CONGRUENT SOFTWARE, INC.
98 Colorado Avenue
Berkeley, CA 94707
(510) 527-3926
(510) 527-3856 FAX

FROM: Peter Johansson

TO: IEEE P1394a Ballot Response Committee

DATE: March 30, 1999

RE: Subaction dual-phase retry protocol

This proposal is an editorial rework of the contributions made by David James and Farrell Ostler in 99-007r0 after the February BRC meeting. The essential content of their proposal has been placed in a format similar to that used by IEEE Std 1394-1995. In addition, I discovered that the number of actions in either a state diagram (this document) or a state transition table (James and Ostler) could be significantly reduced if the chosen point of view is the retry code received instead of the current preferred retry phase.

Also, I think the earlier IEEE Std 1394-1995 attempt to characterize the outbound behavior as a transaction layer state machine is misleading: the transaction layer (viewed as a whole) does not participate in the states previously described. What is really happening, on a packet by packet basis, is the selection of the retry code, *rt*, dependent upon the packet's "age" and the most recently received *ack_code*.

My editorial efforts in this document do not attempt to capture Farrell's suggestion that a *corrected* version of the 1995 state machine be published in addition to the *enhanced* state machine contained herein. I would appreciate feedback from the BRC before undertaking the editorial work. In my opinion, it is preferrable to recommend the newer, cleaner algorithm. Does this impose an unreasonable burden on link designers to validate whether or not their designs meet the requirements of the new state machine?

Last, there is essentially no difference between the outbound retry decision table for single- and dual-phase retry protocol. I suggest to the BRC that it might be a useful clarification if all of IEEE Std 1394-1995 clause 7.3.5, "Retry protocols", were replaced and not solely the dual-phase retry protocols described in this document. The information in figure 7-3 in the 1995 standard is so minimal that I think it is unnecessary (perhaps even confusing) to dress it up in the formalism of a state machine; I would recommend replacing it with a paragraph of text.

9.x Subaction dual-phase retry protocol

This clause replaces IEEE Std 1394-1995 clause 7.3.5.2, "Dual-phase retry protocol", in its entirety. Revision of both the outbound and inbound state machines are necessary for the following reasons:

— Reservation time limit ambiguities. IEEE Std 1394-1995 specifies that retry reservations shall be held by the intended recipient for four arbitration fairness intervals before cancellation. Unfortunately, the point from which fairness intervals is measured is not clearly specified. Contemporary link implementations are known to have different (although reasonable) interpretations which are not interoperable.
— No resynchronization. The inbound dual-phase retry state machines in IEEE Std 1394-1995 do not resynchronize reservation histories (phase and count of outstanding reservations) when discrepancies exist between the outbound and inbound nodes.

In order to correct these problems and to add an enhancement (the ability to count the number of outstanding reservations for each phase), this supplement redefines dual-phase retry behavior for both outbound and inbound nodes. The maintenance of reservation counters permits an inbound node to immediately accept subactions that lack a resource reservation if there are no reservations held for pending subactions.

9.x.1 Outbound subaction retry protocol

The specification in IEEE Std 1394-1995 of outbound retry behavior is somewhat confusing since it implies a transaction layer state machine when in fact no such state machine exists in the context of the entire transaction layer. More accurately, the transaction layer shall implement a decision table that selects a retry code, *rt*, each time an asynchronous primary packet is transmitted. The choice of retry code, specified by table 9-x, depends upon the packet's history, *i.e.*, both its "age" and the last acknowledge code received for the subaction.

Table 9-x — Outbound subaction retry code decision table

Subaction age	Prior acknowledge code	Retry code	
		Single-phase	Dual-phase
Oldest	—	*retry_X*	*retry_1*
	ack_busy_X		
	ack_busy_A	—	*retry_A*
	ack_busy_B	—	*retry_B*
Not oldest	—	*retry_X*	
	ack_busy_X *ack_busy_A* *ack_busy_B*		

Request and response retries and their associated reservations shall be processed independently of each other. At any point in time there may be both an oldest request and an oldest response.

