Link Aggregation Control Protocol

Mick Seaman

While not reiterating basic concepts, this note attempts a complete description of the protocol, with the exception of flush mechanisms which are orthogonal to LACP itself.

This revision summarizes my understanding[1] of the P802.3ad Link Aggregation Control Protocol described in D1.0 and proposes some minor changes. The latter flesh out the previously agreed manual configuration defaults for Partner parameters. These were not fully incorporated into D1.0, as is highlighted by the Editor's Note on page 96 of the draft.

The proposed changes result in some simplification of the state machines, particularly of the periodic machine, as well as increased flexibility in protocol use. The information communicated by the protocol is unchanged.

The description given in Rev 1.6 and Rev 1.1 of this note has also been updated to reflect the management terminology and the 802.3 shared variable communication state machine conventions used in P802.3ad D1.0[2].

A separate note describes possible protocol extensions, outside the scope of the proposed standard, that ensure predictable protocol operation for links attached to shared media. Such links cannot occur if 802.3 standard equipment is used: the scope of the proposed link aggregation standard is limited to full duplex links, and there is no standardized full duplex repeater.

This note is not a ballot comment, but provides background for ballot comments in the usual form. Most of the commentary on proposed changes is provided as footnotes, to allow me to strip these out as and when ballot comments are resolved.

[1] Churn detection is missing from this description. Simply lack of time.

[2] This has had one particularly interesting effect. In the description in D1.0 and prior versions of P802.3ad, where signaling between state machines was event based, common signals 'LACPDU Received' and 'infoExpired' were used to signal to a number of machines. The use of common variables for signaling is less attractive, since confusion may arise as to which receiving machine should reset the variable, if that is required to allow for some delay in executing the necessary actions. In this revised description the receive machine and match logic together now signal 'not selected' to the selection logic, and 'matched, not matched' to the mux control machine. The periodic transmission machine operates directly on the relevant operational state variables and does not require any additional signalling variables. The signaling of 'need to transmit' through setting of the variable 'ntt' by various machines is unchanged – there is only one recipient of the signal, the transmission machine.

Protocol Participants

The protocol is described from the point of view of the physical ports, that represent a single point of attachment to a transmission medium. Each physical port that may be aggregated with other physical ports is a participant in the protocol.

Aggregation is modelled by the attachment of one or more physical ports to a Mux that distributes and collects frames from and to the aggregate port[3]. Aggregate ports represent a point of attachment of higher layer protocols.[4]

In the familiar case of individual links there is a trivial one to one correspondence between physical and aggregate ports which is so obvious that we do not distinguish them[5].

When it is clear that protocol exchanges between participants in separate systems are being discussed (rather than the aggregate behavior of participants in a single system) the term "participants" refers to the local participant, sometimes called the "**actor**" for clarity, and his remote "**partner**".

Protocol Data Units

A single message type, the LACPDU, is transmitted by protocol participants. It comprises the following information both for the transmitting actor, and its remote partner : the partner information being the actor's current view of its partners parameters.

- Port Number
- System ID
- Key
- Status

The Status information communicated comprises the following flags[6]:

- LACP_Activity
- LACP_Timeout
- Aggregate
- Synchronization
- Collecting
- Distributing

The **LACP_Activity** flag indicates a participant's intent to transmit periodically to detect and maintain aggregates. If set[7] the flag communicates **Active LACP**, if reset **Passive LACP**. A passive participant will participate in the protocol if it has an active partner.

The **LACP_Timeout** flag indicates that the participant wishes to receive frequent periodic transmissions[8], and will aggressively times out received information. If set the flag communicates **Short Timeout**, if reset **Long Timeout**.

The **Aggregation** flag indicates that the participant will allow the link to be used as part of an aggregate. Otherwise the link is to be used as an **individual link**, i.e. not aggregated with any other. This flag is set or reset as a consequence of local key management : the participant may know that the link has a unique key[9] and hence will not be aggregated. Signaling this information allows the receiving actor to skip protocol delays that are otherwise invoked to allow all links with the same system id and key combinations to be collected into one aggregate port without successive rapid changes to aggregate ports and accompanying higher layer protocol disruption.[10] If set the flag communicates **Aggregatable**, if reset **Individual**.

The **Synchronization** flag indicates that the transmitting participant's mux component is in sync with the system id and key information transmitted. This accommodates multiplexing

[3] I think I have a terminology problem with respect to D1.0, my 'aggregate port' is its 'aggregator' (this was Cisco's aggport I believe), my 'physical port' is D1.0's 'aggregate port'.

[4] So if you are in the aggregation 'layer' you look down through physical ports to the transmission medium and up through aggregate ports to the users of that medium.

[5] Most network protocols were originally designed to run over a single link layer access point, and as the requirement to provide transparent multiplexing over a number of links as emerged this distinction between 'user down' ports and 'provider up' ports has been widely introduced and is familiar to the designers and user of wide area protocols. Most routers provide a universal abstraction for this concept across the details of the particular multiplexing technology.

[6] It is proposed that these flags be encoded in a LAC PDU as bits in a transmitted octet. Since only 6 flags are currently defined, the treatment of the two 'spare' flags is explictly defined to facilitate interoperability and protocol upgradeability should that ever be desired. All the received flags are reflected into the actor's view of the partner's state.

[7] Strictly speaking we are discussing interpretation of the flag within the protocol entity here, not the encoding of LAC PDUs.

[8] Probably because it lacks confidence that its own hardware will indicate a disabled physical link quickly

[9] An alternative approach to explictly signaling "individual" would have been to allow the protocol machine to change the key to a reserved null value with the same semantics. However this blurs the network administrator's original intention (in setting the key value) with operational actions taken by the protocol entity (deciding a link is individual on the basis of its own information rather than having to consult with its partner). Using 'clever' encodings which have this blurring effect add nothing to protocol simplicity, transparency, or upgradability, and we are hardly short of the encoding space for one bit.

[10] Asserting "individual" is a potential exit route for the protocol machine in future scenarios and it is anticipated that it will be useful if an extension to shared media is ever standardized. Another reason not to confuse this functionality with the administrator assigned key.

hardware[11] that takes time to set up or reconfigure.[12] If set the flag communicate **In Sync**, if reset **Out of Sync**.

The **Collecting** flag indicates that the participant's collector, i.e. the reception component of the mux, is definitely on. If set the flag communicates **collecting**.

The **Distributing** flag indicates that the participant's distributor is not definitely off. If reset the flag indicates **not distributing**.

Protocol Machine

As an aid to understanding and analysis the protocol machine for each participant is partitioned into the following components:

- Receive Machine
- Match Logic
- Periodic Transmission Machine
- Selection Logic
- Mux Control Logic and Machine
- Churn Detection Machine
- Transmit Machine

The **Receive Machine** maintains partner information, recording protocol information from LACPDUs sent by remote partner(s). Received information is subject to a timeout, and if sufficient time elapses the receive machine will revert to using default partner information[13].

As the Receive Machine processes received LACPDUs, it uses[14] the **Match Logic** to determine if:

a) the participants have both agreed on the protocol information exchanged so that the physical port can be used in an aggregate

b) differences exist between the actor's protocol information and the partner's view of that information, requiring the actor to transmit a further LACPDU[15].

The **Periodic Transmission Machine** establishes the desire of the participants to exchange LAC PDUs periodically to maintain an aggregate, and how often periodic transmission should take place.

The **Mux Control Machine** attaches the physical port to an aggregate port[16], using the **Selection Logic** to choose an appropriate port, and turns the distributor and collector for the physical port on or off as required by protocol information.

