ELSEVIER

Computer Networks and ISDN Systems 30 (1998) 1135–1156

COMPUTER NETWORKS and ISDN SYSTEMS

An efficient reservation connection control protocol for gigabit networks [1]

Emmanouel A. Varvarigos [*], Vishal Sharma

Data Transmission and Networking Laboratory, Department of Electrical and Computer Engineering, University of California, Santa Barbara, CA 93106-9560, USA

Abstract

The *Efficient Reservation Virtual Circuit* protocol (or ERVC) is a novel connection control protocol designed for constant-rate, delay-insensitive traffic in gigabit networks. We explain the operation of the protocol, discuss its features and advantages, and present its performance characteristics. The ERVC protocol is appropriate for sessions that require an explicit reservation of capacity and can tolerate the round-trip delay associated with the reservations. In the ERVC protocol, the durations of the sessions are recorded, and every node keeps track of the *utilization profile* of each outgoing link, which describes the amount of residual capacity available on the link as a function of time. This feature allows capacity to be reserved only for the duration of the session, starting at the time it is actually needed. Therefore, the protocol utilizes capacity considerably more efficiently than regular reservation schemes do and results in markedly lower blocking probability for new sessions. The ERVC protocol also has the ''reservation ahead'' feature, which allows a node to calculate the time at which the requested capacity will be available and reserve it in advance, avoiding in this way the wasteful repetition of the call setup phase.

Keywords: Constant-rate traffic; Connection establishment; Protocol; Reservation-based; Performance characteristics

1. Introduction

In the era of gigabit networking, which is rapidly becoming a reality with advances in VLSI technology and fiber-optic transmission systems, networks will be limited by the propagation delay on the channel [1]. This is a limitation imposed by physics that will not change with improved implementation. The important network issues of connection establishment, flow control, buffering, and congestion control therefore need to be re-examined in this changed communication environment. As Partridge [2] in his series of thought questions on the challenges of gigabit networking rightly points out, high bandwidth-delay product and increased propagation latency are two crucial factors that will differentiate gigabit networks from most present day networks, and that will impact not only the performance of such networks, but also the protocols designed to manage them and the applications designed to use them.

Propelled by the advances in new communication technologies, a number of prototype high-speed networks have been developed in the US and Europe to date. These include the PARIS high-speed network (see [3,4] for various aspects of its design), and its

[*] Corresponding author.

[1] Research supported by ARPA under Contract DABT63-93-C-0039.

0169-7552/98/$19.00
PII: S0169-7552(98)00009-9

successor the plaNET/ORBIT high-speed network [5] that will be used in the AURORA gigabit testbed [6]. Aurora is one of the five gigabit testbeds being overseen by the Corporation for National Research Initiatives [7]. The other networks in this initiative include the Vistanet [8], CASA [9], Blanca [10], and Nectar [11] gigabit testbeds. Other gigabit research initiatives within the US include the MIT Lincon Labs.-based all-optical network consortium [12]. High-speed networks outside the US include the MultiG project funded by the Swedish Institute of Computer Science [13], the Cambridge backbone ring in Britain [14,15], the Berkom project in Germany [16], and TEN-34 [17], a new gigabit network testbed deployed in Europe.

The ERVC protocol, which is the subject of this paper, is part of the connection and flow control protocols designed for the Thunder and Lightning network (see, for instance, [18–21]), which is currently being built at UCSB under the sponsorship of DARPA [22]. The Thunder and Lightning network is a virtual-circuit switched, fiber-optic network that will operate at serial link speeds of up to 40 Gbit/s, and is projected to carry a diverse mix of traffic types. Our objectives in designing the connection and flow control algorithms for this network are to ensure lossless transmission, efficient utilization of capacity, minimum pre-transmission delay for delay-sensitive traffic, and packet arrival in correct order. The two new connection control protocols that we have proposed to meet these objectives are the Ready-to-Go Virtual Circuit (or RGVC) protocol and the Efficient Reservation Virtual Circuit (or ERVC) protocol. The RGVC protocol [20], which will be used for best-effort service and for traffic that has little tolerance for delay, uses back-pressure and buffering at intermediate nodes, whereas the ERVC protocol, which is appropriate for constant-rate sessions that require guaranteed bandwidth, uses explicit reservations and requires little buffering at intermediate nodes.

The very high link speeds in the Thunder and Lightning network were proposed because of the realization that at least a sizable portion of traffic in future gigabit networks would involve high-speed transfer of massive amounts of data at nearly constant rates, and would require guaranteed lossless delivery and an explicit reservation of bandwidth. The ERVC protocol, proposed in this paper, provides efficient transfer of such data, and has been motivated by several design considerations. First, for several applications such as high-speed program-data transfer between supercomputers or bulk file transfer, loss-based traffic integration will not be practical [23]. This is because providing a reliable transport service requires that, ideally, not even a single cell be lost (here loss refers to the loss due to statistical multiplexing and not that due to transmission errors), because each such loss can force retransmission of large volumes of data. Clearly, in gigabit networks like the Thunder and Lightning network, the bandwidth-delay product, being very large, can result in the discarding of substantial amounts of data in case of retransmissions if efficient lossless transfer is not provided. Second, for high-speed file-transfer type applications, long burst transmissions can easily overload the network, unless they have pre-negotiated at least a minimum bandwidth with the network. This has also been realized by several other researchers, many of whom have advocated burst-based bandwidth reservation as a viable and prudent choice (see Hiu [24], Ohnishi et al. [25], Suzuki and Tobagi [26], and Iwata et al. [27]). Therefore, from the point of view of both transmission integrity and network efficiency, traffic of this type should be transferred only after a specific and explicit allocation of resources precedes each data burst.

The ERVC protocol is employed for call setup if the session is not critically delay-sensitive and requires an explicit reservation of bandwidth. In what follows, the word session will be applied both to a new session, or to a new burst for an ongoing session that has a virtual path to the destination but may have no bandwidth reserved on it before hand. New sessions are generated at each source with a specified destination, duration, tolerable delay, and bandwidth requirement. For a new session, a path with adequate residual capacity is computed at the source, based on the topology and link utilization information that it has at that time. For a burst belonging to a continuing session, this computation is performed at the connection set up phase and is not required at this time. In the ERVC protocol, the duration of a session (or burst) is recorded during the call establishment phase, and each network node keeps track of the capacity available on its outgoing links as a

function of time. A setup packet is sent over the path to make the appropriate reservations and set the routing tables. Each intermediate node reserves the required capacity starting at the time at which this capacity will actually be used (which is at least one round-trip delay after the arrival of the setup packet at the node), and for a length of time equal to the session duration. If the session duration is unknown, it is treated as infinite, and capacity is reserved for that session for an unspecified duration (as in regular reservation schemes). If the capacity is not available at the requested time, the setup packet may make a reservation starting at the first time the capacity becomes available. Therefore, the ERVC protocol has the ''reservation ahead'' feature, in the sense that capacity may be reserved in advance for use at a later time. This differs from other reservation virtual circuit protocols, where session durations are not recorded and capacity is reserved starting at the time the setup packet arrives at a node. The idea of advance reservation of resources has been used in the past mainly in connection with video-teleconferencing centers (see [28–31]). In these works, there is a (single-server or multi-server) central reservation office that users call to request conferencing facilities for a given future date. This is quite different than the reservation ahead feature of the ERVC protocol, where the objective is the efficient utilization of the network links, and capacity is reserved in a distributed manner, through the collaboration of nodes on a session's path.

In the ERVC protocol, capacity is blocked for other sessions only for the duration of the call, what we call the *timed reservation* feature of the protocol, and is available for the remaining time. As the discussion of Section 2 and the performance results presented in Section 4 will indicate, this is particularly important for high-speed networks, where propagation times are large compared to transmission times, because it reduces blocking and allows a greater number of sessions to be served. It also avoids the wasteful repetition of the call establishment process, because it enables a session to reserve the required capacity in its first attempt, probably at a time later than the requested time. If the time at which adequate bandwidth first becomes available is unacceptable because of the delay requirements of the session, the call is blocked and is reattempted later, probably using a different path. The ERVC protocol requires a pre-transmission delay at least equal to the round-trip propagation delay between the source and the destination (as all reservation protocols do).

Several implementation issues, some of which we discuss in the latter sections of our paper, are important for the correct and efficient operation of the ERVC protocol. Correct timing is important in order to ensure that data transmission starts after all reservations are made and terminates before any intermediate node releases the reserved capacity. Also, since reservations are made in a distributed, asynchronous way, through the collaboration of the nodes on a session's path, and capacity on a link may not be available at the time requested for by the setup packet but at some future time, it is important to ensure that the time intervals reserved on each link are consistent with each other. For the ERVC protocol to function efficiently, the queueing and processing delays of control packets have to be small and predictable, and the uncertainties and round-off errors in recording the various time parameters have to be controlled. The information required by the protocol (rates and session durations) has to be recorded and processed in an efficient way. As we discuss in Section 3, we use a linked-list structure to store the rates and durations of the sessions, and record times as relative times, which enables a fast list update to be performed at a switch. Finally, the protocol has to cope with link and node failures.

The remainder of the paper is organized as follows. In Section 2 we provide the motivation for the ERVC protocol by discussing its advantages over other reservation protocols for high-speed networks, and we describe some applications that would benefit from it. In Section 3 we explain the operation of the protocol. Specifically, in Section 3.1 we describe the call setup procedure, and in Section 3.2 we describe the data structures that are required and the way they are updated. The connection control actions performed by the source, intermediate, and destination nodes are outlined in Section 3.3. In Section 3.4 we comment on the delays experienced by control packets, and discuss how the protocol handles uncertainties in timing. In Section 4, we present the performance characteristics of the ERVC protocol, and we compare its performance to that of regular

reservation schemes. Finally, in Section 5 we give some concluding remarks, while in the appendices we give the details of the connection control actions performed at the nodes and discuss how the protocol copes with link and node failures.

2. Why the ERVC protocol?

In this section, we discuss some drawbacks of other reservation schemes and explain how the ERVC protocol helps to overcome them. We use the term immediate reservation virtual circuit schemes (abbreviated IRVC) to refer to schemes where the capacity required by a session at an intermediate node is reserved starting at the time the setup packet arrives at the node, and which, unlike the ERVC protocol, do not allow for reservations to be made for future time instants (we discuss this aspect shortly). This includes several recently proposed reservation schemes such as the FRP/DT protocol (Fast Reservation Protocol with Delayed Transmission) proposed by Boyer and Tranchier [32], the fast bandwidth reservation schemes of Suzuki et al. [26], the fast resource management (FRM) protocols mentioned by Fotedar et al. [33] and discussed in detail by Tranchier et al. [34], and the connection establishment scheme proposed by Cidon et al. [3]. The scheme proposed by Cidon et al. uses a logical tree to execute a distributed capacity check algorithm that speeds up the reservation phase, but it still suffers (even though to a lesser extent) from a drawback common to immediate reservation schemes as we now explain. A cause of the inefficiency in these schemes arises because the capacity reserved for the session is actually used at least one round-trip delay after the arrival of the setup packet at the node. This is because the setup packet has to travel from the intermediate node to the destination, an acknowledgement has to be sent from the destination to the source, and the first data packet of the session has to travel from the source to the intermediate node (see Fig. 1).