The description of a subaction's age is not meant to imply the necessity for timers in a link design. When an asynchronous primary packet is available for transmission and there are no subactions awaiting retry, the packet is by definition the oldest packet and may be transmitted with a retry code of *retry_1*. When that subaction completes with a terminal acknowledge code (any acknowledgment, including *ack_pending*, other than *ack_busy_X*, *ack_busy_A* or *ack_busy_B*), another subaction awaiting transmission (or retransmission) may be designated oldest. The details as to which other subaction is elected oldest are implementation-dependent and are not important to the proper behavior of the retry protocols so long as the following requirement is observed:

> An asynchronous primary packet shall not be transmitted with a retry code of *retry_1* so long as a different subaction of the same type (request or response) once transmitted with a retry code of *retry_1* has not yet been completed with a terminal acknowledge code or been abandoned.

Subactions that do not yet have a history, *i.e.*, this is the first time transmission has been initiated, are indicated by the absence of a prior acknowledge code.

It is not necessary for the node transmitting a subaction to have *a priori* knowledge as to whether or not the intended recipient (inbound node) has implemented the single- or dual-phase retry protocol. Designs capable of dual-phase retry should select the initial retry code, *retry_X* or *retry_1*, from the right-hand column in table 9-x while designs that restrict themselves to single-phase retry use a retry code of *retry_X* in all cases.

In order for the dual-phase retry protocol to be able to guarantee forward progress, an outbound node should be capable of retransmission of a subaction within four fairness intervals; this is the period of time for which an inbound node guarantees a retry reservation. Although the inbound dual-phase retry protocol state machine resets itself properly if a reservation is not utilized within this time limit (see X for details), the outbound node may fail to make forward progress if it loses retry reservations because of delayed retransmission. Fairness intervals are counted from the receipt of the *ack_busy_A* or *ack_busy_B* that granted the reservation; if an outbound node is unable to retransmit the subaction before four arbitration reset gaps have been observed it may assume that the reservation has been cancelled.

9.x.2 Inbound subaction dual-phase retry protocol

The intended recipient of an asynchronous subaction, request or response, may be unable to accept the packet because of transient resource limitations: the node is busy. In a simple (single-phase) retry protocol, senders retransmit the subaction until resources are available or they abandon the transaction. Single-phase retry does not guarantee forward progress because it makes no attempt to reserve resources for the oldest subactions. The dual-phase protocol described in this clause reserves resources when congestion is encountered and keeps them reserved for particular subactions identified by a retry code. As subactions complete, the inbound node resources are once again made available to all subactions, with or without reservations.

The transaction layer shall allocate resources independently for request and response queues; this is necessary to prevent interdependent live-locks or starvation conditions. The dual-phase retry protocol specified by this clause shall be separately implemented for request and response subactions.

In the description that follows, the size of the reservation counter is implementation-dependent. These counters are unsigned numbers and shall not be decremented to a value smaller than zero nor incremented to a a value larger than $2^{n} - 1$,where *n* is the size of the counter, in bits. Consequently, C code notation employed in X for the sake of brevity is not strictly accurate. The expression `reservations[i]++` is correctly rendered by `reservations[i] += (reservations[i] < MAX_RESERVATIONS)`; in the same vein, `reservations[i] -= (reservations[i] > 0)` is the true expression of `reservations[i]--`.

The dual-phase retry protocol state machine concerns itself solely with nonbroadcast request or response packets received with both valid format and CRC, *i.e.*, the value of packet status in the LK_DATA.*indication* shall be GOOD. Transaction layer responses to packets with different packet status values are not part of the subaction retry protocol and are specified by IEEE Std 1394-1995.

The name *ack_subaction_done* refers collectively to any acknowledge code defined by IEEE Std 1394-1995 or this supplement except *ack_busy_X*, *ack_busy_A* or *ack_busy_B*.

