bar64] suggested the Hot-Potato heuristic routing where at each switch the routing table stores a list of outgoing links per destination. The list is sorted in decreasing cost order where the first link is the preferred for routing the second one is used should the first is blocked, and the i-th link is used if the first $i-1$ links are not available. Later works [GH92], suggested to use arbitrary link should the preferred link fail and thus simplify the maintenance of the routing table. However all the works on WANs [Bar64, FK71, Rud76] relies on global knowledge that should be distributed in the entire network that maybe combined with local knowledge of the node's queue length [FK71, Rud76]. Our proposal takes these idea one step ahead by allowing nodes to examine the two-hop locale. Deflection routing was also suggested for local area networks (LANs), multihop lightwave networks and for interconnection networks and switching fabrics. In these architectures, deflection decisions are easy to make since the network topology has a simple regular structure [Max87, Uru91, AS91, KS91], or since a sense of direction exists [OY90]. Our scheme works for general topologies and thus cannot rely on the simple decision rules.

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 small cells 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 [Tob90] of *port processors* (PPs) each supporting a single link [CG88, CGG$^+$93]. The routing of packets that arrive at the input links is done directly by these PPs. Only packets that require more complex processing (e.g., control packets) are forwarded to a more sophisticated control unit. Naturally, the suggested algorithms are designed to be performed by the PPs.

The rest of the paper is organized as follows. In section 2 we describe a fast bypass algorithm for networks that employ source routing. Next we describe in section 3 a fast bypass algorithm for ATM networks. Then we demonstrate the algorithm performance by analysis and simulation. In section 4, we analyze the algorithm performance in terms of reservation success probability by assuming independence in the success probability. The independence is justified by the fact that a burst is mainly competing against other applications that have constant bit rate. To show the interaction among bursts we report simulation results in section 5. Finally, we conclude with remarks about how the algorithm can be applied to other network models.

2 A Fast Bypass Algorithm for Networks with Source Routing

The PPs in a switch share the routing tables that are used for the presented algorithm. Thus, it is convenient to treat the switch PPs (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 non-reserved 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 information 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 reservation packet for a burst arrives to node S with the route $(1,2)$ written in its header and it cannot reserve sufficient bandwidth on link 1. Three types of bypasses are possible for this reservation 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. To gain more flexibility and deflection opportunities, *RoutingTable* is checked to ensure the ability to reserve bandwidth along the two hops of the local segment, even when the first hop has sufficient bandwidth. If the local segment is blocked *RoutingTable* is first searched for a bypass that does not increase the route length (types a and b) and then, if the first link in the local segment is blocked, for one that bypasses only the first link (bypass type c). If the search succeeds the new local segment replaces the original one in the reservation packet header and the packet is forwarded along the deflected route; otherwise, a negative acknowledge is sent to the source.

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 reservation 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 [Seg83].