Over long transmission distances, the round-trip propagation delay may be comparable to, or even larger than, the holding time of a session. In particular, if a typical session requests capacity r bits/s, and transfers a total of M bits over a distance of L

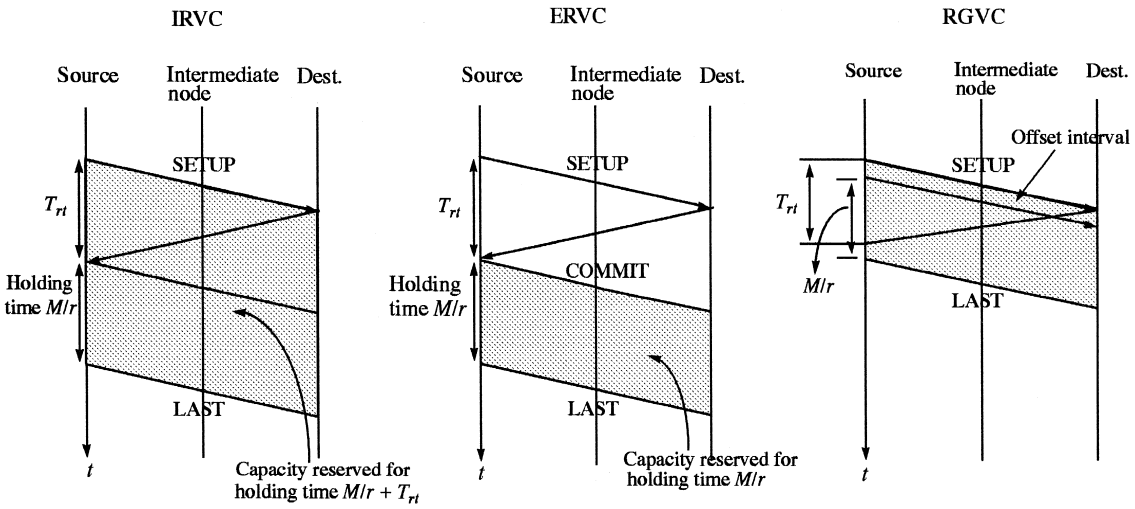


Fig. 1. IRVC, ERVC, and RGVC protocols for the case where the setup packet is successful in making the appropriate reservations. In IRVC protocols, where session durations are not recorded, the capacity is blocked for duration equal to $(M/r) + T_{rt}$, where T_{rt} is the roundtrip propagation delay. In the ERVC protocol, capacity is blocked for other sessions only for the holding time M/r. In the RGVC protocol (a tell-and-go type of protocol that will also be used in the Thunder and Lightning network), capacity is occupied for time M/r plus the time offset between the transmission of the setup packet and the first data packet of the session. In the RGVC protocol [20], the setup packet is first transmitted along the path, followed after a short offset-interval by the data packets, with back-pressure exercised if needed.

kilometers, then the maximum percentage of time that the capacity is efficiently used in an IRVC protocol is

$$e = \frac{M/r}{2L\nu/c + M/r}, \quad (1)$$

where $\nu/c = 5$ μs/km is the propagation delay in the fiber. Typical values of the above parameters for gigabit networks like the Thunder and Lightning network are expected to be $r = 10$ Gbit/s, $M = 0.2$ Gbit (corresponding, for instance, to the transfer of a medium-sized file in a high-speed data-transfer operation), and $L = 1500$ km (long haul communication), which yields $e = 0.57$. In other words, for the above parameters, the capacity reserved by a typical session with an IRVC protocol stays idle for a round-trip delay of 15 ms, and is used for time equal to the typical holding time of a session, which is equal to 20 ms. This efficiency factor e becomes even smaller as r or L increase, or M decreases. In contrast, the efficiency factor e for the ERVC protocol can be as large as $e = 1$, independently of the parameters r, L, and M. In the simulations results presented in Section 4, we show that the ERVC protocol can achieve link utilization close to one while keeping the blocking probability for new sessions below some reasonable threshold, say 0.1. By contrast, IRVC protocols achieve their maximum utilization (which is always less than that of the ERVC protocol) at the cost of considerable blocking of new sessions, which imposes a further penalty on the network through multiple reservation attempts.

To understand the advantage of recording session durations and the ''reservation ahead'' feature of the ERVC protocol, consider the situation shown in Fig. 2 where a setup packet requests 10 Gbit/s of capacity on an outgoing link l that has only 5 Gbit/s of capacity available at that time. If an IRVC protocol is used, such a call will be blocked. Since, however, reservations on link l are such that 10 Gbit/s of capacity will become available after 14 ms, and the first data packets of the new session will arrive at the link after at least 30 ms (a round-trip delay), the session should be accepted. If the ERVC protocol is used, such a call will be accepted, because each node records the session durations and the setup packet will be able to reserve 10 Gbit/s of capacity starting

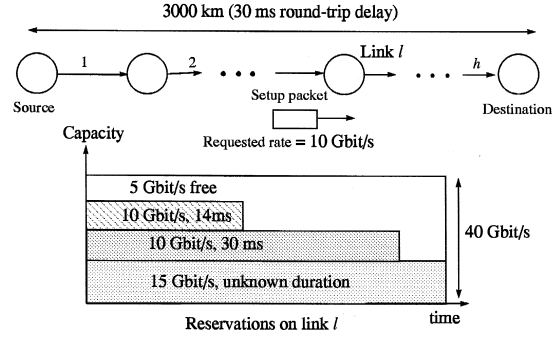


Fig. 2. The advantage of recording session durations in the ERVC protocol and its ''reservation ahead'' feature.

at a time 30 ms after its arrival at the node. The ERVC protocol also has the ''reservation ahead'' feature, which allows sessions to reserve capacity ahead in time and avoids repetition of the call setup phase. To see this, assume that the round-trip delay of the setup packet in Fig. 2 is only 10 ms, while the acceptable delay of the session is 18 ms. The setup packet now requires 10 Gbit/s of capacity on link l starting at a time 10 ms after its arrival at the intermediate node. Since 10 Gbit/s of capacity is available on link l after 14 ms and this time is within the range of delays that can be tolerated by the session, the call will be accepted on its first attempt. Thus, the reservation ahead feature avoids unnecessarily prolonged call setup phases and the associated waste of bandwidth, reduces a session's susceptibility to blocking, and leads to efficient utilization of the available capacity. We discuss the performance advantages that result from this feature when presenting our simulation results in Section 4.

We see at the present time, several potential applications where the ERVC protocol will be useful. One such application is in the area of distributed network computing [35], especially computing involving several powerful supercomputers that will need to exchange bulk data, relatively frequently and without any loss. Virtual circuit connections with a minimum guaranteed bandwidth will have to be set up between these sites to facilitate the efficient transfer of data when needed [36]. For geographically separated sites where the round-trip delay is comparable to or larger than the data transmission time, it will be most efficient to reserve capacity only for the

duration of the transfer, rather than suffer from the overhead of reserving bandwidth for a round-trip delay before the data can be transmitted. The ERVC protocol will also be useful for applications that run for part of the time on a specific machine suited to execute a portion of the computation most efficiently, and for part of the time on a different machine suited to execute a different portion of the computation [2]. Typically, such applications manipulate very large volumes of data that must be moved efficiently from one machine to another in mid computation. Another area where the ERVC protocol can be useful is in digital medical imaging applications, which often require the transmission of diagnostic X-rays [35] involving as many as 10 to 50 images (amounting to between 2 and 10 gigabits of information). In this case also, virtual paths between several locations can be pre-established, with virtual circuits requiring specific bandwidths being established on an ''as needed'' basis. By reserving fairly large chunks of the link capacity only for the duration of the data transfer, the protocol will enable better use of a network's resources, and thus streamline network operation.

The timed reservation feature of the ERVC protocol (whose details we give shortly) is a relatively inexpensive way to effect a substantial gain in network efficiency, especially in view of a number of network applications where the size of the data transfer is expected to be known beforehand, thus allowing for reservations to be made only for the durations of the sessions. For instance, in most file-transfer or data-transfer type applications (such as those involving transfers over the world-wide web), the size of the network transfer is known in advance. In virtual-circuit switched networks, therefore, such transfers lend themselves immediately to the use of the ERVC protocol. In this case, the setup packet would travel from the source (user) to the destination (remote database or data site), while making timed reservations on links in the reverse direction for the data to be sent from the data-site to the user.

3. Operation of the ERVC protocol

In this section, we explain the call setup mechanism, and describe how the reservations are made. We then discuss the data structures that are used for this purpose, and the way they are updated, and indicate the connection control actions that must be performed at the source, intermediate, and destination node (the state diagram representations of these actions and a detailed description appear in Appendix A). Finally, we comment on some implementation issues.

3.1. Call setup in the ERVC protocol

We begin by explaining the operation of the ERVC protocol when there are no link or node failures, and when propagation and other delays are known. In Section 3.4 we explain how the protocol deals with timing uncertainties, and in Appendix B we discuss how it handles link and node failures. We assume the availability of local clocks but do not assume that the clocks are synchronized. We also assume, as is the case for the Thunder and Lightning network, that error control and retransmission are performed at the transport layer, and are not part of the connection control protocol.

The path followed by a session is computed at the source based on the topology and link utilization information available at the source. The protocol can be modified to deal with the case where adaptive routing is used; in this paper, however, we only consider the case where source routing is used. Each outgoing link of a node has an identifier, which is unique within that node, and is globally known.

A SETUP packet contains several fields as shown in Fig. 3. The path of the SETUP packet is specified as a sequence of link identifiers $L_1, L_2, \ldots, L_h$, corresponding to the links that the packet must traverse. The switch processor (SP) reads the first link identifier of the incoming SETUP packet to determine the outgoing link to which the packet should be routed, and cyclically rotates the link identifiers so that the one just read becomes last. Therefore, at each hop, the first link identifier always specifies the outgoing link of the switch to which the packet should be sent. The SETUP packet also contains the virtual path identifier field VPI; a session can have different

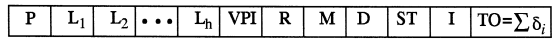


Fig. 3. The various fields of the SETUP packet.

values of VPI for various segments of its path. The requested rate field R is the rate required by the session. The minimum rate field M is the minimum rate guarantee with which the session will be satisfied; in general, we may have $M \neq R$. The start time field ST of a SETUP packet is initially set equal to the round-trip propagation delay T_{rt} between the source and the destination, and is updated at each node in a manner to be described later. When the SETUP packet leaves the ith intermediate node on its path, the value of its ST field is denoted by ST_i. Upon reception of the SETUP packet at an intermediate node, the ST field specifies the time (relative to the present time) at which the reservation of capacity at its outgoing link should begin. The maximum delay field D specifies the maximum delay that the session can tolerate. It is the time, relative to the time the SETUP packet starts transmission at the source, within which the source should either transmit its first data packet or learn that the session cannot be served because the delay or the capacity requirement could not be met. Since the ERVC protocol requires a pre-transmission delay equal to at least T_{rt}, it can be used only when $D \geq T_{rt}$ (otherwise, a tell-and-go type of protocol, like RGVC is used for the session). The information field I specifies the amount of information that will be transmitted during the holding time of the session, if known. Finally, the time-offset field TO, which is updated at each hop, contains the time, following the reception of the acknowledgement packet at the source, after which the source should start transmission.