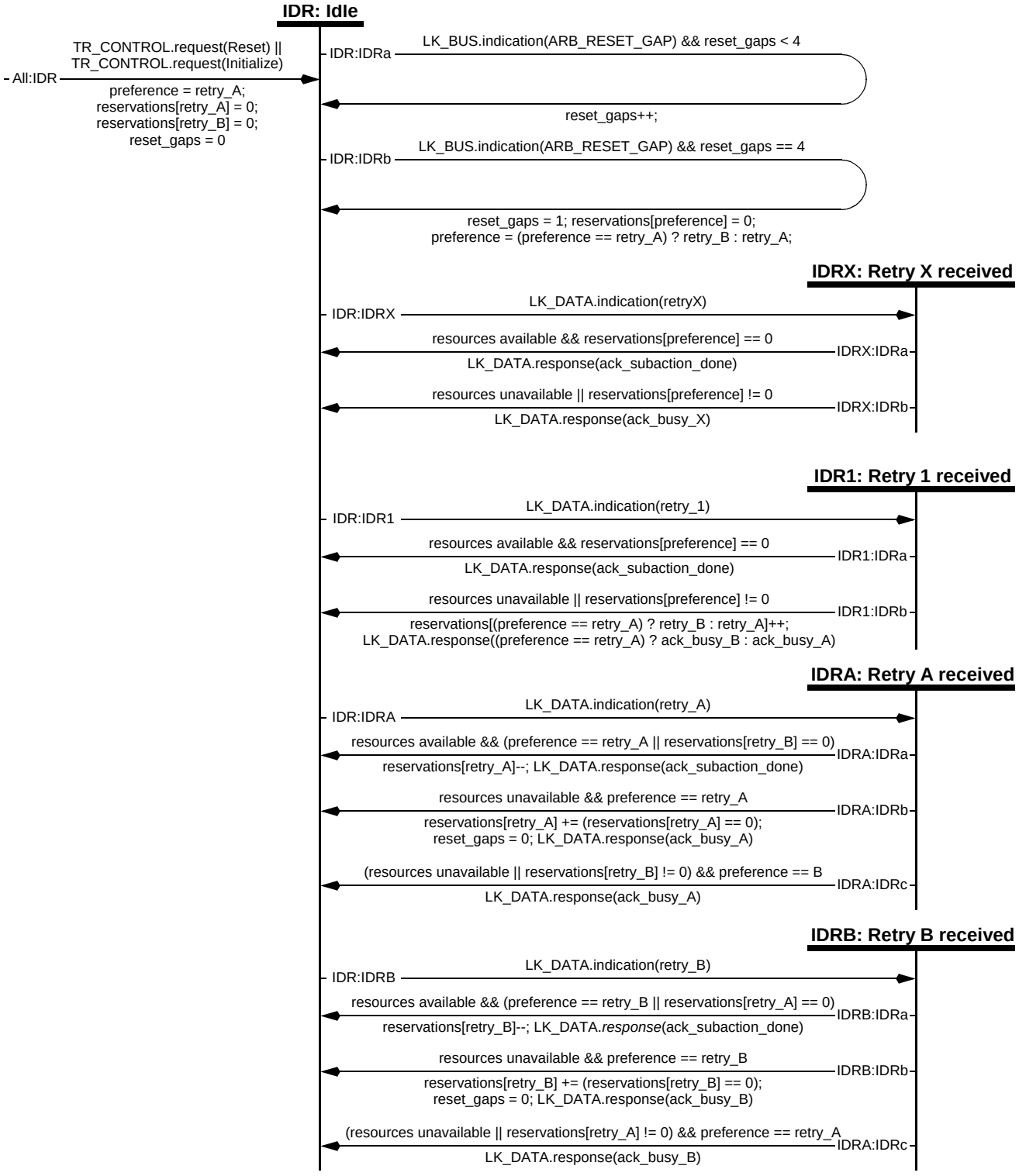

Figure 9-x — Inbound subaction dual-phase retry state machine

The state machine transitions are briefly described below.

Transition All:IDR. The receipt of a TR_CONTROL.*request* with an action of either Reset or Initialize shall cause the inbound dual-phase retry state machine to set its reservation preference to *retry_A* and zero all retry counters.

State IDR: Idle. The transaction layer is potentially ready to accept a request or response subaction from the link. Whether or not the subaction is serviced by the transaction layer will be determined by the availability of resources (such as FIFO space), outstanding reservation counts and the retry history of the subaction itself.

Transition IDR:IDRa. The end of a fairness interval has been detected, indicated by an arbitration reset gap. If the accumulated count of reset gaps is less than four, the transaction layer shall increment the count..

Transition IDR:IDRb. The end of the last fairness interval available to the outbound node for the retry of a subaction for which the inbound node granted a reservation. The inbound node abandons all reservations for the currently preferred retry phase, switches its preference to the opposite retry phase and counts this fairness interval as the first of the four available to holders of reservations in the now preferred phase.

Transition IDR:IDRX. A LK_DATA.*indication* has been received from the link with a packet status of GOOD and a retry code of *retry_X*.

Transition IDR:IDR1. A LK_DATA.*indication* has been received from the link with a packet status of GOOD and a retry code of *retry_1*.