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

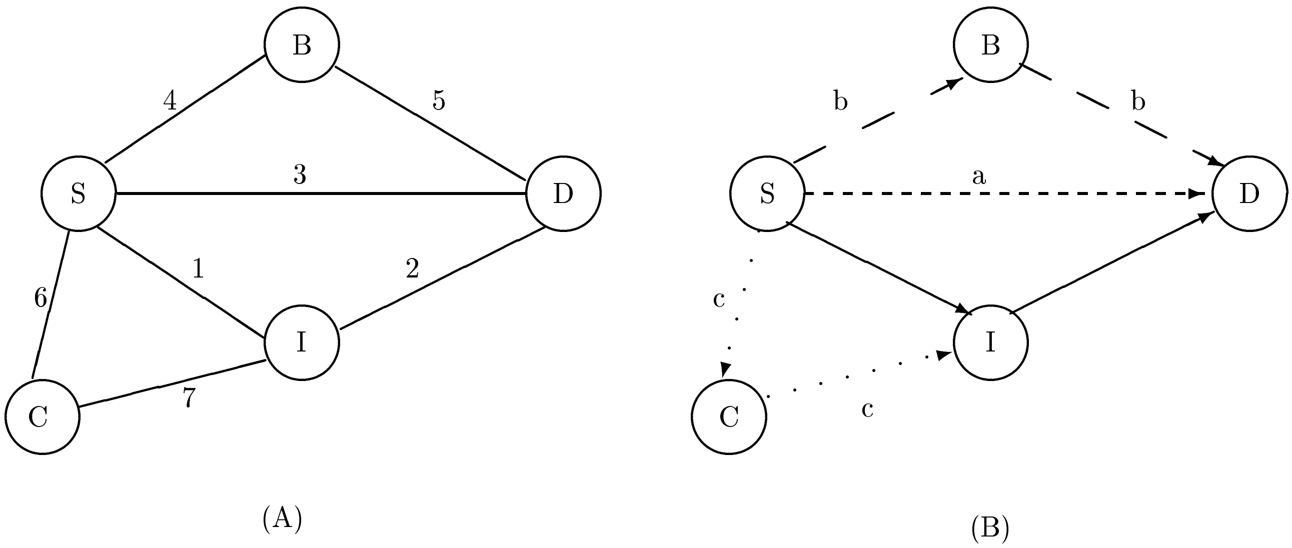


Figure 1: An example network with the three bypass types

2.1 A Detailed Description

In this section, we describe in details how the algorithm data tables are maintained. Then we discuss how the algorithm works for long bursts that use fast reservation, i.e., when prior to the burst transmission a reservation packet is sent to reserve bandwidth for the burst. The applicability for short bursts is discussed in the end of the section.

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

- 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 emanates 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 route then the preferred load field in the entries of its local segment group are updated. If its load is lighter than the one of the preferred route of its local segment group then the local segment is written in the preferred route field and its load is written in the preferred route 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.

For long bursts a reservation packet is sent prior to the burst transmission [BT92]. This packet carries a list of the port ids it should traverse. In addition, while the reservation packet traverses the network, a backward route is built and stored in the packet. When a reservation packet arrives to the switch, the local segment entry in *RoutingTable* is checked to see if sufficient bandwidth is available. The following cases are possible:

- There is sufficient capacity for the burst along the original local segment — Reservation is made in the link emanating from the switch, and the reservation packet is forwarded to the next hop.
- There is insufficient capacity for the burst along the original local segment but a type a- or type b bypass with sufficient capacity exists — The preferred local segment from *RoutingTable* is inserted to the reservation packet header instead of the original local segment, reservation is made in the first link in the preferred local segment, and the reservation packet is forwarded to the next hop.
- There is insufficient capacity for the burst along the original local segment and only a type c bypass with sufficient capacity exists — If the first link in the local segment has sufficient capacity the reservation is treated as if sufficient capacity exists for the entire route, if the first does not have sufficient capacity the preferred local segment for this link is used as described for the type a- and type b bypasses.
- There is insufficient capacity for the burst along the original local segment and no bypass with sufficient capacity exists — A negative acknowledgment is sent to the source using the backward route that is stored in the reservation packet, the negative acknowledgment carries the route the reservation packet traversed before it was blocked to allow the release of the bandwidth from the part where it was allocated.

When the reservation packet arrives at the destination node, A positive acknowledgment is sent to the source using the backward route that is stores in the reservation packet. The positive acknowledgment carries the route the reservation packet traversed which is put by the source in the header of all the packets in the burst. This ensures that all the burst packets travel along the route where reservation was made.

Short Bursts

As explained in the introduction, short bursts are sent without reservation, or with on-the-fly reservation packet that precede them. The direct use of on-the-fly reservation scheme with our deflection mechanism is not possible for source routing networks since the switches do not hold tables about the connections where the reservation and the routing information can be stored. Thus we suggest two alternatives: 1) to deflect each packet of a short burst in a way that will try to route all the packets together, and 2) to add tables to the source routing switches.