When the ith intermediate node s_i receives a SETUP packet, it finds the first time $t \geq ST_{i-1}$ at which enough residual capacity is available to accommodate the call. If $t \geq D$ (that is, the delay until the capacity becomes available is unacceptable), the session is blocked. If $t \leq D$, intermediate node reserves the capacity and updates the ST field to $ST_i := t$ (see Fig. 4). The node updates the TO field by adding to it the time offset $\delta_i = t - ST_{i-1}$ introduced at s_i, and forwards the SETUP packet to the next node s_{i+1}. The destination node also uses a similar algorithm to process the SETUP packet, except that instead of checking for the availability of adequate capacity, it checks for the availability of adequate memory to store the I information bits of the session. The destination acknowledges the

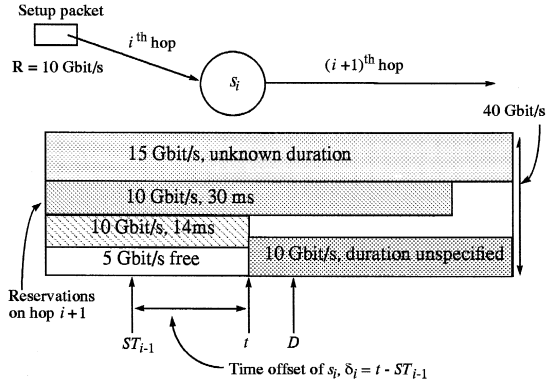


Fig. 4. Advantage of the ERVC protocol. Here the session requesting 10 Gbit/s that arrives at intermediate node s_i, is not rejected. Instead a reservation for it is made at the first time t, where $ST_{i-1} \leq t \leq D$, at which enough capacity is available. At the same time the time offset (TO) field of the SETUP packet is incremented by $\delta_i = t - ST_{i-1}$.

SETUP packet by sending an ACK packet back to the source, which contains the rate finally allocated to the call and the time offset $TO = \sum_{i=0}^{h} \delta_i$. After receiving the ACK, the source waits for time $\sum_{i=0}^{h} \delta_i$ and transmits a COMMIT(A, H) packet, containing the allocated rate A and the expected session holding time $H = I/A$. If the reservation is unsuccessful, the node where the session is blocked returns a REJ packet. If nothing is received within a time T, which is an appropriate function of the round-trip delay, REJ is assumed.

Fig. 5 illustrates the timing considerations for the ERVC protocol. As long as the propagation delay on a link is the same for all packets (that is, for the SETUP packet and the subsequent data packets), it is sufficient for a node to reserve capacity (relative to present time as per its local clock) starting at the time specified in the ST field of the SETUP packet. The effect of delay uncertainties and modifications required to handle them are described in Section 3.4.

3.2. Reservation procedure and data structures

The way the reservations for a session are made and the data structures required for this purpose are explained next. Each switch processor has to keep a record of the rate $G(V)$ granted to each session V that uses that port. The processor also records the total rate of sessions with unknown duration, or

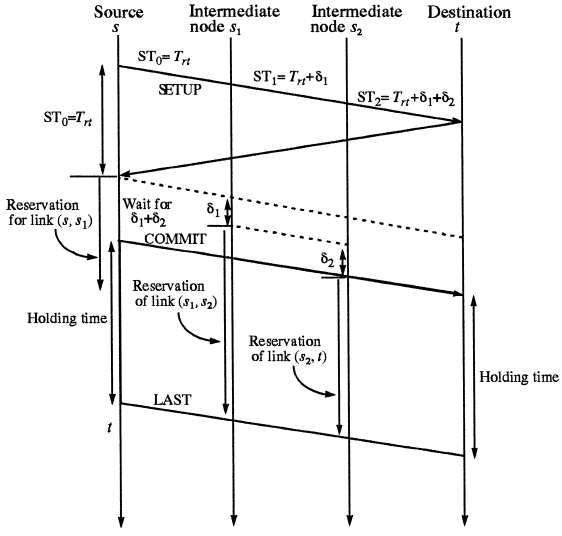


Fig. 5. To illustrate the timing considerations of the ERVC protocol, consider the following scenario. The SETUP packet is transmitted from the source s with start time field $ST_0 := T_{rt}$. At each node s_i on the path, the ST field is incremented by the time offset δ_i introduced by s_i, and capacity is reserved starting at time ST_i relative to the arrival time of the SETUP packet at s_i. Therefore, link (s, s_1) is reserved starting at a time ST_0 after the SETUP packet is sent from s, link (s_1, s_2) is reserved starting at a time ST_1 after the SETUP packet arrives at node s_1, and link (s_2, t) is reserved starting at a time ST_2 after the SETUP packet arrives at node s_2. When the packet arrives at the destination t, the ST field is $ST_2 = T_{rt} + \delta_1 + \delta_2$. After receiving the ACK, the source waits for $\delta_1 + \delta_2$ time units before transmitting the COMMIT packet and the data packets. Therefore, the COMMIT packet arrives at node s_1, δ_2 time units after the starting time of the reservation on link (s_1, s_2), and it arrives at node s_2 exactly at the starting time of the reservation on link (s_2, t). This ensures that when the data packets of the session arrive at an intermediate node, capacity is available for them.

sessions whose rate is unrecorded for other reasons (for example, for small sessions with rate less than, say, 0.1 Gbit/s, the duration may not be recorded in order to reduce complexity), or for sessions using other connection control protocols (like immediate reservation protocols). Sessions whose duration is recorded will be called *type* 1 *sessions*, while all other sessions will be called *type* 2 *sessions*. Note that type 2 sessions are also sessions that need bandwidth guarantees and require explicit reservations, and therefore do not include available bit-rate type sessions. For type 2 sessions we assume that $M = R$. For dealing with type 2 sessions, either one of the following two options may be used. Sessions for which both the roundtrip delay and the holding time are uncertain can also be treated as IRVC sessions, with their aggregate rate being recorded in a variable S. Sessions for which the roundtrip delay is known but whose duration is uncertain are treated like type 1 sessions, with capacity being reserved for them, initially for an infinite duration, and being freed when the LAST packet of the session arrives. In the remainder of the paper, we will assume for simplicity that all type 2 sessions are treated as IRVC sessions, with aggregate rate S.

In the ERVC protocol, each node has a record of the capacity reserved on its outgoing links as a function of time. We let $r^l(t)$ denote the capacity reserved on link l by type 1 sessions, at time t relative to the present time. The function $r^l(t)$ is called the *utilization profile* of link l. The capacity available on link l at time t is then equal to $C - S - r^l(t)$. The utilization profile is a stepwise function, with discontinuities at the points at which reservations begin or end (see Fig. 6), and has to be updated dynamically. In our implementation, the node keeps track of the utilization profile by recording only the jumps in $r^l(t)$. In particular, a network node maintains a linked list $\mathscr{L}_l$, called *utilization list*, for each outgoing link l. Each record of $\mathscr{L}_l$ stores one jump of the link utilization profile, and is composed of two fields, the *rate field* ($\mathscr{L}_l$*_rate*) and the *time field* ($\mathscr{L}_l$*_time*). The field $\mathscr{L}_l$*_rate* is equal to the capacity reserved by type 1 sessions, at a time recorded in the field $\mathscr{L}_l$*_time* (relative to the time in the preceding record). The first element denoted by $\mathscr{L}_l[t_0 = 0, r_0]$ records the capacity r_0 that is reserved during the interval $[0, t_1)$, where $t_1 = \tau_1$. The second element $\mathscr{L}_l[\tau_1, r_1]$ records the capacity r_1 that is reserved during the interval $[\tau_1, \tau_1 + \tau_2)$, or equivalently during the interval $[t_1, t_2)$, and so on till the $(m-1)$th element $\mathscr{L}_l[\tau_{m-1}, r_{m-1}]$, which records the capacity r_{m-1} reserved during the interval $[\sum_{i=1}^{m-2}\tau_i, \sum_{i=1}^{m-2}\tau_i + \tau_{m-1})$, or equivalently during the interval $[t_{m-2}, t_{m-1})$. We will use the convention that the last element $\mathscr{L}_l[\tau_m, r_m]$ of the list has fields $t_m = \infty$ and $r_m = 0$. This is because at time $t_m = \infty$, all sessions with finite (known) durations will have ended, and the capacity r_m reserved by them will be zero.

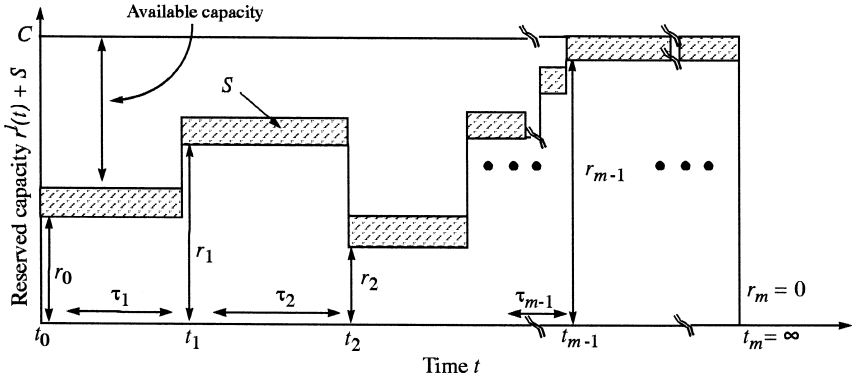


Fig. 6. Illustrating the link *utilization profile* $r^l(t)$ of a link l. This function describes the capacity reserved on link l by *type* 1 sessions at each time instant t. Observe that it is enough for a node to know the values of $r^l(t)$ at the points where there is a step or jump in the function $r^l(t)$. Here C is the total capacity of the link and S is the portion of capacity allocated to *type* 2 sessions.

Note that in the utilization list, the time field $\mathscr{L}_l\_time$ of each element records time relative to the time in the preceding element. Thus, the time field τ_1 of the second element records the time relative to the present time $t_0 = 0$ recorded in the first element. The time τ_1 is updated every millisecond by decrementing its value by one, an update that takes constant time and is independent of the size of the list. Since the utilization list must always have a first element with $t_0 = 0$ to represent the current time, the first element of the list $\mathscr{L}_l[t_0, r_0]$ is deleted only when the time field τ_1 of the second element $\mathscr{L}[\tau_1, r_1]$ reduces to zero. At that point, the second element moves to the first position, and the first element is deleted.