The **Transmit Machine** formats and transmits LACPDUs as required by the Periodic Transmission Machine and by other machines if the partner's view of the actor's state is not current. It imposes maximum transmission rate limitations on LACPDUs.

Protocol Time Constants

The following time constants are specified for the protocol, and shall not be changed[17] by management or other means.

- Fast Periodic Time
  - 1 second
- Slow Periodic Time
  - 30 seconds
- Short Timeout Time
  - 3 seconds
- Long Timeout Time
  - 90 seconds
- Aggregate Wait Time[18]
  - 2 seconds[19]

[11] And software if the multiplexing is under the control of a separately scheduled software process communicating. with the actor's port based LACP protocol entity.

[12] While a principal goal of this protocol is ensuring high availability, that does not require that new links or physical ports be added to aggregates rapidly, simply that links or physical ports that have failed be removed in a timely fashion. Since the mechanism prompting addition will usually involve administrator intervention either at an administrative console or simply adding a physical link in a plug and play environment that is fortunate. This observation can be taken advantage of in structuring the protocol design and ensuring that it applies to the widest possible set of existing and new hardware and systems.

[13] The receive machine does not revert to default partner information immediately. This allows a physical port which has been "unplugged" (disabled) to continue to select the same aggregate port, minimizing disruption to higher protocol layers, particularly if the plug is put back in later allowing the port to resume its role in the aggregate. However while the port is unplugged the received information will expire so there is no danger of a "match" being reported on the basis of very old information from a protocol partner.

[14] Placing the Match Logic inside the Receive Machine avoids the need to keep a record of the partner's view of the actor's protocol parameters, the comparison being done on receipt. This is a change to the description in D1.0. It is possible to pull out the Match Logic, though the use of partner default information makes this more cumbersome than previously.

[15] a) and b) are not the same.

[16] Making the Mux Control Machine responsible for attaching and detaching the physical port from an aggregate port, as well controlling the collector and distributor, avoids having to duplicate state from a separate Selection Machine. This is a proposed change to D1.0. Here the selection logic, unchanged from D1.0, is invoked by the Mux Control Machine.

[17] The relationship of some of these values is important, and the aim is to promote interoperability. Moreover these parameters do not restrict the responsiveness of an implementation that focuses on high availability, not should they prove onerously quick.

[18] I believe this to be a more accurate name than 'Selection Wait Time', used in D1.0.

[19] Given as 5 seconds in D1.0. I can't recall the logic behind that but I now believe it is too long.

Protocol Variables

The following variables model an implementation of the protocol. Conformance to the protocol is purely in terms of observable protocol and management operations, so an actual implementation may hold the data described in any it chooses.

Aggregate Ports

The following variables are logically associated with each aggregate port. This note describes LACP implementations that have an aggregate port available for each physical port[20], and a single key value that applies to both the aggregate port and the physical port. For these implementations it is not necessary to record these variables separately from those recorded for the physical port.

- Aggregator MAC Address
- Aggregator Actor Port Priority[21] and Port Number
- Aggregator Actor System Priority and System ID
- Aggregator Actor Key
- Aggregator Partner Port Priority and Port Number
- Aggregator Partner System Priority and System ID
- Aggregator Partner Key
- Aggregator Aggregation Flag
  - True for an Aggregate that may include more than one physical port, False for an Individual link.

Physical Ports

The following variables are associated with each physical port.

- Actor's Port Priority and Port Number
- Actor's System Priority and System ID
- Actor's Operational Key
- Actor's Admin Key
- Actor's Operational Status[22]
- Actor's Admin Status
- Partner's Operational Port Priority and Port Number
- Partner's Operational System Priority and System ID
- Partner's Operational Key
- Partner's Operational Status
- Partner's Admin[23] Port Priority and Port Number
- Partner's Admin System Priority and System ID
- Partner's Admin Key
- Partner's Admin Status
- Receive Machine State : Rxm_current, Rxm_expired, Rxm_defaulted, Rxm_disabled
- selected : True if the Selection Logic has selected an aggregate port since the LAG ID last changed
- matched : True if the protocol partner has matched the actor's half of the lAG ID[24].
- aggregate : True if the link can be aggregated with others, False if it has to be an individual link.
- Aggregate Port : An identifier for the aggregate port that the physical port is attached to or desires to be attached to[25]. May be null if none is available.
- attach : True if Mux Control logic has instructed local system resources to attach to the aggregate port.
- attached : True if the local system resources have attached the physical port to the aggregate port.
- attach_when : Count of the number of seconds before the physical port may be attached to an aggregate port. Initial value Aggregate Wait Time[26].
- ntt : True if a transmission is required.
- hold_count[27] : The number of LACPDUs transmitted since hold_while was initialized, a new request for transmission is held if this is not less than the maximum number of transmissions allowed in a hold_while interval (one second)
- hold_while : Count of the number of seconds (initialized to 1 when started) of the number of seconds before the hold_count is reset.

[20] See the description of the Selection Logic and selection rules.

[21] This and the following variables comprise the LAG ID for the aggregate port.

[22] Note that the state of the Periodic Machine (see below) does not appear in this list. The actor's and partner's operational status hold all the information required.

[23] Referred to as 'Default' rather than 'Admin' in D1.0.

[24] Detailed description below.

[25] Should correspond to 30.7.2.9 aAggPortCurrentAggID in D1.0.

[26] Between them attach, attached, and attach_when encode D1.0 30.7.4.5 aAggPortDebugSelectionState.

[27] Together hold_count and hold_while limit the maximum LACPDU transmission rate as required by D1.0, however the particular way this is done should be left up to individual implementations.

Receive Machine and Match Logic

The Receive Machine extracts the following from each received LACPDU:

- LACPDU Actor Port Number[28]
- LACPDU Actor System ID
- LACPDU Actor Key
- LACPDU Actor Aggregate flag[29]

and compares[30] this information with that already recorded as the:

- Partner[31]'s Operational[32] Port Number
- Partner's Operational System ID
- Partner's Operational Key
- Partner's Operational Aggregate flag

These may have been extracted from a previously received LACPDU[33], or may be administrative defaults supplied by management of the actor. If the information has changed a new protocol partner has been detected and 'not selected' is signaled to the Selection Logic by setting the shared state variable[34] **selected** false[35,36].

[28] Throughout this description the Port Numbers and System IDs are treated as containing their priority components. This simplifies the description and the protocol, as there is no need to describe the priority components except in the management sections. It does have the effect that a change in a priority component is treated as if a different identifier was being used. Management changes of priority are envisaged to be so rare that this should not present a problem. It is important that this approach (or its converse) be made explicit in P802.3ad. As described here it would be possible to "borrow" from the priority space to enlarge the identifiers.

[29] Encoded as part of the Actor's State, see below.

[30] This comparison could be done later, by the Selection Logic, at the cost of maintaining a separate copy of this information for currently selected aggregate.

[31] The "partner" referred to here was the transmitter of the LAC PDU, so of course this the actor's information in the PDU.

[32] These values are referred to as "Operational" because they are the values used by the remainder of the protocol machine, and reflected back to the partner. Each "operational" value has a corresponding "Administrative" or "Admin" value. These hold the administrative defaults set by management. In the D1.0 management sections an incomplete mix of "Oper" and "Default" variables are present. I suggest we remove the latter and have a complete administartive set. This is consistent with the way that the proposed receive machine uses the administrative (values).

[33] Only information from the last LACPDU is ever recorded.