Next, we describe how to deflect each of the data packets of short bursts according to $RoutingTable$. When a packet arrives at a switch it is routed according to the preferred route in $RoutingTable$ regardless of the availability of buffer space (the "bandwidth" of the non-reserved traffic) in the original route. The mandatory use of $RoutingTable$ ensures that, in case of marginal buffer occupancy, i.e., when the buffer pool is full and the input and output rates to the buffer are almost equal, packets of a burst will not be spread between the original route and the preferred route. The changes of the preferred route entries are not frequent, thus, the chances that a short burst will be sent over different routes are small. The probability of packets arriving out of order can be kept smaller by using hysteresis function to change between routes in $RoutingTable$. Note, that even in case where the packets arrive out of order, many transport protocols, e.g., TCP, are capable of handling them correctly. The routing along the preferred route in $RoutingTable$ is also useful in directing the non-reserved traffic to the less condensed links of the network, leaving more free bandwidth in the highly used links, and thus lowering the reject probability and the delay along these links.

Although the probability of packets arriving out of order is small, we suggest here another algorithm adaptation for the case where it is not acceptable. For this end we need a $BypassTable$ to be maintained in the switches; the table contains an entry for each bursty connection that uses short bursts with on-the-fly reservation. When such a connection is established an entry is opened in $BypassTable$, this entry will hold the selected local segment for the burst. When an on-the-fly reservation packet arrives at the switch, reservation is attempted for the current preferred local segment taken from $RoutingTable$. In case of success, the preferred local segment is written in $BypassTable$; this local segment is written to the header of each data packet of this burst. The burst is succeeded by a release packet that releases the reserved bandwidth. If reservation cannot be made, a negative acknowledgment is sent back to the source, and the $BypassTable$ is set to a null value that cause the discarding of the burst packets.

2.2 Avoiding Loops

Deflection routing in general and BA in particular may cause a packet to cycle in loops. We next demonstrate how BA can cause a packet 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

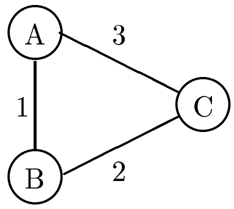


Figure 2: An example of a bypass loop

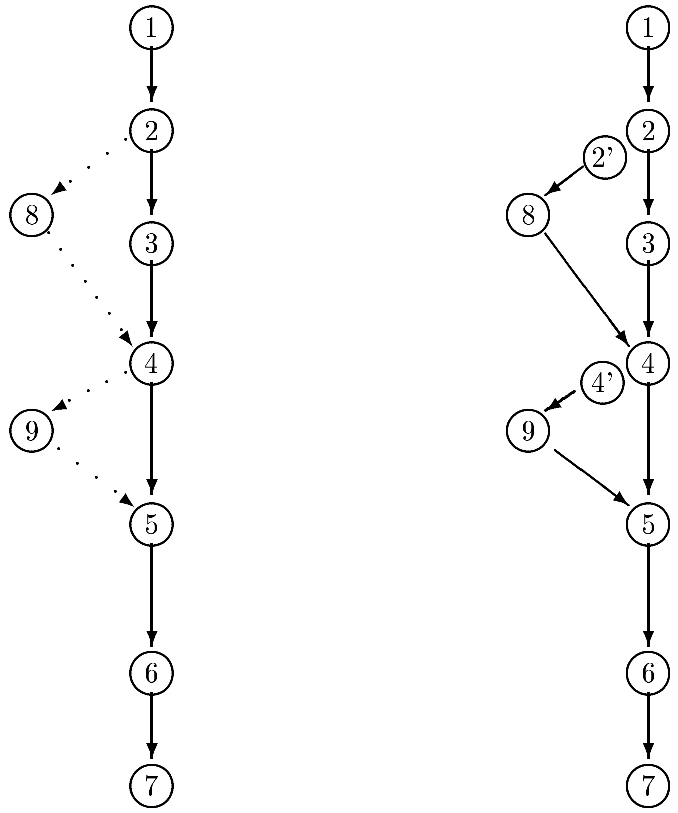
the queue and when it reaches node B link 2 has no free buffers but according to *RoutingTable* the preferred bypass to segment (2) is $(1, 3)$ since links 1 and 3 are not totally full now. As a result the packet is deflected back to node A.