Consider a SETUP packet that arrives at the pth intermediate node s_p on its path, and wants to use link l for its $(p+1)$th hop. [For simplicity, in the following we will assume that the reservation for a session is initially made for an unknown (infinite) duration. The capacity reserved in excess of the holding time of the session is released upon the arrival of the COMMIT packet. When the entire procedure has been explained, the reader will easily see that the initial reservation of capacity on a link l at the $(p+1)$th hop of the path needs to be made only for time equal to $D - ST_{p-1} + I/M$, provided that I is known during the set-up phase, which is an upper bound on the time for which the session may need the capacity on link l.] Let k be the maximum integer such that the time fields of the elements of $\mathscr{L}_l$ satisfy

$$\sum_{i=1}^{k} \tau_i < ST_{p-1},$$
$$\text{and } C - S - r_i \geq M \text{ for all } i \in \{k, k+1, \ldots, m\}. \quad (2a)$$

In other words, $\sum_{i=1}^{k} \tau_i$ is the largest time that is less than the time at which the link is requested, for which adequate capacity is available at all times $t \geq \sum_{i=1}^{k} \tau_i$. If such a k exists, capacity can be allocated to the session starting at the requested time ST_{p-1}, without adding any further delay to the start time of the session. If a k satisfying the above condition does not exist, the node searches for the minimum integer k such that the time fields of the elements of $\mathscr{L}_l$ satisfy

$$ST_{p-1} \leq \sum_{i=1}^{k} \tau_i \leq D,$$
$$\text{and } C - S - r_i \geq M \text{ for all } i \in \{k, k+1, \ldots, m\}. \quad (2b)$$

That is, the node searches for the first time $\sum_{i=1}^{k} \tau_i$ after the requested time ST_{p-1} at which adequate capacity is available, and which is less than the maximum allowable delay D. If such a k exists, the node allocates that capacity to the session starting at time $\sum_{i=1}^{k} \tau_i$. In this case, however, a delay $\delta_p = \sum_{i=1}^{k} \tau_i - ST_{p-1}$ is added to the start time of the

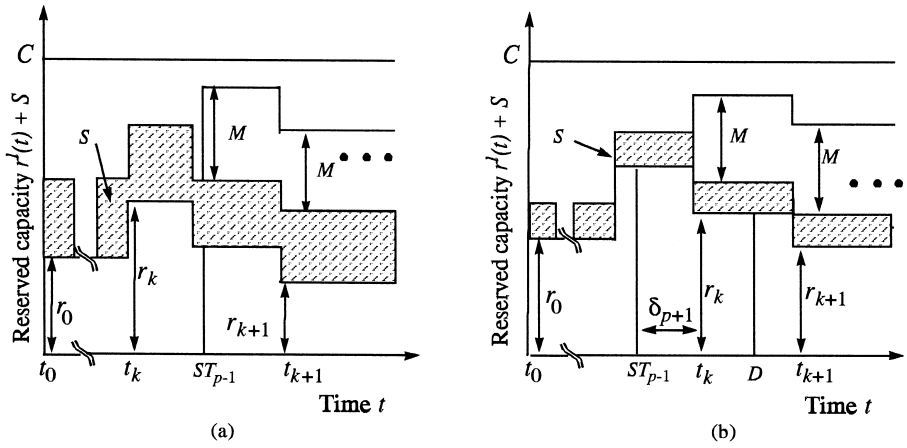


Fig. 7. Illustrates how capacity is reserved for a new session. The setup packet arrives at link l after its pth hop. In (a) $t_k < ST_{p-1}$ and capacity $\geq M$ is available at all subsequent times. Thus, a reservation is made starting at time ST_{p-1}, and a new record is inserted in the list to account for the discontinuity introduced at that time by the new session. In (b), $ST_{p-1} \leq t_k \leq D$ and adequate capacity is available at all times after t_k. As a result, capacity is reserved starting at time t_k and the start time ST_{p-1} of the session is incremented by the offset $\delta_p = t_k - ST_{p-1}$ introduced at the $(p+1)$th hop.

session. If a k satisfying condition (2a) or condition (2b) does not exist, then the session is blocked, and a REJ is transmitted to the source (Fig. 7).

In what follows we describe how the data structures are updated to perform the reservations. We will first describe the updates for type 1 sessions, and then for type 2 sessions.

For a type 1 session, if a k satisfying condition (2a) exists, capacity equal to $\min(R, C-S-r)$ is reserved for the session starting at time ST_{p-1}, where $r = \max_{k \leq i \leq m}\{r_i\}$. This is done by inserting a new element with $\mathscr{L}_l\_rate := r_k$ and $\mathscr{L}_l\_time := \sum_{i=1}^{k}\tau_i - ST_{p-1}$ between the kth and the $(k+1)$th position of the list (that is, the new element becomes the $k+1$ element of the revised list). The time field of the $(k+2)$th element of this revised list is updated to

$$\tau_{k+2} = \tau_{k+2} - \tau_{k+1},$$

and the residual capacities of all elements following $\mathscr{L}_l[\tau_k, r_k]$ are updated according to

$$r_i := r_i + \min(R, C-S-r)$$

for all $i \in \{k+1, \ldots, m+1\}$.

If a k satisfying condition (2b) exists, no new element needs to be added to the list, but the elements following $\mathscr{L}_l[\tau_{k-1}, r_{k-1}]$ are updated according to

$$r_i := r_i + \min(R, C-S-r) \quad \text{for all } i \in \{k, \ldots, m\}.$$

In both cases, the fields of the SETUP packet are updated according to

$$R^{\text{new}} := \min(R^{\text{old}}, C-S-r),$$

$$ST_p := \max\left(ST_{p-1}, \sum_{i=1}^{k}\tau_i\right),$$

$$\delta_p := \max\left(ST_{p-1}, \sum_{i=1}^{k}\tau_i\right) - ST_{p-1},$$

and

$$TO^{\text{new}} := TO^{\text{old}} + \delta_p.$$

At this time the node also records the new call parameters in its call tables, with the rate granted field set to $G = R^{\text{new}}$.

We now describe the procedure followed when node s_p receives a COMMIT(A, H) packet, which confirms that capacity A has been allocated to the session, and that the session holding time is $H = I/A$. Recall that for a type 1 session the rate A finally allocated to the session may be smaller than the rate G granted to it on an intermediate link during the call setup phase. When the COMMIT(A, H) packet for a session arrives, all capacity reserved for this session beyond its holding time H must be freed.

Further, the excess capacity $G-A$, must also be freed. We let n be such that

$$\sum_{i=1}^{n} \tau_i \le H < \sum_{i=1}^{n+1} \tau_i.$$

If $\sum_{i=1}^{n}\tau_i < H$, then a new element is added to $\mathscr{L}_l$ in the $(n+1)$th position with fields given by $\mathscr{L}_l\_time := H - \sum_{i=1}^{n}\tau_i$ and $\mathscr{L}_l\_rate := r_n$. That is, if there is no n such that the sum of the time fields of elements 1 through n is equal to the time at which the new session ends, a new element has to be added to $\mathscr{L}_l$ to record the discontinuity introduced by the termination of the new session. The rate field of all elements is then updated according to

$$r_i := r_i - G \text{ for all } i \in \{n+1, n+2, \ldots, m+1\},$$

and

$$r_i := r_i - (G-A) \text{ for all } i \in \{0,1,\ldots,n\}.$$

If $\sum_{i=1}^{n}\tau_i = H$, no new element is added to $\mathscr{L}_l$, and the rate field of all elements is updated according to

$$r_i := r_i - G \text{ for all } i \in \{n, n+1, \ldots, m\},$$

and

$$r_i := r_i - (G-A) \text{ for all } i \in \{0,1,\ldots,n-1\}.$$

Note that, in practice, both during the process of session admission and when updating the capacity allocated to a session on the arrival of the COMMIT(A,H) packet, a node can accomplish the update by scanning the list only once (see implementation details in [37]).

When the setup packet for a type 2 session arrives at a node, the procedure followed to update the variables is different. If the session can be accepted, that is

$$C - S - r_i \ge M \text{ for all } i \in \{0,1,\ldots,m\}, \quad (3)$$

the node creates a new record for it and sets the rate granted field G equal to the requested rate R (which for type 2 sessions is also equal to the minimum rate M). The total capacity of all type 2 sessions is then updated according to $S := S + R$. When the COMMIT packet for a type 2 session arrives at the node the reservation is confirmed, and no further action needs to be taken.

Lastly, we need to consider how the utilization list $\mathscr{L}_l$ is updated when a session either terminates normally, or is dropped either due to a time-out or because it is blocked at a subsequent node. When the session terminates normally, the source transmits a LAST packet to free the resources along the path, and each intermediate node updates its local call tables by deleting the corresponding record. If the session is of type 1, the capacity allocated to it is automatically released at this time. If the session is of type 2, the node updates S according to $S = S - G$. When the session is blocked after having reserved capacity on some of its intermediate links, the node i where the session is blocked transmits a REJ packet to the source, containing the value of the *TO* field of the SETUP packet. After receiving a REJ, the source waits for time $TO + \max(0, T_{rt} - t)$, where t is the time at which the REJ is received at the source, and then transmits a RELEASE packet to free the capacity reserved. This ensures that the RELEASE packet arrives at the intermediate nodes $1,2,\ldots,i-1$ at the time starting at which a reservation was made for the session at these nodes. When the RELEASE packet is received at an intermediate node, the node removes the record corresponding to this call from its call table. For a type 1 session, the rate field of all elements in the utilization list $\mathscr{L}_l$ is updated according to $\mathscr{L}\_rate := \mathscr{L}\_rate - G$, whereas for a type 2 session, the variable S is updated according to $S := S - G$.

During the holding time of the session, the source periodically transmits to the destination REFRESH messages that are copied by the intermediate nodes. A session is dropped due to a time-out when no REFRESH messages arrive at the node within a required time interval. The node then removes the record corresponding to the session from its call table. If the session is of type 1, the capacity allocated to it is released automatically when its duration expires, whereas if it is of type 2, the variable S is updated according to $S := S - G$.

3.3. Connection control actions performed at the nodes

In this section we outline the state parameters kept at each node for a call. The actions that a source, intermediate, or destination node must take are detailed in Appendix A. The parameters and

terminology that we use is similar to that found in [3].

In normal operation the source transmits every T ms a REFRESH(A) packet to the destination, which is copied by intermediate nodes, and is acknowledged by the destination with an ACK. The REFRESH(A) packets play an important role in providing robustness to link and node failures. They ensure that each node periodically learns about the status of an ongoing call, and can drop the call if the REFRESH(A) packet does not arrive within the required time. This guarantees that all reserved bandwidth is eventually released even in the presence of link and node failures, which we discuss in Appendix B. The REFRESH(A) packet is similar in concept to the validation cell used by Boyer and Tranchier in the FRP/DT protocol [32], or the REFRESH packets used by Cidon et al. in the connection establishment schemes for the PARIS network [3]. The REFRESH packet contains the rate A finally allocated to the call, so that if the COMMIT packet is lost an intermediate node still learns about the rate allocated to the call and can free any excess capacity reserved for this call during the setup phase. The ACK sent by the destination informs the source that the call path and the destination are functional. The record maintained for a call at each intermediate node includes an *activity-bit*, which is set to one when a REFRESH(A) packet arrives, and signifies that the connection is active.