[34] As per 802.3 state machine conventions.

[35] This informs the Selection Logic that the partner is no longer using the previous LAG ID.

[36] This will cause the Mux Control Machine to detach the physical port from its current aggregate port, and (eventually) select a new aggregate appropriate for the new partner. This has to happen **after** the new protocol information is available. The description given here is adequate if the machines operate atomically (as per the 802.3 conventions), or sequentially completing the processing of each event before another machine runs.

The operational Port Number, System ID, Key, and Status information for the partner is then updated with the information from the LACPDU[37].

The Receive Machine extracts the following from each received PDU:

- LACPDU Partner Port Number
- LACPDU Partner System ID
- LACPDU Partner Key
- LACPDU Partner Aggregate Flag

and compares this information with that recorded as the:

- Actor's Operational Port Number
- Actor's Operational System ID
- Actor's Operational Key
- Actor's Operational Aggregate flag

If this comparison succeeds[38] or if the Partner's Operational Aggregate flag is false[39], the **Match Logic** considers the partner to have matched the actor's LAG (link aggregation group) identifier and signals that to the mux control logic by setting the shared state variable **matched**[40]. Otherwise matched is reset.

Additionally, if the partner Port Number, System ID, Key, or Status do not match the operational information held for the actor, 'need to transmit' is signaled to the **Transmit Machine** by setting the shared state variable **ntt** to true.

Following receipt of a valid LACPDU, the Receive Machine enters the **Current** state. The LACP Timeout flag in the Actor's Operational Status is set to the value in the Actor's Administrative Status[41], and the **current while** timer[42] started or restarted with an initial value[43] of **Short timeout** or **Long timeout** as appropriate.

If the current while timer expires, the Receive Machine enters the **Expired** state, and the

[37] This is the actor's information in the PDU, see a previous footnote.

[38] The protocol partner knows all about the actor so the actor can safely aggregate the link.

[39] The only way the link can be used is as an 'individual link' i.e. as an aggregate of one, so it does not matter if the partner has the actor's full details yet.

[40] D1.0 distinguishes a 'matched individual' from a 'matched aggregate', this is unneccessary since the aggregate selection provides the necessary selection. This simplification is made possible because the introduction of the administrative (default) parameters for the partner means there are always partner operational parameters, so the individual/aggregate distinction does not need to be captured by the match logic.

[41] Recovering, if necessary, from temporarily advertising Short timeout in the Expired state.

[42] The receive machine uses a single timer.

[43] Throughout this description timers are described as down counters which expire when they reach zero.

current while timer is restarted with an initial value of Short timeout. The Actor's Operational LACP Timeout flag is set to Short timeout[44], as is the Partner's[45]. The shared state variable 'matched' is reset to signal 'not matched' to the Mux Control Logic.

If the current while timer expires in the Expired state, the Receive Machine enters the **Defaulted** state. The Partner's Operational Port Number, System ID, Key, and Status are set to the Partner's Administrative values, and 'matched' is set.

Any change to the actor's or partner's operational LACP Active or LACP Timeout flags causes the Periodic Machine to reevaluate its current state[46].

The LACP Entity is generally managed by changing the actor's and partner's[47] Administrative rather than Operational parameters. The following are always updated immediately, so there is no need for visibily different administrative and operational states:

- Actor's Port Number[48]
- Actor's System ID[49]
- Actor's Operational Status[50] :
    - LACP_Activity
    - LACP_Timeout

[44] This should prompt the partner to transmit before the Rxm_expired state is left. This is most important on initialization.

[45] Ensuring that the actor will transmit rapidly throughout the short duration of the Expired state. Assuming that the partner's operational LACP_Timeout is short stimulates rapid transmission initially to discover a partner rapidly while allowing for a slow periodic rate in the steady state to ensure that no partner changes are missed.

[46] Since the 802.3 state machine conventions specify that a state block continually evaluates its exit conditions until one is specified (802.3-1998 Clause 21.5.1) no formal signaling to the Periodic Machine is required, the later merely needs access to the relevant data. However this subtlety might well be lost on software engineers unfamiliar with the 802.3 conventions, hence the above note. By the same argument, if the Receive Machine stored a complete copy of the received LACPDU there would be no need for any explicit inter machine signalling at all. Moreover since each machine is deemed to execute atomically there would be no need to guard sections where several shared variables are changed together. This is not a satisfactory basis for a trouble free specification for the target audience, more used to optimizing execution for use of a single sequential processor, or at the other extreme coping with preemptive interleaved execution.

[47] By which I mean the parameters local to the actor that claim to represent the Partner's Administrative state, not the parameters held by the partner itself.

[48] Including a priority component, which may the only part manageable by a network administrator.

[49] Including a priority component which may be the only part manageable by a network administrator.

[50] Although there is no need for separate administrative and operational values for these flags, it proves convenient to treat the State flags as a whole for management. Thus there are Operational and Administarive versions of the Actor's State.

The Actor's Operational Key may be varied by an implementation to deal with aggregation constraints not easily represented by a Key value. A simple implementation will keep Operational and Administrative values the same[51].

The Aggregation flag in the Actor's Status is not directly manageable but is a consequence of key management : it is reset if the port has a unique key. The Synchronization, Collecting, Distributing and reserved flags[52], are not manageable but are set as a consequence of protocol operation.

Following changes to the actor's Port Number, System ID, Operational Key, or Aggregation flag, 'not selected' is signaled to the Selection Logic, 'out of sync' to the Mux Control Logic, and 'need to transmit' to the Transmission Machine.

The following operational parameters are only updated from their corresponding administrative[53] versions if the Receive Machine is in the Defaulted state, and on first initializing the Receive Machine:

- Partner's Operational Port Number[54]
- Partner's Operational System ID
- Partner's Operational Key
- Partner's Operational Status[55] :
    - LACP_Activity
    - LACP_Timeout
    - Aggregation
    - Synchronization
    - Collecting
    - Distributing and Reserved bits[56]

[51] Such a simple implementation can still deal with the simpler form of aggregation constraints, such as having a maximum number of physical links in a given aggregate. A more complete description of constraints and rules for changing keys will be added to this document.

[52] Bits 6 and 7 of the State octet. This are transmitted as 0 in the Actor's state, unchecked on receipt, and reflected in the Partner State octet in LACPDUs.

[53] In P802.3ad D1.0 Clause 30 these are called xxxDefaultxxx while the operational versions lack the designattion Operational. Unless there is some established convention against such a use of administrative and operational I believe there it is clearer to use xxxOperxxx and xxxAdminxxx for these parameters.

[54] A default version of this parameter is missing from D1.0, it is required to resolve simple aggregation constraints and can serve as a useful check that the link leads to the expected port on the partner system.

[55] Allowing the administrative values of these flags to be set and brought into play in the Defaulted state offers a wider range of useful behaviors than specified in D1.0, including continuing or discontinuing LACP_Activity in the Defaulted state (requires that the Actors's LACP_Activity is False), transmitting LACPDUs on a frequent or an infrequent basis, and having the link be active in an aggregation or not. The Periodic Transmission machine is simplified since it just takes note of the current settings of LACP_Activity and LACP_Timeout and not of any changes in Receive Machine state.

If the physical port's MAC is disabled, the **Disabled**[57] state is entered. The previous Partner's Operational Port Number, System ID, Key and LACP_Activity are retained, but 'matched' is reset. The current_while timer is stopped if running. The Receive Machine is initialized in the Disabled state.