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 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 bypasses is kept small. The analysis in section 4 shows that allowing only one deflection significantly decrease the rejection probability, while the residual contribution of additional deflections to the success probability of the packet decrease for every additional allowed bypass.

3 A Fast Bypass Algorithm for ATM Networks

In this section we first adapt the bypass algorithm presented in the previous section to long bursts in ATM networks, the adaptation to short bursts is discussed in the end of this section. 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 [CS94]. 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. As already stated, bursty VCs are brought up with no bandwidth reservation, and use a fast reservation algorithm whenever there is a burst to transmit [BT92]. In ATM networks, fast reservation algorithm 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.



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

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

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 cause the deletion of the corresponding entry from $BypassTable$.

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 the bypassing of a bypassed route, e.g., in figure 3 if the link between switches 9 and 5 is loaded one may wish to use a direct link between switches 9 and 6. Other uses of tree shaped VPs can be found in [CPSWL96].

Short Bursts

In ATM networks, unlike with networks that employ source routing, we can use on-the-fly reservation for short bursts to ensure that all the cells of a short burst use the same route. To this aim, we prepare for each VC in the set-up procedure an entry in $BypassTable$ that points to the primary VP link. When a short burst is sent it is preceded by a reservation cell and followed by a release cell. On the arrival of the reservation cell the PP set $BypassTable$ according to the available bandwidth in the primary and secondary routes. The data cells that are sent always with the secondary VPI are routed according to the information in $BypassTable$.

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. This independence assumption is a standard assumption used in the literature (see [Wid95] and the references therein). 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. Even in the case when bursts are competing against each others, we showed by analysis and simulations [CRS96] that after smoothening at the network entrance, the success probability for bursts is almost constant. 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. If no local segment with sufficient bandwidth is found the burst (or reservation cell) can be discarded. For the case where only the second hop is blocked and especially in the case of fast reservation algorithm when a reservation cell is sent, we suggest to forward the reservation request in the hope that a bypass will be found. The suggested implementation increase the reservation success probability at the price of increased switch complexity and cost. Another variant is to check $RoutingTable$ only if the burst can not be forwarded, which implies 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 VP 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 $h-1$ ($h \geq 1$) 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 [GK82]) gives the expression for $S(h)$, $h \geq 2$

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

Triangulated Graphs

A planar graph where every region is bounded by a circuit of three edges is said to be *triangulated* [BS65]. Consider, first, a lattice of triangles (figure 4). 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 $1-(1-p^2)^2$.

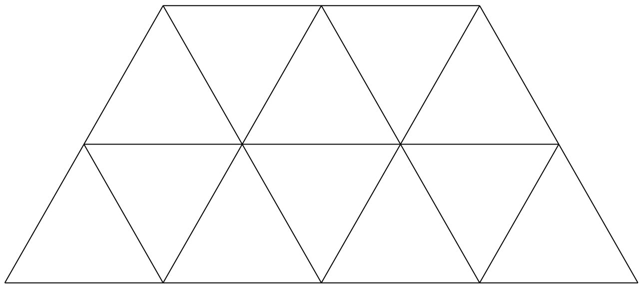


Figure 4: A lattice of triangles

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 given by summing all the possible combinations to have K or less reservation failures along the path:

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

For $K = h$ we get

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

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 VPs, no two links in a VP 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)

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

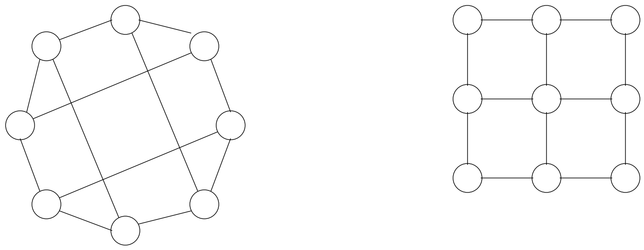
If K bypasses are allowed the success probability is, as above:

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

For $K = h$ we get

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

Grids and Chordal Rings



A. A chordal ring B. A grid

Figure 5: (A) a chordal ring with a three node chord and (B) a 3X3 grid.