The capacity reserved by the session is confirmed when the COMMIT packet arrives at the node, or, in case the COMMIT packet is lost, when the first REFRESH(A) packet arrives at the node.

At any time a call can be in one of the states *inactive*, *checking*, *pending*, *active*, or *recover*. When the call has not been initiated at a node, its state is *inactive*. During the time a node performs computations to determine whether the call should be accepted, the state of the call at that node is *checking*. After capacity is reserved for a call at a node, and while the node waits for a COMMIT packet (at an intermediate or destination node) or an ACK packet (at the source node) to arrive, the call is in state *pending*. After the COMMIT/ACK packet arrives and till the time that the call is terminated, the call is in state *active*. When the call terminates and the reserved capacity is freed, the call returns to state *inactive*. In addition, at an intermediate node a call can be in state *recover* when its associated SP is recovering from a failure. The state diagrams that describe the way a call is handled at the source, intermediate, and destination node are illustrated in Figs. 14–16 (in Appendix A).

A proof of correctness of the ERVC protocol, based on an elegant approach used by Cidon et al. [3], is given in [37], with appropriate modifications to handle timing information and the possibility of future reservations in the ERVC protocol, both of which were not issues in [3].

3.4. Delay considerations and handling timing uncertainties

The processing and queueing delays encountered by the control packets have to be small and have small variability for the ERVC protocol to function efficiently. In the Thunder and Lightning network, the control packets have preemptive priority over the data packets. Therefore, the total delay experienced by a control packet at a node consists of three components: the *processing delay* (which is the time required to process a control packet), the *communication queueing delay* (which is the time required to transmit all packets ahead of it in the control buffer), and the *processor queueing delay* (which is the waiting time for control packets ahead of it to be processed by the SP). To obtain estimates on these delay components, it is important to estimate the rate at which control packets are generated. If a typical session transmits a total of M Gbits and all links have capacity equal to C Gbit/s, a SETUP packet will be transmitted through a port of the switch every at least M/C seconds on the average. Therefore, the rate at which SETUP packets are handled by a network switch satisfies

$$\lambda_{\text{handled}}^{\text{SETUP}} \leq \frac{CK}{M}, \tag{4}$$

where K is the number of outgoing links at the switch. If a typical path has H links, the average number of setup packets that are generated per unit of time at a node and are successful in establishing a connection satisfies

$$\lambda_{\text{generated}}^{\text{SETUP}} = \frac{\lambda_{\text{handled}}^{\text{SETUP}}}{H} \leq \frac{CK}{MH}. \tag{5}$$

The SETUP packets will generate an equal number of ACK/REJ packets. Thus, in the Thunder and Lightning network, where each of the four ports of a switch has an associated processor, each of the four SP's of a node will handle of the order of 160 control packets per second. Our estimate of the processing delay per control packet (assuming a 20 MHz processor) is of the order of 15–20 μs (this would be the time required for the 20 MHz processor to perform an update of the utilization list), while the transmission time of a control packet (424 bits) is of the order of 11 ns. Therefore, the rate at which the control packets can be processed by a SP and the rate at which they can be transmitted are about 10^2 or 10^5 times larger, respectively, than the rate at which they arrive at a SP. Consequently, the queueing delay of a setup packet at a switch will be very small, and its total delay will be close to its processing delay of 15–20 μs, and its variability will be small.

Another cause of uncertainty is related to the estimation of propagation delays. For the Thunder and Lightning network, the length of a 1000 km path segment is expected to be known to within 1 km. As a result, the uncertainty in the propagation delay for coast-to-coast communications over a distance of 3000 km, will only be about 30 μs or about a thousand times less than the propagation delay. As the above calculations illustrate, the uncertainty in the delay experienced by a control packet in the Thunder and Lighting network will be very small when compared with the holding time of a session and the propagation delays present in the network. This justifies the description of the ERVC protocol that we gave in Sections 3.1 and 3.2, where we assumed that the delay suffered by a control packet is predictable.

We now explain how a node handles uncertainties and round-off errors in the various timing and delay components. As pointed out in Section 3.1, session durations will be recorded in milliseconds. Therefore, when processing the SETUP or COMMIT packet of a session the node will be required to round-off the values of the fields *ST* and *TO* of a SETUP packet, and the values $\mathscr{L}_l$*_time* in the utilization list $\mathscr{L}_l$. The rule that will be followed in such a case is that a node, when in doubt, will always underestimate the start time, that is it will always round-off the start time ST downward to the closest millisecond. Similarly, a node, when in doubt, will always overestimate the TO field and the $\mathscr{L}_l$*_time* field, that is, it will round-off the *TO* field of the SETUP packet and the $\mathscr{L}_l$*_time* field of the records in the utilization list $\mathscr{L}_l$ upward to the nearest millisecond. This ensures that capacity is always reserved starting at a time strictly before it is actually needed by the session, so that when the data packets of the session arrive at the node capacity will be there for them.

4. Performance results

In this section, we present performance results for the ERVC protocol, which provide a quantitative estimate of the efficiency gains of the protocol, and we compare its performance with that of IRVC or regular reservation schemes. Our results substantiate the following claims: that for the same offered load, the ERVC protocol provides a substantially lower probability P_{blk} of blocking new sessions than IRVC protocols do, and that for the same probability of blocking new sessions, the ERVC protocol utilizes links much more efficiently than IRVC schemes do, and can achieve link utilization close to unity.

We consider the simulation setup illustrated in Fig. 8. There are N sources that generate sessions as per a Poisson process of rate λ sessions per unit time. This traffic is routed through a link l with a

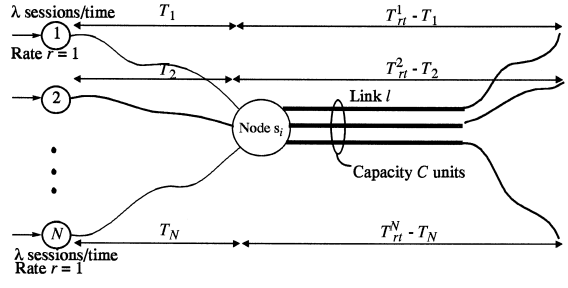


Fig. 8. The simulation setup used to obtain the performance characteristics for the ERVC protocol. For our simulation model, we chose $N = 20$ nodes, with link l of capacity $C = 10$ units. Each session in our model asks for $r = 1$ unit of capacity, and session holding times are exponentially distributed with mean $\overline{X} = 1$. The roundtrip delay T_{rt}^i for each source-destination pair was assumed to be the same.

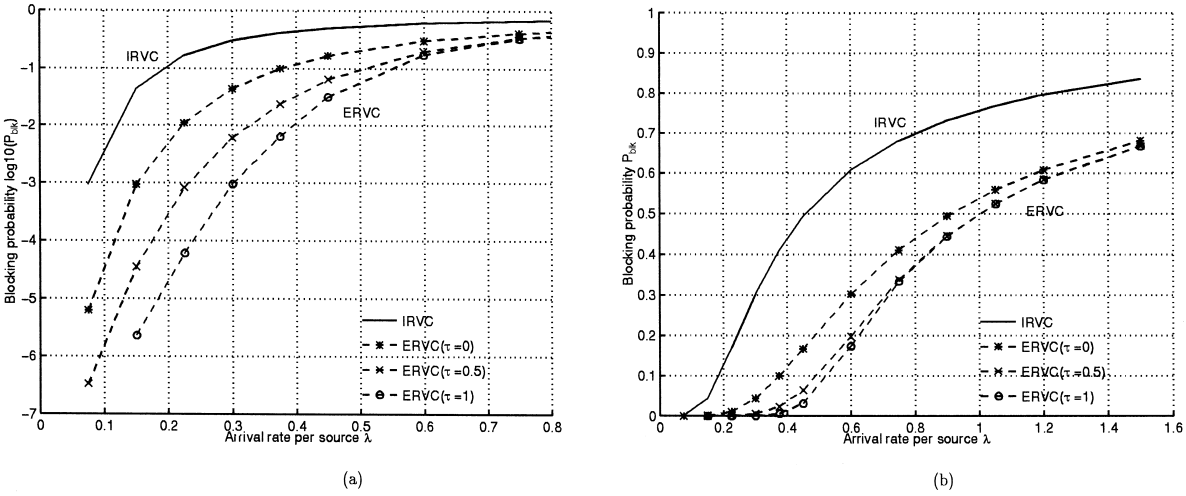


Fig. 9. The blocking probability P_{blk} for the ERVC protocol compared with that for IRVC protocols, when the delay tolerance τ of ERVC sessions is the parameter varied. The logarithmic scale on the left is used to better illustrate the improvements achieved by the ERVC protocol at light loads, while the linear scale on the right is used to highlight the details at heavy loads. Here $T_{rt} = 1.0$, and $\overline{X} = 1.0$.

capacity of C units located within the network, which we assume to be the only bottleneck link on the paths followed by the sessions. This is equivalent to assuming that the links leading from a source i to link l have ''infinite'' capacity and play no role in restricting the traffic. Upon its arrival at link l, each session demands r units of capacity, and is blocked (never to appear again) if adequate capacity is unavailable. The holding times of the sessions are exponentially distributed with mean $\overline{X}$. The roundtrip delay between source i and its destination is denoted by T_{rt}^{i}, while the propagation delay from source i to link l is denoted by T_i. Both delays are normalized by the mean session holding time $\overline{X}$.

We first compare the performance of the ERVC protocol with that of IRVC schemes, when the delay

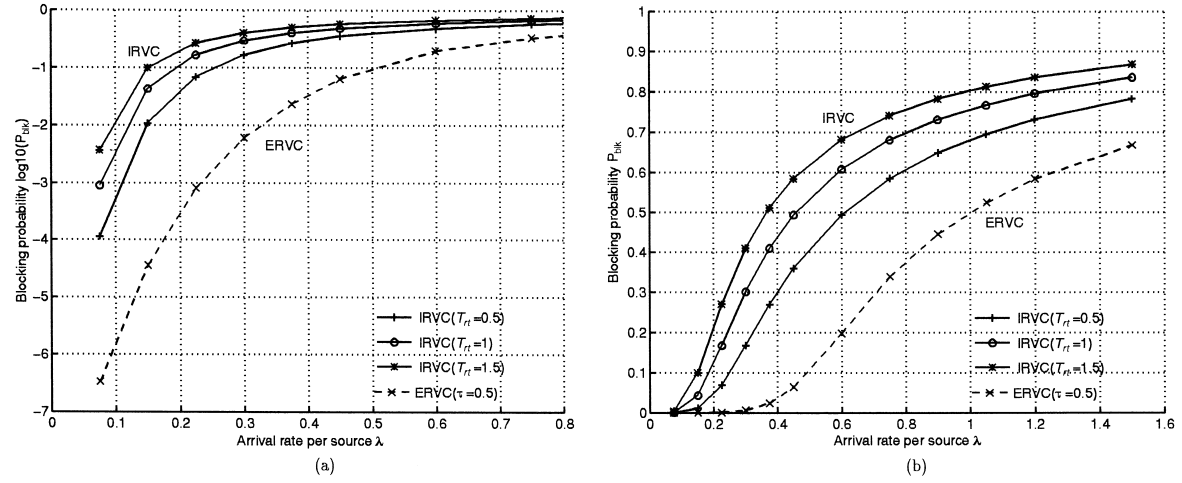


Fig. 10. Illustrates the blocking probability P_{blk} for the ERVC protocol and its comparison with the blocking probability for IRVC protocols, when the roundtrip delay T_{rt} is varied. Here $\tau = 0.5$, and $\overline{X} = 1.0$.