Transition IDR:IDRA. A LK_DATA.*indication* has been received from the link with a packet status of GOOD and a retry code of *retry_A*.

Transition IDR:IDRB. A LK_DATA.*indication* has been received from the link with a packet status of GOOD and a retry code of *retry_B*.

State IDRX: Retry_X received. The outbound node (originator of the request or response subaction) has not requested a reservation if resources are unavailable to service the subaction. Attempt to process the subaction so long as resources are free and not allocated to a different subaction that holds a reservation.

Transition IDRX:IDRa. Resources are available and there are no reservations outstanding for the currently preferred phase. The transaction layer shall accept the subaction and return an appropriate terminal acknowledge code.

Transition IDRX:IDRb. Resources are unavailable or there are reservations outstanding for the currently preferred phase. The transaction layer shall refuse the subaction without creating a reservation.

State IDR1: Retry_1 received. The outbound node (originator of the request or response subaction) has requested a reservation if resources are unavailable to service the subaction. Attempt to process the subaction so long as resources are free and not allocated to a different subaction that holds a reservation. Otherwise, create a reservation in the opposite phase and indicate the phase of the reservation by means of the acknowledge code returned to the outbound node.

Transition IDR1:IDRa. Resources are available and there are no reservations outstanding for the currently preferred phase. The transaction layer shall accept the subaction and return an appropriate terminal acknowledge code.

Transition IDR1:IDRb. Resources are unavailable or there are reservations outstanding for the currently preferred phase. The transaction layer shall refuse the subaction but create a reservation for the opposite phase. The acknowledge code returned to the outbound node, either *ack_busy_A* or *ack_busy_B*, shall indicate the phase of the newly created reservation.

State IDRA: Retry_A received. The outbound node is attempting retransmission of an earlier request or response subaction for which the inbound node created a reservation. So long as resources are available, the inbound node grants preference to reservations earlier created for the current phase. Otherwise, if there are no outstanding reservations for the current phase, opposite phase retry attempts are serviced as resources permit.

Transition IDRA:IDRa. Resources are available and either *retry_A* is the currently preferred phase or else there are no reservations outstanding for the opposite phase. The transaction layer shall accept the subaction and return an appropriate terminal acknowledge code.

Transition IDRA:IDRb. Resources are unavailable and *retry_A* is the currently preferred phase. If the reservation count for *retry_A* is nonzero, there are reservations outstanding for the currently preferred phase. The transaction layer shall refuse the subaction; the *ack_busy_A* acknowledge code returned indicates that the outbound node should continue retry attempts in the same phase. If the *retry_A* reservation count is zero, the outbound and inbound state machines are out of synchronization with respect to each other; the inbound dual-phase state machine corrects this by incrementing the reservation count.

Transition IDRA:IDRc. Although a retry code of *retry_A* was received from the outbound node, the currently preferred retry phase is for *retry_B*. If either resources are unavailable or there are *retry_B* reservations outstanding, the transaction layer shall refuse the subaction but continue to hold a *retry_A* reservation for the outbound node.

State IDRB: Retry_B received. The outbound node is attempting retransmission of an earlier request or response subaction for which the inbound node created a reservation. So long as resources are available, the inbound node grants preference to reservations earlier created for the current phase. Otherwise, if there are no outstanding reservations for the current phase, opposite phase retry attempts are serviced as resources permit.

Transition IDRB:IDRa. Resources are available and either *retry_B* is the currently preferred phase or else there are no reservations outstanding for the opposite phase. The transaction layer shall accept the subaction and return an appropriate terminal acknowledge code.

Transition IDRB:IDRb. Resources are unavailable and *retry_B* is the currently preferred phase. If the reservation count for *retry_B* is nonzero, there are reservations outstanding for the currently preferred phase. The transaction layer shall refuse the subaction; the *ack_busy_B* acknowledge code returned indicates that the outbound node should continue retry attempts in the same phase. If the *retry_B* reservation count is zero, the outbound and inbound state machines are out of synchronization with respect to each other; the inbound dual-phase state machine corrects this by incrementing the reservation count.

Transition IDRB:IDRc. Although a retry code of *retry_B* was received from the outbound node, the currently preferred retry phase is for *retry_A*. If either resources are unavailable or there are *retry_A* reservations outstanding, the transaction layer shall refuse the subaction but continue to hold a *retry_B* reservation for the outbound node.