A chordal ring (see figure 5A) is a ring structured network with an even number of nodes in which each node has an additional link, called a chord, to some other node in the network [AL81]. We shall term the links that connect a node to its ring neighbors *neighbor links*. A point in a route where a packet is switched from a neighbor link to a chord or vice versa is called a *turn*. In a grid network (see figure 5B) a turn in a route is a point where a packet is switched from a horizontal link to a vertical link or vice versa. In the two topologies, bypasses are possible only around the two links that are connected to a turn. Let the number of turns be m and let the number of links in a path be H.

When only one bypass is allowed, the success probability is

$$p^H \left(1 + m(1-p)\right) \tag{13}$$

for both network models. If K bypasses are allowed the success probability is

$$p^H \sum_{k=0}^{K} \binom{m}{k} (1-p)^k \tag{14}$$

if the turns are at least three hops apart. For $K = m$ we get

$$p^H \left(2-p\right)^m \tag{15}$$

5 Interaction Among Bursty Connections

This section complements the previous section by examining via simulation the mutual interaction among bursty connections. To isolate this effect the simulations test the mutual interaction among two identical bursty connections that use the fast reservation protocol: $0 \to 2 \to 4$ and $1 \to 2 \to 3$ (see figure 6). Two simulations were run: one with almost zero delay on the lines and one were all the line delays are set to one unit. The rest of the simulation parameters are:

- The burst size is exponentially distributed with mean 50.
- Line capacity is 5 bursts.

Figure 7 compares the success probabilities for the two simulations. Every point in the graphs represents the average of five simulations, most simulations count at least 100,000 bursts (the exceptions are the very lightly loaded simulations where the success probability is very close to 1). The

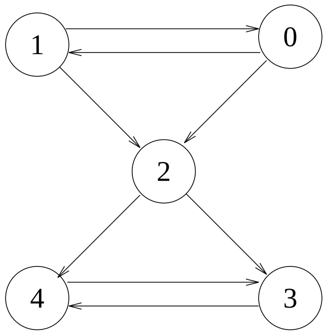


Figure 6: The graph used in the simulations

10% confidence interval for the points in the two top graphs of figure 7 where below 1/2% for loads smaller than 5, and between 1/2 and 3/4% for loads higher than 5. The improvement in the success probabilities is 5-10% for loads between 0.5 and 2; for lower loads the success probability is almost 1 even without the bypass algorithms, leaving a small room for improvements; loads higher than 2 are not often seen and do not last for long. For very high loads, there is a decrease of up to 2% in the success probability when the delay is not negligible. The reason for this decrease is that as the network becomes congested the average reserved route becomes longer (see figure 9) and thus the link holding time for a connection, which equals the sum of the connection holding time and twice the end-to-end propagation delay, becomes longer.

It is interesting to compare the measured improvement of the bypass algorithm that is depicted in figure 7 to an analytical queueing model. In the figure 8, we depict the increase in the success probability between two queueing systems that closely model the simulations described above. System A models the simulation without the bypass algorithm. Since there is no interaction between the two connections, and each link can support five bursts simultaneously, system A is an M/M/N/N queue with five servers, where the service time is distributed as the burst length in the simulation (i.e., exponentially with mean 50) and the arrival process is the same as the burst arrival process for each connection in the simulation (i.e., Poisson with rate in the range [0.01, 1]). System B models the simulation when the bypass algorithm is used. Here the links that are represented by the servers in the analytical model are shared by the two connections. Thus, system B is an M/M/N/N queue with ten servers, where the service time is distributed as the burst length in the simulation, and the arrival process is the sum of the burst arrival processes for the two connections in the simulation (i.e., Poisson with rate in the range [0.02, 2]). Figure 8 depicts the increase in the success probability of customer to receive service in system B compared to this probability in system A, i.e.,

$$\frac{1 - P_{rej,A}(\lambda)}{1 - P_{rej,B}(2\lambda)} \tag{16}$$

where

$$P_{rej,A}(\lambda) = \frac{(\lambda/\mu)^5}{5!} \Big/ \sum_{i=0}^{5} \frac{(\lambda/\mu)^i}{i!} \tag{17}$$

$$P_{rej,B}(\lambda) = \frac{(\lambda/\mu)^{10}}{10!} \Big/ \sum_{i=0}^{10} \frac{(\lambda/\mu)^i}{i!} \tag{18}$$