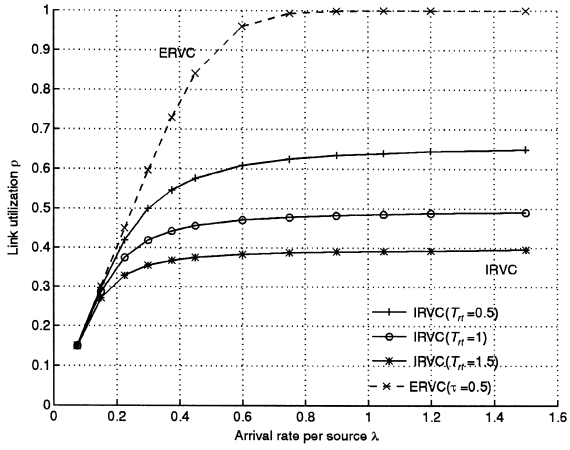


Fig. 11. Illustrates the link utilization for the ERVC protocol, and its comparison with the link utilization achieved by IRVC schemes, for varying roundtrip delay T_{rt}. As before, $\tau = 0.5$ and $\bar{X} = 1.0$.

τ (beyond the roundtrip delay T_{rt}) that ERVC sessions can tolerate before beginning transmission is the parameter that is varied. Fig. 9a and b illustrate the blocking performance of the protocol. The vertical scale in Fig. 9a is logarithmic to indicate the order of magnitude improvements achieved by the ERVC protocol over IRVC protocols for light loads, while the vertical scale in Fig. 9b is linear to better illustrate the improvements at heavy loads. We see that at light loads, say $\lambda \leq 0.4$, the blocking probability of the ERVC protocol improves by about an order of magnitude for every 0.5 increase in the delay tolerance τ of ERVC sessions. This indicates that the reservation ahead feature of the ERVC protocol can substantially improve performance, when the delay that can be tolerated is increased. At heavy loads, say $\lambda \geq 0.6$, the blocking probability of the ERVC protocol begins to saturate with increasing τ, and gradually approaches that for the case when $\tau = 0$.

We then compare the performance of the ERVC protocol (with a fixed delay tolerance τ) with IRVC protocols, for various values of T_{rt}. As expected, the performance of IRVC schemes worsens with increasing roundtrip delay. The performance of the ERVC protocol, however, does not depend on the roundtrip delay. This is because for a single link, like in the model considered here, a different roundtrip delay only means that the arrivals of sessions on link l are translated in time by a different amount. As a result, the picture in terms of load (and consequently the blocking) as seen by newly arriving sessions remains the same, irrespective of the roundtrip delay. From Fig. 10a, we observe that when the blocking probability P_{blk} for IRVC schemes reaches a nominal value of say 0.1, the blocking probability for the ERVC protocol is still at least two orders of magnitude better. By contrast, from Fig. 10b, we see that when the ERVC protocol reaches a blocking probability of $P_{blk} = 0.1$, the blocking probability of IRVC

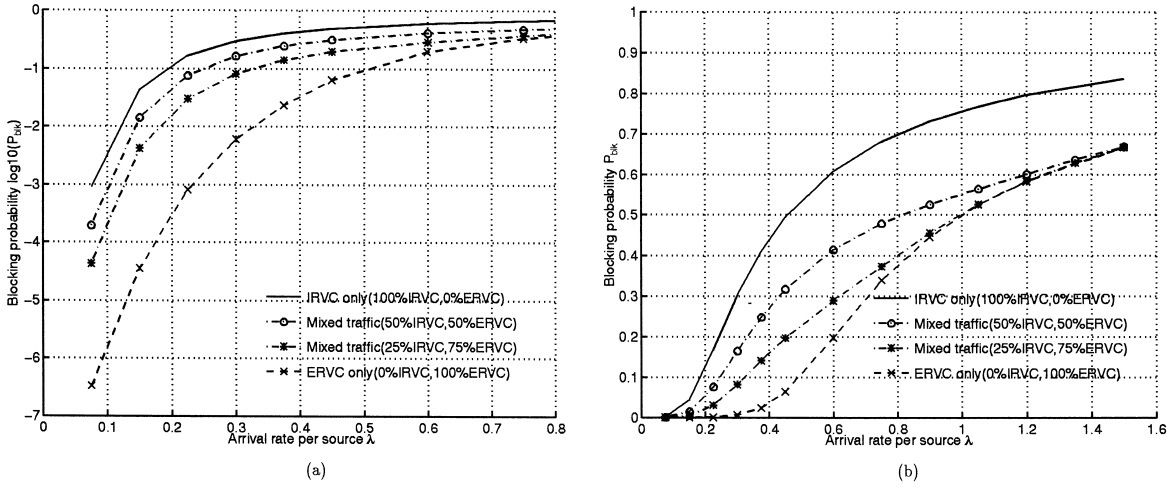
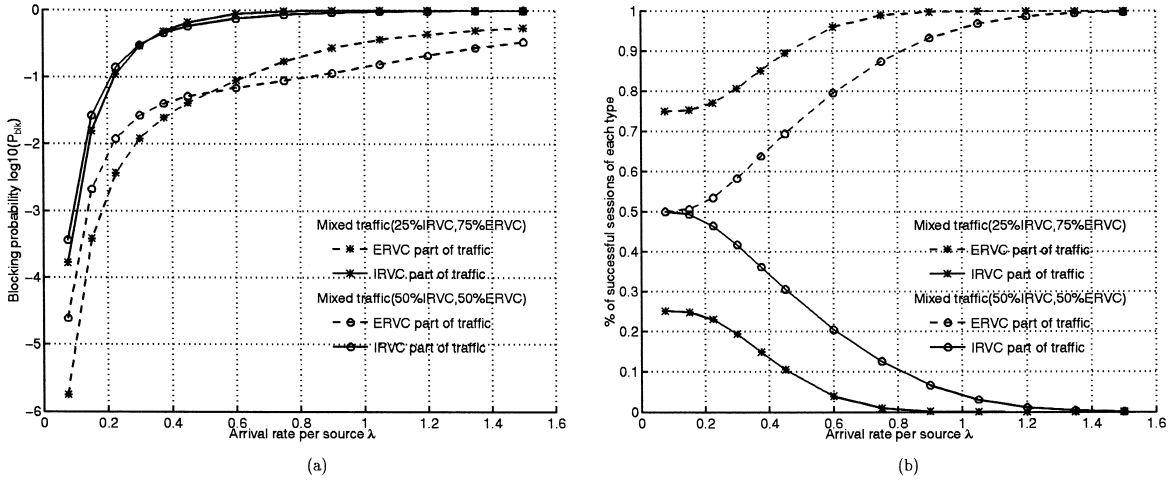


Fig. 12. Compares the blocking probability P_{blk} for the ERVC protocol with two types of mixed traffic, with that for IRVC schemes and the ERVC protocol. Here $\tau = 0.5$ and $\bar{X} = 1.0$.



(a) (b)

Fig. 13. Illustrates the blocking probability P_{blk} for the each of the two components of the traffic mixture whose performance is illustrated in Fig. 12. (b) Illustrates the percentage of sessions of each type among the total number of successful sessions, for both mixtures.

schemes is four times higher at $P_{blk} = 0.4$. Furthermore, Fig. 11 shows that the ERVC protocol then achieves a link utilization of about 0.90 (about $(T_{rt} + \overline{X})/\overline{X}$ times higher than than achieved by IRVC schemes, or about 56% higher than IRVC schemes for $T_{rt} = 0.5\overline{X}$, at $\lambda = 0.6$).

In Fig. 12a and b, we illustrate the performance of the protocol when we have mixed traffic, consisting of two session types: ERVC sessions, whose durations and roundtrip delays are known, and IRVC sessions, whose durations and roundtrip delays are unknown. We consider two types of mixed traffic: one in which 25% of the arriving sessions are IRVC sessions, and the other in which 50% of the arriving sessions are IRVC sessions.

The performance curves in Fig. 12a and b, can be viewed as representing the variation in performance of the ERVC protocol as the percentage of sessions with unknown characteristics varies from 100% (where all sessions are IRVC sessions) down to 0% (where all sessions are ERVC sessions). In this case also, we observe that at light loads the blocking performance of the ERVC protocol with mixed traffic (even with sessions of unknown durations constituting as much as 50% of the incoming traffic) is superior to that of IRVC schemes, and at higher loads it approaches that of the ERVC protocol, even though one would have expected it to lie in between that for IRVC schemes and that for the ERVC protocol (Fig. 13). This behavior can be understood by considering the individual blocking performance of each session type in the mixture and by looking at the percentage of successful sessions that belong to each session type. Fig. 13b shows that at low loads, the ratio of each session type in the sessions that are successful is the same as that of each session type in the arriving sessions. With increasing load, however, the ERVC sessions are treated better than IRVC sessions, and constitute the main fraction of the accepted traffic. As such, the percentage of ERVC sessions in the successful sessions gradually approaches one. This was expected since the ''reservation ahead'' feature of the ERVC protocol enables ERVC sessions to be more successful in reserving capacity than IRVC sessions.

5. Concluding remarks

We consider the ERVC protocol that we have presented a viable and competitive candidate for network protocols in future high-speed networks. Our aim was to design a protocol that would use the capacity efficiently in the presence of large propagation delays, minimize susceptibility to blocking, and avoid unnecessarily prolonged set-up phases. The performance characteristics of the protocol obtained via simulations demonstrate that the ERVC protocol

does particularly well in terms of utilizing capacity efficiently and that its blocking performance is significantly better than than of regular reservation schemes, indicating that the timed reservation of resources can contribute to considerably improved network operation in gigabit networks. We believe therefore that the protocol is particularly suited to take advantage of the long propagation delay in emerging very high-speed networks, and is suitable for several important applications in such networks.