If the physical port's MAC is subsequently enabled, the Actor's Operational LACP Timeout flag is set to Short timeout[58], as is the Partner's, and the Expired state is entered, restarting the current_while timer with an initial value of Short timeout.

Receive State Machine Description

This section presents a more formal description of the Receive Machine, introduced informally above.

Receive Machine States and Timer

The receive machine has four states, other than those implied by the stored data previously described:

- Rxm_current
- Rxm_expired
- Rxm_defaulted
- Rxm_disabled

a single timer, the **current while** timer that is started in the Rxm_current and Rxm_expired states with an initial value[59] of either[60]:

- Short timeout

  or
- Long timeout

depending on the value of the Actor's Operational Status LACP_Timeout, as transmitted in LACPDUs.

Receive Machine Events

The following events can occur:

- participant created or reinitialized[61]
- received LAC PDU
- physical MAC enabled
- physical MAC disabled
- current while timer expiry

The physical MAC disabled event indicates that either or both of the physical MAC transmission or reception for the physical port associated with the actor have become non-operational.

The received LAC PDU event only occurs if both physical transmission and reception are operational, so far as the actor is aware[62].

Receive Machine Actions

The receive machine and match logic can take the following local actions:

- record partner operational parameters from received LACPDUs
- update partner operational parameters from partner administrative parameters
- set the partner and actor's operational LACP_Timeout to Short timeout
- set the partner and actor's operational LACP_Timeout to their administrative values
- start the current while timer
- reset the 'selected' shared state variable
- set or reset the 'matched' shared state variable
- signal need to transmit to the Transmit Machine by setting the ntt shared state variable.

[56] No actions are taken as the result of these but it is easier to view the update process as picking up the entire state thatn just parts of it since there is no difference. This allows for the fact that the rserved bits may have some future meaning.

[57] This was previously described as a variation on the Expired state with the current while timer not running, this seems a cleaner description, particularly bearing in mind the need to translate to 802.3 conventions.

[58] This should prompt the partner to transmit before the Rxm_expired state is left. This is most important on initialization.

[59] Throughout this description timers are described as down counters which expire when they reach zero.

[60] The values of these timeouts are set by the protocol specification; they are not changed by implementations or network administrators except to select the short or long value.

[61] This is a management event whose purpose is to restore the protocol entity to its initial state gracefully without dropping any loose ends. Typically protocol descriptions omit the specification of basic management operations thus inviting implementation problems. We attempt to avoid this mistake.

[62] This removes the need for the receive machine to explicitly track the physical MAC operational states.

Receive State Machine

	Rxm_current	Rxm_expired	Rxm_defaulted	Rxm_disabled
create or reinitialize, physical MAC disabled	current_while = Stopped, partner oper = partner admin, selected = False, matched = False Rxm_disabled			
physical MAC enabled	X			actor oper timeout = Short , partner oper timeout = Short, current_while = Short timeout, Rxm_expired
physical MAC disabled	current_while = Stopped, matched = False, Rxm_disabled			X
receivedLACPDU	if (pdu actor != partner oper) selected = False matched = (pdu partner == actor oper) || !pdu actor aggregation if (pdu partner != actor oper) ntt = True partner oper = pdu actor current_while = actor oper timeout = actor admin timeout Rxm_current			X
current_while timer expiry	actor oper timeout = Short , partner oper timeout = Short, current_while = Short_timeout, matched = False, Rxm_expired	if (partner oper != partner admin) selected = False, partner oper = partner admin, matched = True, Rxm_defaulted	X	

Periodic Transmission Machine

This machine establishes the desire of the participants to exchange LAC PDUs periodically to maintain an aggregate, and how often periodic transmission should occur.

The machine has three states, simply determined by the Actor's Operational LACP_Activity flag and the Partner's Operational LACP_Activity, and LACP_Timeout flags:

- No_Periodic transmission
- Fast_Periodic transmission
- Slow_Periodic Transmission

These control the use and initial value of a single timer, the **periodic transmission** timer. This stimulates periodic transmissions to ensure that the actor's information is not timed out by a protocol partner, and, if the actor is active, to discover a new partner.

Periodic exchanges[63] will take place if either participant so desires[64]. If both the actor and the partner are passive LACP it is No_periodic. Otherwise transmissions occur at a rate determined and communicated by the receiving participant (or assumed for such a participant). If the partner's is using a Short Timeout, the machine's state is Fast_Periodic, and the timer is restarted on expiry with an initial value of Fast Transmit. If the partner is using a Long Timeout, the machine's state is Slow Periodic, and timer's an initial value is Slow Transmit.

[63] This machine only governs periodic transmission. If management operations cause both participants to be passive, there may be exchanges that need to take place to move the physical port gracefully to an individual link. These will occur and LAC PDU transmissions will stop only when the configuration has reached a steady state.

[64] If the protocol were to be extended to shared media, periodic exchanges would take place if **any** participant so desired and at the fastest rate desired by any participant.

Selection Logic

The selection logic chooses the aggregate port for the physical port. It determines the Link Aggregation Group Identifier (LAG ID) for the physical port, and finds the[65] aggregate port with the same LAG ID.

If both the actor's and its partner's Operational Aggregation flag[66] are set, the LAG ID comprises the actor's Operational System ID and Key, and the partner's Operational System ID and Key[67]. However if the partner's System ID and Key are the same as the actor's, the link is always treated as 'Individual' as follows[68].

If either Aggregation flag is reset, communicating 'Individual', the LAG ID comprises the Actor's Port Number, System ID, and Key[69].

The default rules for aggregate port selection:

a) do not require additional MAC addresses to those provided by the physical MACs

b) are deterministic (history independent) in assigning physical ports to aggregate ports

c) should match the users' intuition in the trivial cases where individual links result[70].

They are also compatible with an alternative view of link aggregation as physical ports bonding together, rather than of physical ports attaching to aggregate ports.

These rules are not required by the protocol[71], which can accommodate greater flexibility in the relationship of aggregate and physical ports[72].

[65] There will never be more than one, although there may be none. Hopefully the latter case is temporary.

[66] By introducing administrative defaults we have ensured that operational information for the partner is always present, so there is no longer any need to call out 'Selected as Aggregate' or 'Selected as Individual' explicitly.

[67] By convention this is written with the participant with the lower numbered System ID first.

[68] Otherwise two ports on the actor, connected by a link, would be aggregated together and produce an accidental loopback.

[69] And the only aggregate port that can be selected according to the rules proposed below is that naturally associated with the physical port. In this case the 'global' LAG ID is the concatenation of this value with the partner information.

[70] Important when introducing the protocol when customers may be sceptical as to its value because it is not implemented in all attached devices. Counter intuitive behavior of systems conforming to the standard, but not providing additional functionality in this period, would be a significant negative.

[71] Except that interoperability requires that the same link s are held as 'standby' by two participants if both of them have constraints on the links that can be aggregated beyond those easily expressed by a key value.

[72] The default rules shall be implemented, additional rules that tradeoff determinism for greater resiliency are optional. As a practical matter, it is necessary to have one commonly understood set of rules, with the properties described, to ensure acceptance of link aggregation.

Under these rules:

- Each physical MAC has (comes equipped with) both a physical port and aggregate port.
- Every physical port always has one aggregate port selected at any point in time.
- A physical port that is operating as an individual link always selects, and has first claim on its own aggregate port.
- A number of physical ports in an aggregate always select the lowest numbered port[73] for their aggregate port. The corresponding physical port may not be in a state that allows data to be transferred on its physical link but it has selected that aggregate port.