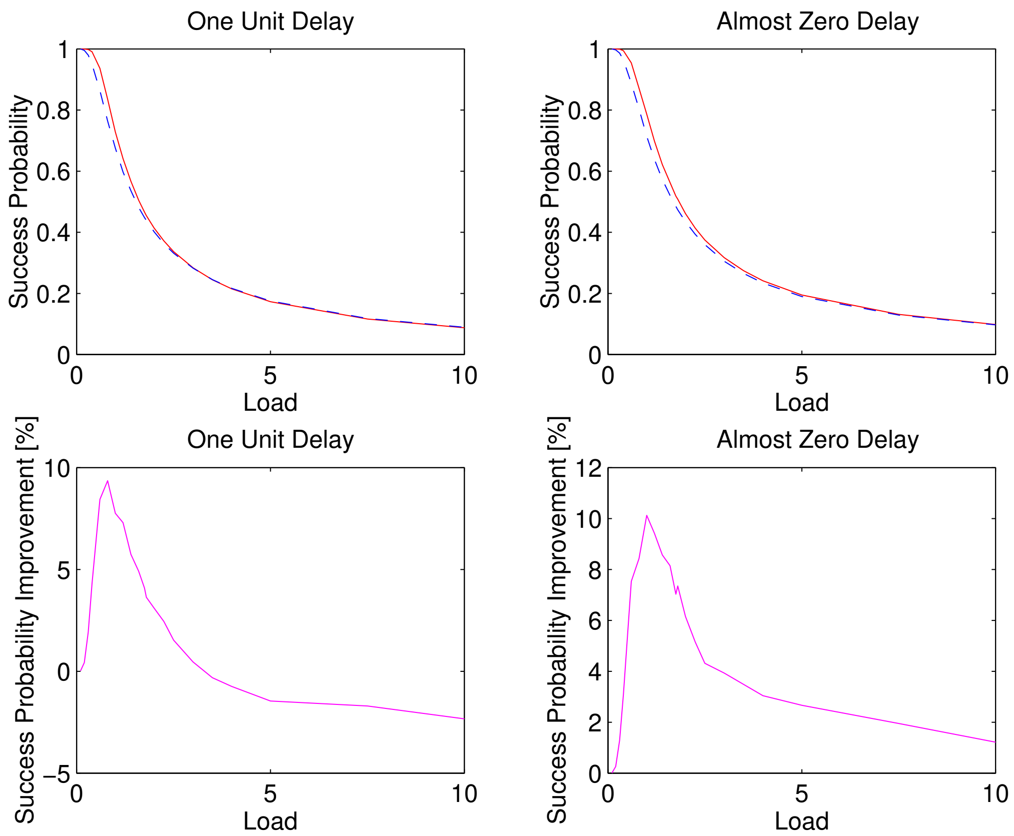


Figure 7: The effect of the bypass algorithm on the success probability of bursts

6 Concluding Remarks

The algorithms presented here can be used in both models, ATM and source routing, for long bursts that use fast reservation algorithms and for short bursts. For bursts that use fast reservation, our algorithms adhere to the reservation principle of the network whether it is ATM based or source routing. For short bursts, we suggest to use on-the-fly reservation in the ATM model and either on-the-fly or no reservation in source routing networks.