Appendix A. Detailed connection control actions and protocol state diagrams

We will assume, as in [3], that there is a variable K_s such that out of K_s consecutive transmissions, at least one round-trip transmission is successful with high probability when the links are up. Thus, if the timer at the source expires $K_s + 1$ times without receiving an ACK, the source assumes that either the call path or the destination is down and drops the call. Similarly, if the timer at the destination expires $K_s + 2$ times without receiving a REFRESH(A) packet, the destination drops the call. If the timer at an intermediate node expires $K_s + 4$ times without receiving a REFRESH(A) packet, the intermediate node drops the call. (Note that because we do not assume that the network has bidirectional links, the ACK's are not constrained to follow the reverse path from that followed by the REFRESH and data packets, and therefore they cannot be used to directly convey information to the intermediate nodes.) The selection of these values for the time-out durations ensures that the source and destination always drop the call a sufficient time before the intermediate nodes. As a result, no data packets arrive at an intermediate node after it drops a call and frees the capacity reserved for it. The proof appears in [37].

As mentioned in Section 3.3, at any time a call can be in one of the states *inactive*, *checking*, *pending*, or *active*. In addition, at an intermediate node a

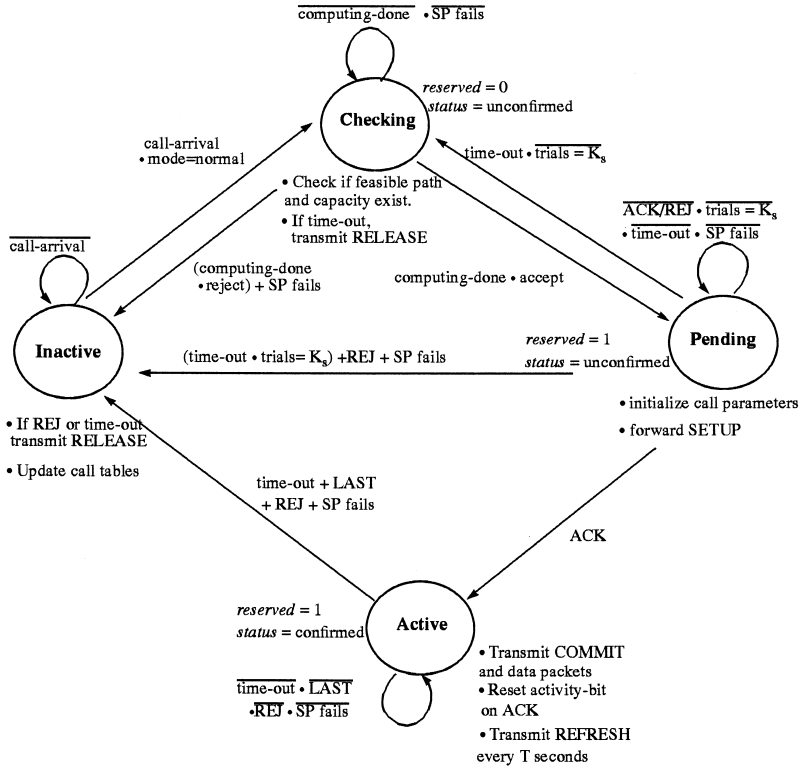


Fig. 14. Illustrates the state diagram for a call at the source node.

call can be in state *recover* when its associated SP is recovering from a failure. The state diagrams that describe the way a call is handled at the source, intermediate, and destination node are illustrated in Figs. 14–16, respectively, where the · denotes the logical AND operator and + denotes the logical OR operator. The overbar denotes logical negation.

When a call is initiated by the higher layers at a source node, the source transitions from state *inactive* to state *checking*, and sets *reserved* = 0 and (reservation) *status* = unconfirmed. If a path with adequate capacity (according to the information available at the source) is found, the source transmits a SETUP packet along that path, sets *reserved* = 1, and transitions to state *pending*. If the timer expires before receiving an ACK or a REJ, and less than K_s trials have been made to establish the connection, the source sets *reserved* = 0, transmits a RELEASE packet, and re-enters state *checking*. Another attempt is then made to establish the connection. When the number of unsuccessful attempts made by the source becomes equal to K_s, the source drops the call and enters state *inactive*. Upon reception of an ACK from the destination the source sets *reserved* = 1, and enters state *active*. At the same time, the rate granted field G is set equal to the allocated rate A, and the utilization list is updated as described in Section 3.2. The source waits for time $TO = \sum_{i=0}^{h} \delta_i$ specified in the *TO* field of the ACK, and then transmits the COMMIT(A, H) packet followed by the data packets. Every T seconds thereafter, the source transmits the REFRESH(A) packet. When the source receives an ACK for a REFRESH(A) packet, it sets the *activity-bit* of the call to one. At K_sT ms intervals, the source examines the activity-bit of all calls in progress, sets to zero all activity-bits that are equal to one, and drops those calls whose activity-bit is equal to zero. Thus, a source drops a call between

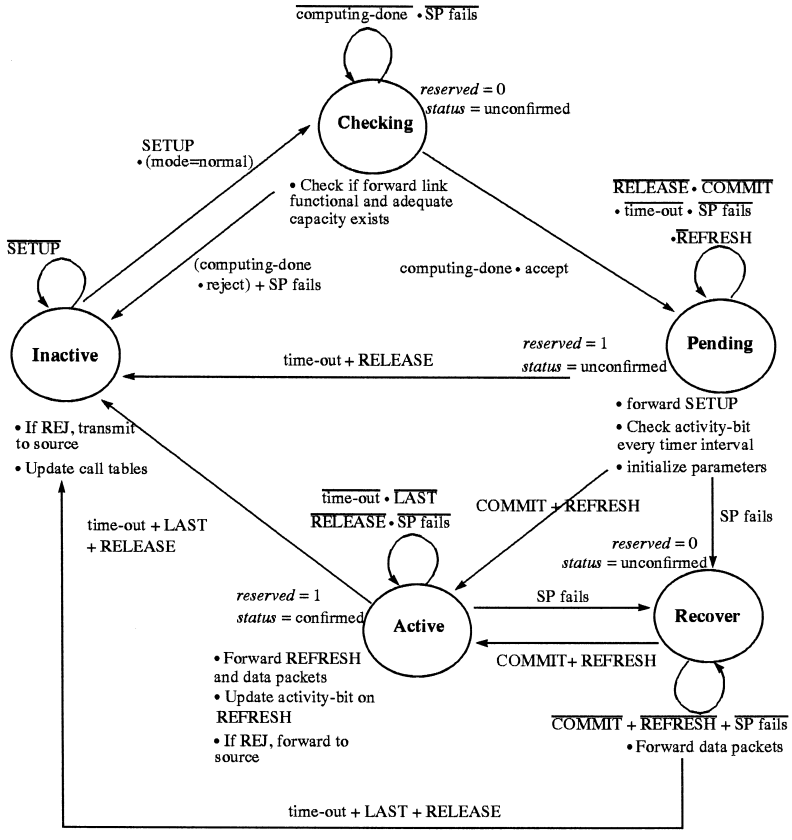


Fig. 15. Illustrates the state diagram for a call at an intermediate node.

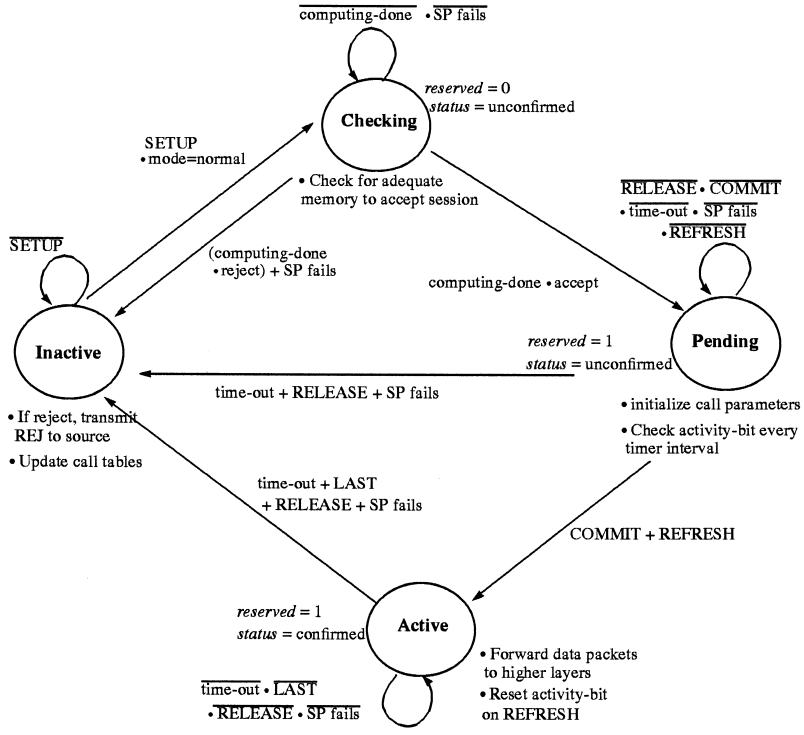


Fig. 16. Illustrates the state diagram for a call at the destination node.

K_sT and $2K_sT$ ms following the receipt of the last ACK from the destination. After all data packets of the session have been transmitted, the source transmits the LAST packet to free the resources along the path, and returns to state *inactive*. The call's record is deleted from the call table and the utilization list $\mathscr{L}_l$ is updated.

When the SETUP packet of a new session is received at an intermediate node, the call transitions from state *inactive* to state *checking*. The reservation procedure described in Section 3.2 is then followed to determine whether the call can be accepted or not. If the answer is negative, a REJ is transmitted to the source, and the call returns to state *inactive*. If the answer is positive, the node creates a record for the call, initializes the rate granted field G, sets $reserved = 1$, and transitions to state *pending*. The node forwards the SETUP packet (after updating it as described Section 3.2), and waits either for a COMMIT, a RELEASE, or a REFRESH packet, or for a time-out. If a COMMIT or a REFRESH packet arrives, the node sets $status = \text{confirmed}$ and updates G and the utilization list $\mathscr{L}_l$ as described in Section 3.2. The call then enters state *active*. Every time a REFRESH(A) packet arrives, the activity-bit of the call is set equal to one. The SP checks the activity-bit of all calls going through it every $(K_s + 4)T$ ms, drops those calls whose activity-bit is equal to zero, and sets to zero all activity bits that are equal to one. When the node receives a LAST or a RELEASE packet, or if it times out without receiving a REFRESH(A) packet, the procedure described in Section 3.2 is followed and the call returns to state *inactive*. For the sake of brevity, the actions that an intermediate node takes when the SP is down are omitted; the reader is referred to [37] for more details.

The actions at the destination node are similar to those at an intermediate node, except that the destination node checks for the availability of sufficient memory to store the data to be transmitted. If the call can be accepted, it transmits to the source an ACK packet that contains the rate A finally allocated to the call and the final value of the time offset TO.

Appendix B. Handling link and node failures

The REFRESH(A) packets play an important role in providing robustness to link and node failures. They ensure that each node periodically learns about the status of an ongoing call, and can drop the call if the REFRESH(A) packet does not arrive within the required time. This guarantees that all reserved bandwidth is eventually released.