The following diagrams illustrate the rules. Figure 1 shows a valid configuration and Figure 2 an invalid one.

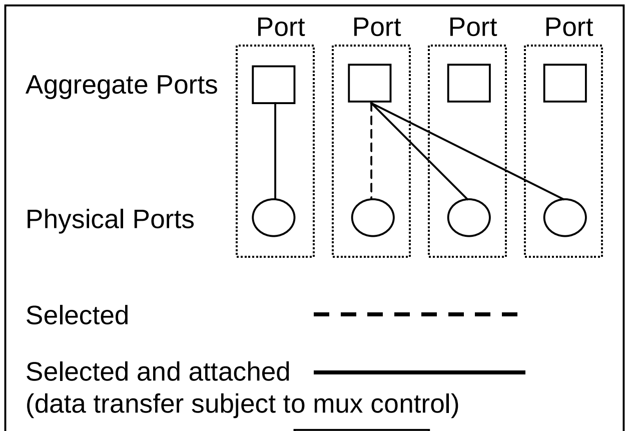

Figure 1

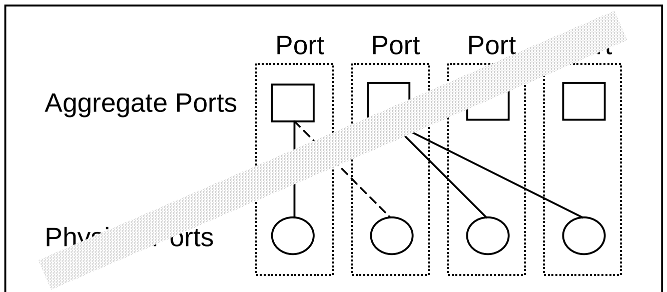

Figure 2

Where there are constraints on attachment, such as a maximum number of physical ports in an aggregate, the Selection Logic also determines the relative priority of the physical ports for attachment to the aggregate port

[73] An arbitrary rule of course.

When the selection logic has chosen an aggregate port it sets the shared state variable **selected** true to signal to the Mux Control Machine.

The Receive Machine will reset 'selected' if the LAG ID being communicated by the protocol partners changes[74]. This may be because a LACPDU with new partner information has been received, the default (administrative) partner parameters have been selected, or these or the actor's parameters have been changed by management.

If the selection parameters for a given physical port are changed, other ports in the system may have to reselect their aggregate ports[75].

The selection logic is invoked by the Mux Control machine, whenever a physical port is not attached to and has not selected an aggregate port[76].

Mux Control Machine

The mux control machine attaches and detaches a physical port to and from an aggregate ports, and turns the distribution and collection on or off as required by the selection and match logic.

LACP accomodates a wide range of possible implementations, system constraints, and local representations of mux state. However for protocol purposes an actor's operational state can be summarized in three flags:

- Synchronization
    - In_Sync, if the physical port is attached to the aggregate port last chosen by the selection logic and the state variable selected is still true
    - Out_of_Sync[77] otherwise
- Collecting, true if any user data frames received on the link will be collected and delivered to the user of the aggregate port
- Distributing[78], true if any user data frames transmitted by the user of the aggregate port may be transmitted on the link.

And the operation of the mux is best specified in terms of the goals for attaching and detaching, collecting and distributing, given the above and the output of the match logic.

If the actor's mux is Out_of_sync or the partner's match is Out of sync, then both collector and distributor should be turned off.

If the actor's mux is In_Sync and the partner's match is In_Sync, then the collector should be turned on.

If the actor's mux is In_sync, the partner's match is In_sync, and the partner's collector is turned on, then the distributor should be turned on.

If the mux hardware is **coupled**, i.e. forces the distributor to turn on when the collector is turned on then the above rules also apply.

If the mux hardware is **independent**, i.e. not coupled, then if the partner's collector is turned off, the distributor should be turned off[79].

If the actor is 'not selected'[80] but attached to an aggregate port, it shall be detached. A physical port shall not be attached to an aggregate port with attached physical ports that are 'not selected', i.e. are in the process of being detached[81,82]. The mux control machine may further delay reattachment of a physical port to a

[74]So it no longer corresponds to the previous selection. Selected is reset even if the same aggregate port will be chosen again, the receive machine cannot always know, not does it try to optimize particular cases. This means reselection will take place whenever the partner changes so it may be used as an indication to management functions concerned with peer authorization and related matters. To do otherwise would run the risk of link aggregation failing to preserve the desirable port down up characteristics of physical links.

[75] The search for other ports that may have to select the same aggregate can be narrowed significantly. Unless the local key has changed it can be restricted to those ports with a matching key. Further if the port whose parameters have changed was not the lowest numbered port in its previous selection and is not the lowest numbered in its new selection it will not affect the choice of aggregate port by other physical ports.

[76] When 'not selected' is signaled by the Receive Machine, the Mux Control Machine has to first turn off collection and distribution, and detach the physical port from the current aggregate port. That being done it can invoke the selection logic to select a (possibly) new aggregate port. The Mux Control Machine may delay a new attachment to minimize aggregate port user disruption. The wait_while timer described in D1.0 serves this function. This description moves it to the Mux Control Machine. Since its value is system dependent and does not affect protocol correctness (the Synchronization flag ensures that) it is probably better to allow the Mux Control machine to insert such a delay without specifying the precise mechanism and controls upon it in the standard.

[77] For example the hardware may not have yet responded to changed protocol information and be Out_of_Sync, but Collecting, in which case the partner better not transmit any frames because they will be misdelivered.

[78] Not required by the state machines described in this note, but strictly specified in case new machines may use this in the future, this information is in any case useful for debug.

[79] While graceful removal of a link from an aggregate is not currently specified this behavior supports managing that graceful removal from one end of the aggregate without having to invoke higher layer coordination.

[80] In which case it will be 'Out of Sync'.

[81] To prevent misdelivery. . This rule also ensures that a change of partner does bring the aggregate port down even if there are hardware switching delays which might allow new additions to the port just after a change of partner to keep the port up

[82] The machine thus accomodates hardware delays, though there is no requirement to delay artificially. Although a description at this level of detail is necessary to explain what the higher level protocol user of the aggregate port may see, and to validate the protocol's fitness for deployment across a wide range of hardware, this detail is not communicated in the protocol. This allows both "instantaneous" and more convoluted implementations to be accomodated by the protocol.

new aggregate port by up to **Aggregate wait delay** after the last selection of that aggregate port by any physical port[83,84].

The following table illustrates these rules for a hypothetical independent mux hardware implementation only capable of supporting one action at a time.

Partner Match		Actor's Mux			Actor's Target Mux State		Action
in_sync	collecting	in_sync	collecting	distributing	collectiing	distributing	
1	1	1	1	1	1	1	-
1	1	1	1	0	1	1	en. dist,
1	1	1	0	X	1	1	en. coll.
1	X	0	X	1	0	0	dis. dist.
1	X	0	1	0	0	0	dis. coll.
1	X	0	0	0	0	0	repair sync
1	0	1	1	1	1	0	dis. dist.
1	0	1	1	0	1	0	-
1	0	1	0	1	1	0	en. coll.
1	0	1	0	0	1	0	en. coll.
0	X	X	X	1	0	0	dis. dist.
0	X	X	1	0	0	0	dis. coll.
0	X	1	0	0	0	0	-
0	X	0	0	0	0	0	repair sync

Rearranged, to collect rows with the same action together this table becomes:

Partner Match		Actor's Mux			Action
in_sync	collecting	in_sync	collecting	distributing	
1	1	1	1	1	-
1	0	1	1	0	-
0	X	1	0	0	-
1	1	1	1	0	enable distributor
1	1	1	0	X	enable collector
1	0	1	0	1	enable collector*
1	0	1	0	0	enable collector
1	X	0	X	1	disable distributor
1	0	1	1	1	disable distributor
0	X	X	X	1	disable distributor
1	X	0	1	0	disable collector
0	X	X	1	0	disable collector
1	X	0	0	0	repair sync
0	X	0	0	0	repair sync

Transmit Machine

Need To Transmit

The Transmit Machine transmits a properly formatted LACPDU within a Fast transmit time following the signaling of a need to transmit from another protocol machine.