Although only two network models are discussed in this paper, the presented algorithm can be easily adapted for other network model, e.g., the up*/down* routing [SBB$^+$91] of Autonet [SBB$^+$91], AN2 [Owi93], and Myrinet [FDCF94]. In these networks, the links form a rooted tree on which wormhole routing in performed. The up*/down* routing constrains a message from using a link in the "up" direction after it traverse a link in the "down" direction and thus exhibits deadlock freeness. Two approaches can be taken to adapt the bypass algorithm to this routing discipline:

- Limit the bypass possibilities in the case of a (down,down) local segment to allow only (down,down) or (down) bypasses. This way, if a climbing worm (one that traversed the last link in the "up" direction) arrives allowed bypass possibilities (e.g., (up,down)) are not considered.
- Maintain two preferred routes in *RoutingTable*, one for climbing worms and (possibly different) one for descending worms. This option is not recommended since it complicate the

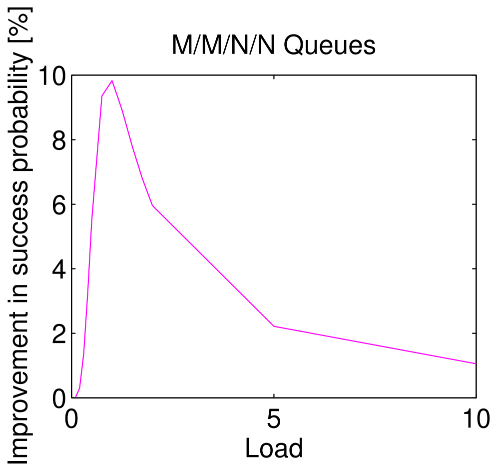


Figure 8: The effect of the bypass algorithm on the success probability of bursts — analytical results.

algorithm and hardware while it improves success probability only for (down,down) local segments, (up,up) and (up,down) type local segment can be followed only by climbing worms.

One of the important merits of our algorithm is the ability to implement it with simple hardware without adversely effecting performance. In [CRS94] 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. This paper is only the first step in evaluating the practicality of our algorithms. An intense simulation study is required as a future work.

References

[AL81] Bruce W. Arden and Hikyu Lee. Analysis of chordal ring network. *IEEE Transaction on Computers*, c-30(4):291 – 295, April 1981.

[AS91] A. S. Acampora and S. I. A. Shah. Multihop lightwave network: a comparison of store-and-forward and hot-potato routing. In *INFOCOM'91*, pages 10 – 19, 1991.

[Bar64] Paul Baran. On distributed communications networks. *IEEE Transactions on Communications Systems*, CS-12(1):1 – 9, March 1964.

[Bou92] Jean-Yves Le Boudec. The asynchronous transfer mode: a tutorial. *Computer Networks and ISDN Systems*, 24:279 – 309, 1992.

[BS65] Robert G. Busacker and Thomas L. Saaty. *Finite Graphs and Networks: an introduction with applications.* McGraw-Hill, 1965.

[BT92] 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.

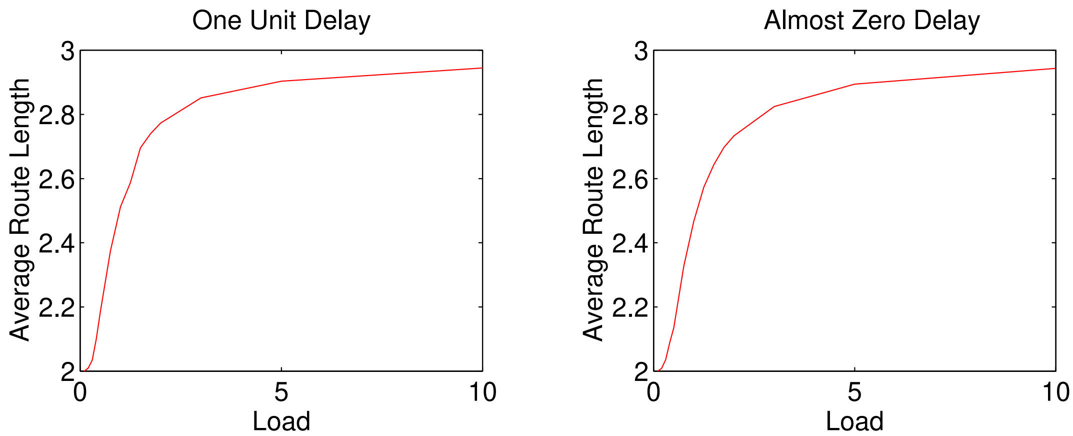


Figure 9: The length of the reserved route as a function of the load.