When a source detects that one of its outgoing links has failed, it drops all calls using that link (whether in state *pending* or *active*) and stops transmitting REFRESH packets for them. All remaining nodes eventually drop the call due to time-out. If the source detects that a switch processor (SP) of the node ahead has failed, it drops all calls in state *pending* going through that port, and transmits a RELEASE packet for them. When an intermediate node detects that an outgoing link has failed, it transmits REJ packets to all sources using that link, but does not drop a call till it receives a RELEASE packet or till the call times out. If an intermediate node detects that the SP of the next node has failed, it sends a REJ packet to the sources of all calls in state *pending* (because these calls cannot transition to state *active* if the SP is not functional). The upstream nodes will drop the call due to a RELEASE packet or a time-out, while the downstream nodes will eventually drop the call due to a time-out.

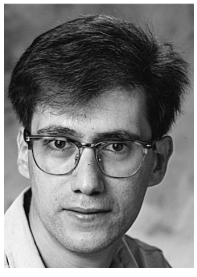
When a SP at the source or destination fails, all calls transition to state *inactive*, and the source or destination SP remains silent (sending no REFRESH or ACK packets, respectively) for the duration of the recovery period, thus ensuring that all calls operational at the time of the failure will be dropped by the time the SP enters normal mode. When the SP at an intermediate node fails, all calls in state *pending* or *active* routed through that port transition to state *recover* (see Fig. 15). When a SP recovers from a failure, it remains in *recovery* mode for a duration equal to $(K_s + 4)$T ms, and restores information on ongoing calls (which was lost when it failed), by recording the information from the REFRESH or COMMIT packets that it receives. It does not accept any new calls, and transmits a REJ packet for them. When the SP receives a REFRESH(A) or COMMIT packet for a session, the session enters state *active*, because the arrival of such a packet means that capacity had already been reserved for this call and that the call was already in or about to enter state *active*. If the COMMIT packet contains duration information, that is, the session is of type 1, the link utilization list is updated. Otherwise, the session is considered to be of type 2 and the variable S is updated.

When a session makes a reservation at an intermediate node for more than $(K_s + 4)$T ms in the future and the intermediate node fails, upon recovery the intermediate node will drop this call, because during the recovery period it receives no REFRESH(A) or COMMIT packet for this call as the source does not begin transmission by that time. One or more of the following options can be used to handle this situation. The first option is to keep a certain (very small) portion of the bandwidth reserved for the transmission of REFRESH packets, and let the source start transmitting REFRESH packets for a session immediately upon receiving the ACK(A, H) packet (even though the transmission of data packets begins only TO ms later). A second option is for the intermediate node to send back a REJ packet for sessions for which it receives data packets but has no capacity reserved, and drop the packets of such sessions.

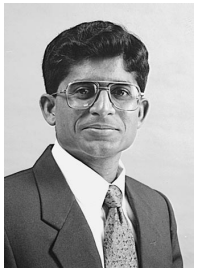
References

[1] L. Kleinrock, The latency/bandwidth tradeoff in gigabit networks, IEEE Commun. Mag. (April 1992) 36–41.
[2] C. Partridge, Protocols for high-speed networks: some questions and a few answers, Comput. Networks ISDN Systems 25 (9) (1993) 1019–1028.
[3] I. Cidon, I.S. Gopal, A. Segall, Connection establishment in high-speed networks, IEEE/ACM Trans. Networks 1 (4) (1993) 469–481.
[4] I. Cidon, I. Gopal, M.A. Kaplan, S. Kutten, A distributed control architecture of high-speed networks, IEEE Trans. Commun. 43 (3) (1995) 1950–1960.
[5] I. Cidon, I. Gopal, P.M. Gopal, R. Guérin, The PLANET/ORBIT high-speed network, J. High Speed Networks 2 (3) (1993) 171–208.
[6] D.D. Clark, B.S. Davie, D.J. Farber, I.S. Gopal et al., The AURORA gigabit testbed, Comput. Networks ISDN Systems 25 (6) (1993) 599–621.
[7] Corporation for National Research Initiatives, A brief description of the CNRI gigabit testbeds, CNRI Report, CNRI, Reston, VA, January 1992.

[8] M.N. Ransom, The VISTA net gigabit network, J. High Speed Networks 1 (1) (1992) 49–60.
[9] P. Messina, High-performance distributed computing applications, in: Proc. IISF/ACM Japan Int. Symp., 7–9 March 1994, Tokyo, Japan, p. 325.
[10] J.A. Terstriep, R.J. Minelli, T.T. Kwan, Experiences with a wide-area gigabit network, in: Proc. 1996 15th Int. Phoenix Conf. on Computers and Commun., 27–29 March 1996, Phoenix, AZ, pp. 179–187.
[11] C.A. Johnston, Architecture and performance of HIPPI-ATM-SONET terminal adapters, IEEE Commun. Mag. 33 (4) (1995) 46–51.
[12] R.A. Barry, V.W.S. Chan, K.L. Hall et al., All-optical network consortium – ultrafast TDM networks, IEEE J. Select. Areas Commun. 14 (5) (1996) 999–1013.
[13] B. Pehrson, P. Gunningberg, S. Pink, Distributed multimedia applications on gigabit networks, IEEE Networks 6 (1) (1992) 26–35.
[14] D.L. Tennenhouse and A.M. Leslie, A testbed for wide area ATM research, Comput. Commun. Rev. 19 (4) (1989) 182–190.
[15] D.J. Greaves, D.R. Milway, D. Garnett, A. Hopper, Design and implementation of an ATM backbone ring, in: Proc. COMPCON'92, 37th IEEE Computer Soc. Int. Conf., San Fransisco, CA, 24–28 February 1992, pp. 255–260.
[16] B. Butscher, J. Kanzow, The BERKOM project: a B-ISDN field trial in Berlin, in: Proc. 10th Int. Conf. on Computer Commun., New Delhi, India, 4–9 November 1990.
[17] M.H. Behringer, TEN-34 and JAMES: technical plans, in: Proc. JENC7, 7th Joint European Networking Conf., 13–16 May 1996, Budapest, Hungary, pp. 131–134.
[18] S. Butner, D. Skirmont, Architecture and design of a 40 gigabit per second ATM switch, in: Proc. Int. Conf. Computer Design (ICCD'95), Austin TX, October 1995.
[19] E.A. Varvarigos, V. Sharma, Lossfree communication in high speed networks, in: Proc. IEEE Sing. Int. Conf. on Networks, SICON'95, Singapore, 3–7 July 1995, pp. 230–236.
[20] E.A. Varvarigos, V. Sharma, The ready-to-go virtual circuit protocol: A loss-free connection control protocol for gigabit networks using FIFO buffers, IEEE/ACM Trans. Networks 5 (5) (1997) 705–718.
[21] S.E. Butner, R. Chivukula, On the limits of electronic ATM switching, IEEE Networks 10 (6) (1996) 26–31.
[22] M.C. St. Johns, D. Fisher, Survey of US gigabit-class network research, in: Proc. INET'94/JENC5, Annual Conf. of the Internet Society and 5th Joint European Networking Conf., Vol. 2, pp. 631–634.
[23] B.G. Kim, P. Wang, ATM network: goals and challenges, Commun. ACM 38 (2) (1995) 31–38.
[24] J.Y. Hui, Resource allocation for broadband networks, IEEE J. Select. Areas Commun. 6 (9) (1988) .
[25] H. Ohnishi, T. Okada, K. Noguchi, Flow control schemes and delay/loss tradeoff in ATM networks, IEEE J. Select. Areas Commun. 6 (9) (1988) 1609–1616.
[26] H. Suzuki, F. Tobagi, Fast bandwidth reservation scheme with multi-link and multi-path routing in an ATM network, in: Proc. INFOCOM'92, Vol. 3, Florence, Italy, May 1992, pp. 2233–2240.
[27] A. Iwata, N. Mori, C. Ikeda, H. Suzuki, M. Ott, ATM connection and traffic management schemes for multimedia networking, Commun. ACM 38 (2) (1995) 31–38.
[28] H. Luss, A model for advanced reservations for intercity visual conferencing services, Oper. Res. Quart. 28 (2) (1977) 275–284.
[29] H. Luss, A model for advanced reservations for large scale conferencing services, J. Opt. Res. Soc. 31 (1980) 239–245.
[30] J. Roberts, K. Liao, Traffic models for telecommunication services with advance capacity reservation, Comput. Networks ISDN Systems 10 (3–4) (1985) 221–229.
[31] J.T. Vitramo, A model of reservation systems, IEEE Trans. Commun. 40 (1) (1992) 109–118.
[32] P.E. Boyer, D.P. Tranchier, A reservation principle with applications to the ATM traffic control, Comput. Networks ISDN Systems 24 (4) (1992) 321–324.
[33] S. Fotedar, M. Gerla, P. Crocetti, L. Fratta, ATM virtual private networks, Commun. ACM 38 (2) (1995) 31–38.
[34] D.P. Tranchier, P.E. Boyer, Y.M. Rouaud, J.Y. Mazeas, Fast bandwidth allocation in ATM networks, in: Proc. Int. Switching Symp., Tokyo, Japan, 25–30 October 1992.
[35] R.J. Vetter, ATM Concepts, architectures and protocols, Commun. ACM 38 (2) (1995) 31–38.
[36] C. Hung, P. McKinley, Communication issues in parallel computing across ATM networks, IEEE Parallel Distrib. Technol. 2 (4) (1994) 73–86.
[37] V. Sharma, Efficient communication protocols and performance analysis for gigabit networks, Ph.D. dissertation, Dept. of Electrical and Computer Engineering, University of California, Santa Barbara, August 1997.



Emmanouel (Manos) Varvarigos received a B.S. (1988) in electrical engineering from the National Technical University of Athens, Greece and the M.S. (1990), Electrical Engineer (1991), and Ph.D. (1992) degrees in electrical engineering and computer science from the Massachusetts Institute of Technology. In 1990 he conducted research at Bell Communications Research, Morristown. He is currently an associate professor at the department of electrical and computer engineering at the University of California, Santa Barbara. His research interests are in the areas of protocols and architectures for high-speed data networks, parallel and distributed computation, and mobile communications. Dr. Varvarigos received the first panhellenic prize in the Greek Mathematic Olympiad, four Technical Chamber of Greece awards, and is a recipient of an NSF research initiation award.



Vishal Sharma received the B.Tech. degree (1991) in electrical engineering from the Indian Institute of Technology, Kanpur, and the M.S. degrees in computer engineering (1993) and in signals and systems (1993), and the Ph.D. (1997) degree in electrical and computer engineering from the University of California, Santa Barbara. In the summer of 1992, he worked at the Digital Technology Research Laboratory in Motorola's Corporate Research and Development Center, Schaumburg, IL, on the for real-time resizing of decompressed video sequences. His current research interests are in protocols, architectures, and performance analysis for terrestrial or satellite-based broadband networks and all-optical networks. He is active in the IEEE Computer and Professional Communication Societies, and is a student member of the ACM SIGCOMM and SIGDOC.