Hold Timer and Count

The transmit machine limits the maximum transmission rate of the protocol participant to no more than 3 LACPDUs[85] in a Fast transmit (one second) interval. If a need to transmit signal occurs when such limiting is in force the transmission is delayed.

[83] This is what the wait_while timer in D1.0 does.

[84] Thus minimizing thrashing of the higher layers due to the timing out of LACPDUs or the arrival of new information at slighlty different times. Avoiding thrashing is important since port up events may consume considerable numbers of buffers for initial protocol use.

[85] Sufficient to establish an aggregate in the worst case when initial LACPDUs from the participants cross.

Simulation

The following files comprise the core of a model implementation. So far this has been used to simulate a number of simple test cases, so no claims of correctness, fitness for purpose, warranty, support etd. are being made. However they may help the C-literate reader understand some of the implications of the LACP design.

```
/*        lac_types.h      1.10.000 07MAR99 03:22 */
#ifndef   lac_types_h__
#define   lac_types_h__

#include "sys.h"
#include "lac_options.h"
/******************************************************************************
 * LAC : LINK AGGREGATION CONTROL PROTOCOL : PROTOCOL DATA
 ******************************************************************************
 */
typedef int      System_priority;
typedef int      System_id;
        enum    {Null_system = (System_id)0};
        enum    {Null_port   = (Port_no)0};
typedef Port_no  Key;
        enum    {Null_key    = (Key)0};

typedef enum {Short_timeout = True, Long_timeout = False} Lacp_timeout;

typedef struct /* Lac_state */
{
   unsigned lacp_active     : 1;
   unsigned lacp_timeout    : 1;
   unsigned aggregation     : 1;
   unsigned synchronization : 1;
   unsigned collecting      : 1;
   unsigned distributing    : 1;
   unsigned reserved_bit6   : 1;
   unsigned reserved_bit7   : 1;
} Lac_state;

typedef struct /* Lac_info */
{
   Port_no         port_priority;

   Port_no         port_no;

   System_priority system_priority;

   System_id       system_id;

   Key             key;

   Lac_state       state;

} Lac_info;

typedef struct /* Lac_pdu */ /* only the relevant parameters, unpacked */
{
   Lac_info actor;

   Lac_info partner;

} Lac_pdu;

/******************************************************************************
 * LAC : LINK AGGREGATION CONTROL PROTOCOL : STATE MACHINE DATA
 ******************************************************************************
 */
typedef enum {Rxm_current, Rxm_expired, Rxm_defaulted, Rxm_disabled} Rx_machine;

/******************************************************************************
 * LAC : LINK AGGREGATION CONTROL PROTOCOL : SYSTEMS AND PORTS
 ******************************************************************************
 */
typedef struct lac_mac    Lac_mac;
typedef struct lac_port   Lac_port;
typedef struct lac_system Lac_system;

/******************************************************************************
 * LAC : LINK AGGREGATION CONTROL PROTOCOL : PHYSICAL PORT INSTANCES
 ****************************************************************************
 */
struct lac_port /* Lac_port */
{
   Lac_port       *next;
   Lac_system     *system;
       Port_no         port_no;

   Node            mux;
   Node            mac;

   Lac_info        actor;
   Lac_info        partner;
   Lac_info        actor_admin;
   Lac_info        partner_admin;
   Rx_machine      rxm;
   Ticks           current_while;
   Boolean         selected;
   Boolean         matched;
   Boolean         aggregate;

   Ticks           periodic_when;

   Lac_port       *aport;
   Lac_port       *alink;
   Boolean         operational; // aggregator may be taken out of service
   Boolean         attach;
   Boolean         attached;
   Boolean         standby;
   Ticks           attach_when; //wait before attaching to selected aggregate port

   Boolean         ntt;
   int             hold_count;
   Ticks           hold_while;
   Timer           tx_scheduler;

   Timer           tick_timer;

};
/****************************************************************************
 * LAC : LINK AGGREGATION CONTROL PROTOCOL : SYSTEM
 ****************************************************************************
 */
struct lac_system /* Lac_system */
{/*
  *
  */
       Lac_port  ports;

   System_priority priority;
   System_id       id;
};

#endif /* lac_types_h__ */
```

```
/*        lac_machines.h      1.10.000 03MAR99 03:21 */
#ifndef   lac_machines_h__
#define   lac_machines_h__

#include "sys.h"
#include "lac_options.h"

/******************************************************************************
 * LAC : LINK AGGREGATION CONTROL PROTOCOL : STATE MACHINES
 ******************************************************************************
 */
typedef enum /* Lac_event */
{
   Lac_null = 0,
   Lac_init,
   Lac_mgt,
   Lac_tick,
   Lac_pmac_on,             Lac_pmac_off,
   Lac_received,
   Lac_new_info,            Lac_update,
   Lac_attach,              Lac_attached,
   Lac_detach,              Lac_detached,
   Lac_enable_collector,    Lac_collector_on,
   Lac_disable_collector,   Lac_collector_off,
   Lac_enable_distributor,  Lac_distributor_on,
   Lac_disable_distributor, Lac_distributor_off,
   Lac_ntt,                 Lac_txd
} Lac_event;

extern void rx_machine(      Lac_port *port, Lac_event event, Lac_pdu *pdu);

extern void mux_control(     Lac_port *port, Lac_event event);

extern void hw_control(      Lac_port *port, Lac_event event);

extern void churn_detection( Lac_port *port, Lac_event event);

extern void tx_machine(      Lac_port *port, Lac_event event);
extern void tx_opportunity(  Lac_port *port);

#endif /* lac_machines_h__ */
```

```
/*            lac_rx.c   1.00.000 28FEB99 21:05 */

#include "lac_options.h"
#include "lac_types.h"
#include "lac_defaults.h"
#include "lac_machines.h"
/******************************************************************************
 * LAC : LINK AGGREGATION CONTROL PROTOCOL : RECEIVE, MATCH, & PERIODIC
 ******************************************************************************
 */
/*---------------------------------------------------------------------------*/
static void copy_info(Lac_info *from, Lac_info *to)
{/*
  */
   to->port_priority         = from->port_priority;
   to->port_no               = from->port_no;
   to->system_priority       = from->system_priority;
   to->system_id             = from->system_id;
   to->key                   = from->key;
   to->state.lacp_active     = from->state.lacp_active;
   to->state.lacp_timeout    = from->state.lacp_timeout;
   to->state.aggregation     = from->state.aggregation;
   to->state.synchronization = from->state.synchronization;
   to->state.collecting      = from->state.collecting;
   to->state.distributing    = from->state.distributing;
   to->state.reserved_bit6   = from->state.reserved_bit6;
   to->state.reserved_bit7   = from->state.reserved_bit7;
}
/*---------------------------------------------------------------------------*/
static Boolean same_info(Lac_info *a, Lac_info *b)
{/*
  */
   return (  (a->port_priority         == b->port_priority)
          && (a->port_no               == b->port_no)
          && (a->system_priority       == b->system_priority)
          && (a->system_id             == b->system_id)
          && (a->key                   == b->key)
          && (a->state.lacp_active     == b->state.lacp_active)
          && (a->state.lacp_timeout    == b->state.lacp_timeout)
          && (a->state.aggregation     == b->state.aggregation)
          && (a->state.synchronization == b->state.synchronization)
          && (a->state.collecting      == b->state.collecting)
          && (a->state.distributing    == b->state.distributing)