[CG88] 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.

[CGG+93] 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.

[CGS90] Israel Cidon, Inder Gopal, and Adrian Segall. Fast connection establishment in high speed networks. In *ACM SIGCOM'90*, pages 287 – 296, 1990.

[CPSWL96] Reuven Cohen, Baiju Patel, Frank Schaffa, and Marc Willebeek-LeMail. The sink tree paradigm: Connectionless traffic support on ATM LANs. *IEEE/ACM Transactions on Networking*, 4(3), June 1996.

[CRS94] 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. Submitted for publication.

[CRS96] Israel Cidon, Raphael Rom, and Yuval Shavitt. Analysis of one-way reservation algorithms. *Journal of High-Speed Networks*, 5(4):347 – 363, 1996.

[CS94] Reuven Cohen and Adrian Segall. Connection management and rerouting in ATM networks. In *INFOCOM'94*, pages 184 – 191, June 1994.

[FDCF94] Robert Felderman, Annette DeSchon, Dany Cohen, and Gregory Finn. ATOMIC: a high-speed local communication architecture. *Journal of High Speed Networks*, 3(1):1 – 28, 1994.

[FK71] G. L. Fultz and L. Kleinrock. Adaptive routing techniques for store-and-forward computer-communications networks. In *ICC'71*, pages 39.1–8, 1971.

[GH92] Albert G. Greenberg and Bruce Hajek. Deflection routing in hypercube networks. *IEEE Transactions on Communications*, COM-40(6):1070 – 1081, June 1992.

[GK82] Daniel H. Greene and Donald E. Knuth. *Mathematics for the Analysis of Algorithms*. Birkhauser, second edition, 1982.

[KS91] Mark J. Karol and Salman Z. Shaikh. A simple adaptive routing scheme for congestion control in ShuffleNet mulihop lightwave networks. *IEEE Journal on Selected Areas in Communications*, 9(7):1040 – 1051, September 1991.

[Max87] Nicholas F. Maxemchuk. Routing in the Manhattan street network. *IEEE Transactions on Communications*, COM-35(5):503 – 512, May 1987.

[Owi93] Susan S. Owicki. A perspective on AN2: Local area network as distributed system. In 12^{th} *annual ACM Symposium on Principles of Distributed Computing*, pages 1–11, 1993.

[OY90] Y. Ofek and M. Yung. Principles for high speed network control: loss-less and deadlock freeness, self routing and a single buffer per link. In 9^{th} *annual ACM Symposium on Principles of Distributed Computing*, pages 161–175, 1990.

[Rud76] H. Rudin. On routing and 'delta-routing': a taxonomy and performance comparison of techniques for packet-switched networks. *IEEE Transactions on Communications*, COM-24(1):43 – 59, January 1976.

[SBB+91] Michael D. Schroeder, Andrew D. Birrell, Michael Burrows, Hal Murray, Roger M. Needham, Thomas L. Rodeheffer, Edwin H. Satterthwaite, and Charles P. Thacker. Autonet: A high-speed, self-configuring local area network using point-to-point links. *IEEE Journal on Selected Areas in Communications*, 9(8):1318 – 1335, October 1991.

[Seg83] Adrian Segall. Distributed network protocols. *IEEE Transaction on Information Theory*, IT-29(1):23 – 35, January 1983.

[Tob90] Fouad A. Tobagi. Fast packet switch architectures for broadband integrated services digital networks. *Proceedings of the IEEE*, 78(1):133 – 167, January 1990.

[Tur92] Jonathan S. Turner. Managing bandwidth in ATM networks with burtsy traffic. *IEEE Network*, 6(5):50 – 58, September 1992.

[Uru91] Shigeo Urushidani. Rerouting network: A high-performance self-routing switch for B-ISDN. *IEEE Journal on Selected Areas in Communications*, 9(8):1194 – 1204, October 1991.

[Wid95] I. Widjaja. Performance analysis of burst admission-control protocols. *IEE Proceedings – Communications*, 142(1):7 – 14, February 1995.