          && (a->state.reserved_bit6   == b->state.reserved_bit6)
          && (a->state.reserved_bit7   == b->state.reserved_bit7)
          );
}
/*---------------------------------------------------------------------------*/
static Boolean same_partner(Lac_info *a, Lac_info *b)
{/*
  */
   return (  (a->port_priority       == b->port_priority)
          && (a->port_no             == b->port_no)
          && (a->system_priority     == b->system_priority)
          && (a->system_id           == b->system_id)
          && (a->key                 == b->key)
          && (a->state.aggregation   == b->state.aggregation)
          );
}
/*---------------------------------------------------------------------------*/
extern void rx_machine(Lac_port *port, Lac_event event, Lac_pdu *pdu)
{/*
  */
   Boolean need_to_transmit = False;

   switch (event)
   {
   case Lac_init:
      copy_info(&port->partner_admin, &port->partner);
      port->selected      = False;
      port->matched       = False;
      port->standby       = False;
      port->rxm           = Rxm_disabled;
      port->periodic_when = Stopped;
      port->current_while = Stopped;
      /* continue - simulation does not turn pmac_on separately */

   case Lac_pmac_on:
      port->actor.state.lacp_timeout   = Short_timeout;
      port->partner.state.lacp_timeout = Short_timeout;
      port->current_while              = Short_timeout_ticks;
      port->rxm                        = Rxm_expired;
      break;

   case Lac_mgt:
      break;

   case Lac_pmac_off:
      port->current_while = Stopped;
      port->periodic_when = Stopped;
      port->matched       = False;
      port->rxm           = Rxm_disabled;
      break;

   case Lac_tick:
      if (port->current_while != Stopped)
      {   port->current_while--;
         if ((port->current_while == Expired) && (port->rxm == Rxm_current))
         {
   /* current_while timer expiry, Rxm_current */
      port->actor.state.lacp_timeout   = Short_timeout;
      port->partner.state.lacp_timeout = Short_timeout;
      port->current_while              = Short_timeout_ticks;
      port->matched                    = False;
      port->rxm                        = Rxm_expired;
         }
         else if((port->current_while == Expired) && (port->rxm == Rxm_expired))
         {
   /* current_while timer expiry, Rxm_expired */
      if (!same_partner(&port->partner, &port->partner_admin))
         port->selected = False;
      copy_info(&port->partner_admin, &port->partner);
      port->matched     = True;
      port->rxm         = Rxm_defaulted;
      }  }

      if (port->periodic_when != Stopped)
      {   port->periodic_when--;
         if (port->periodic_when == Expired)
         {
   /* periodic_when timer expiry */
      tx_machine(port, Lac_ntt);
      }  }
      break;

   case Lac_received:
      if (!same_partner(&pdu->actor, &port->partner))
         port->selected = False;

      port->matched =     same_partner(&pdu->partner, &port->actor)
                       ||(!pdu->actor.state.aggregation);

      if (!same_info(&pdu->partner, &port->actor))
         need_to_transmit = True;

      copy_info(&pdu->actor, &port->partner);
      port->actor.state.lacp_timeout = port->actor_admin.state.lacp_timeout;
      port->rxm = Rxm_current;

      if (port->actor.state.lacp_timeout == Short_timeout)
         port->current_while = Short_timeout_ticks;
      else
```

```
         port->current_while = Long_timeout_ticks;
      break;

   default:
      break;
   }

   if (port->actor.state.lacp_active || port->partner.state.lacp_active)
   {
      if ( port->periodic_when == Stopped)
           port->periodic_when = Slow_periodic_ticks;
      if ((port->partner.state.lacp_timeout == Short_timeout) &&
          (port->periodic_when > Fast_periodic_ticks))
           port->periodic_when = Fast_periodic_ticks;
   }
   else (port->periodic_when = Stopped);

   mux_control(port, Lac_new_info);
   /* churn_detection( port, Lac_received); */

   if (need_to_transmit)
      tx_machine(port, Lac_ntt);
}
/*---------------------------------------------------------------------------*/
```

```
/*            lac_mux.c   1.10.000 07MAR99 03:20 */

#include "lac_options.h"
#include "lac_types.h"
#include "lac_defaults.h"
#include "lac_machines.h"
/******************************************************************************
 * LAC : LINK AGGREGATION CONTROL PROTOCOL : MUX CONTROL & SELECTION LOGIC
 ******************************************************************************
 */
/*----------------------------------------------------------------------------*/
static Lac_port *find_aport(Lac_port *port)
{
   Lac_port *ap0    = &port->system->ports;
   Lac_port *ap     = &port->system->ports;
   Lac_port *best   =  port;

   while ((ap = ap->next) != ap0)
   {
      if(  (ap->actor.system_priority   == port->actor.system_priority     )
        && (ap->actor.system_id         == port->actor.system_id           )
        && (ap->actor.key               == port->actor.key                 )
        && (ap->partner.system_priority == port->partner.system_priority   )
        && (ap->partner.system_id       == port->partner.system_id         )
        && (ap->partner.key             == port->partner.key               )
        && (ap->aggregate               && port->aggregate                 )
        &&((ap->actor.port_priority     <  best->actor.port_priority )
          ||( (ap->actor.port_priority  == best->actor.port_priority )
            &&(ap->actor.port_no        <  best->actor.port_no       )
        ) ) )
        best = ap;
   }
   return(best);
}
/*----------------------------------------------------------------------------*/
static void select_aport(Lac_port *port)
{
   Lac_port *ap0    = &port->system->ports;
   Lac_port *ap     = &port->system->ports;

   port->aggregate =
             (port->actor.state.aggregation && port->partner.state.aggregation)
          && (port->actor.system_id != port->partner.system_id);

   port->aport = find_aport(port);

   if (port->aggregate)
      port->attach_when = Aggregate_wait_ticks;
   else
      port->attach_when = Now;

   port->selected = True;
   port->standby  = False;
}
/*----------------------------------------------------------------------------*/
static void select_standby_links(Lac_port *aport)
{/** This example implementation constrains the number of actively **/
 /** used links in an aggregate to two.                             **/

   Lac_port *p              = aport;
   Lac_port *ap             = aport;
   Boolean   local_priority = True;
   Lac_port *first_choice   = NULL;
   Lac_port *second_choice  = NULL;
   Port_no   first_priority, second_priority, check_priority;
   Port_no   first_port_no,  second_port_no,  check_port_no;

   if (  (   aport->partner.system_priority  < aport->actor.system_priority)
      || (  (aport->partner.system_priority == aport->actor.system_priority)
         && (aport->partner.system_id        < aport->actor.system_id      )
      )  )
      local_priority = False;

      first_priority = second_priority = 0xffffffff;
      first_port_no  = second_port_no  = 0xffffffff;

   do (p->standby = True); while ((p = p->alink) != aport);

   do
   {
      if (p->selected)
      {
         if (local_priority)
         {
            check_priority = p->actor.port_priority;
            check_port_no  = p->actor.port_no;
         }
         else
         {
            check_priority = p->partner.port_priority;
            check_port_no  = p->partner.port_no;
         }

         if (  (   check_priority <  first_priority)
            || (  (check_priority == first_priority)
               && (check_port_no  <  first_port_no )
            )  )
         {
            second_priority = first_priority;
            second_port_no  = first_port_no;
            first_priority  = check_priority;
            first_port_no   = check_port_no;
            second_choice   = first_choice;
            first_choice    = p;
         }
         else if (  (   check_priority <  second_priority)
                 || (  (check_priority == second_priority)
                    && (check_port_no  <  second_port_no )
                 )  )
         {
            second_priority = check_priority;
            second_port_no  = check_port_no;
            second_choice   = p;
   }  }  } while ((p = p->alink) != aport);

   if (first_choice  != NULL) first_choice->standby = False;
   if (second_choice != NULL) second_choice->standby = False;
}
/*------------------------------------------------------------------------*/
extern void mux_control(Lac_port *port, Lac_event event)
{/*
  */
   Lac_port  *p, *p0, *ap;
   Lac_state *as = &port->actor.state;
   Lac_state *ps = &port->partner.state;
   Lac_event  hw_event = Lac_null;
   unsigned   actor_sync;
   Boolean    need_to_transmit = False;

   switch (event)
   {
   case Lac_distributor_on:
      break;

   case Lac_distributor_off:
      as->distributing = False;
      /* need_to_transmit = True; */
      break;

   case Lac_collector_on:
      as->collecting = True;
      need_to_transmit = True;
```

```
         break;

      case Lac_collector_off:
         break;

      case Lac_tick:
         if (port->attach_when != Now)
            port->attach_when--;
         break;

      case Lac_init:
         port->selected  = False;
         port->aggregate = False;
         port->aport = port; port->alink = port;
         hw_control(port, Lac_init);
         as->distributing = as->collecting = False;
         port->attach = port->attached = port->selected = False;
         return;

      case Lac_detached:
         port->attach = port->attached = False;

         ap  = port->aport;

         if (port != port->aport)
         {/** remove from alink ring of current aport **/
            p = port->aport;
            while (p->alink != port) p = p->alink;
            p->alink    = port->alink;
            port->alink = port;

            port->aport = port;
         }

         if (ap->selected)
         {/** reevaluate standby selection for all ports attached to the    **/
          /** old aport, and send an event to each of these, since the      **/
          /** detaching port may have been the obstacle to new attachments. **/

            select_standby_links(ap);

            p = ap;
            do mux_control(p, Lac_update);
            while ((p = p->alink) != ap);
         }

         break;

      case Lac_attached:
         port->attached = True;
         break;

      case Lac_new_info:
         if (!port->selected)
         {/** check to see if any other ports will have to change their       **/
          /** port. This check  could be confined to ports with the same key  **/
          /** (before or after the information change) and to ports which     **/
          /** had previously selected this port as their aggregate port, or   **/
          /** will do so now.                                                 **/
            p = p0 = &port->system->ports;
            while ((p = p->next) != p0)
            {
               if (p->selected && (p->aport != find_aport(p)))
               {
                  p->selected = False;
                  mux_control(p, Lac_update);
         }  }  }
         break;

      case Lac_update:
         break;
```

```
default:
   break;
}

actor_sync = port->selected && !port->standby
          && port->attach   &&  port->attached;
if (as->synchronization != actor_sync)
{
   as->synchronization = actor_sync;
   need_to_transmit = True;
}

if (   port->matched  && ps->synchronization && ps->collecting
   &&  as->synchronization
   &&  as->collecting && !as->distributing)
{
       as->distributing = True;
       /* need_to_transmit = True; */
       hw_event = Lac_enable_distributor;
} else
if (   port->matched  && ps->synchronization
   &&  as->synchronization
   && !as->collecting)
{
       hw_event = Lac_enable_collector;
} else
if ( (!port->matched || !ps->synchronization || !ps->collecting
                     || !as->synchronization)
   &&  as->distributing)
{
       hw_event = Lac_disable_distributor;
} else
if ( (!port->matched || !ps->synchronization
   || !as->synchronization)
   &&  as->collecting)
{
       as->collecting = False;
       need_to_transmit = True;
       hw_event       = Lac_disable_collector;
} else
if ((!port->selected || port->standby) && port->attach && port->attached)
{
       port->attach = False;
       hw_event     = Lac_detach;
}

if (hw_event != Lac_null)
   hw_control(port, hw_event);

if (     !port->attach   && !port->attached
     &&  !port->selected
     &&  (port->alink == port)
   )
{/** now detached with no other ports attaching **/
   select_aport(port);
   if (port != port->aport)
   {/** insert into alink ring of selected aport, and reevaluate **/
    /** standby selection for all ports attached to the aport.   **/
      port->alink  = port->aport->alink;
      port->aport->alink = port;

      select_standby_links(port->aport);
      p = port->aport;
      do mux_control(p, Lac_update);
      while ((p = p->alink) != port->aport);
}  }

if (    !port->attach   && !port->attached
     &&  port->selected && !port->standby && (port->attach_when == Now)
   )
   {  /** check for detaching or waiting ports ... **/
      p = port->aport;
      while (  (p->selected) && (p->attach_when == Now)
            && (p = p->alink) != port->aport)
      {}
      if (p == port->aport) /* if none, attach all detached ports in alink */
      {
         do
         {
            if (!p->attach && !port->standby)
            {
               p->attach = True;
               hw_control(p, Lac_attach);
            }
         } while ((p = p->alink) != port->aport);
   }  }

   if (need_to_transmit)
      tx_machine(port, Lac_ntt);
}
/*---------------------------------------------------------------------------*/
```

```
/*     lac_tx.c   1.00.000 28FEB99 21:06 */

#include "sys.h"
#include "lac_options.h"
#include "lac_types.h"
#include "lac_defaults.h"
#include "lac_machines.h"
/******************************************************************************
 * LAC : LINK AGGREGATION CONTROL PROTOCOL : LACPDU TRANSMISSION
 ******************************************************************************
 */
static void copy_info(Lac_info *from, Lac_info *to)
{/*
  */
   to->port_priority         = from->port_priority;
   to->port_no               = from->port_no;
   to->system_priority       = from->system_priority;
   to->system_id             = from->system_id;
   to->key                   = from->key;
   to->state.lacp_active     = from->state.lacp_active;
   to->state.lacp_timeout    = from->state.lacp_timeout;
   to->state.aggregation     = from->state.aggregation;
   to->state.synchronization = from->state.synchronization;
   to->state.collecting      = from->state.collecting;
   to->state.distributing    = from->state.distributing;
   to->state.reserved_bit6   = from->state.reserved_bit6;
   to->state.reserved_bit7   = from->state.reserved_bit7;
}
/*----------------------------------------------------------------------------*/
static Boolean tx_lacpdu(Lac_port *port)
{
       Lac_pdu *pdu;

   if (sysmalloc(sizeof(Lac_pdu), &pdu))
   {
      copy_info(&port->actor,   &pdu->actor);
      copy_info(&port->partner, &pdu->partner);

      sys_tx(&port->mac, pdu);
      return(True);
}  }
/*----------------------------------------------------------------------------*/
```

```
extern void tx_machine(Lac_port *port, Lac_event event)
{
   switch (event)
   {
   case Lac_init:
      port->ntt = True;
      port->hold_while = Expired;
      port->hold_count = Zero;
      break;

   case Lac_ntt:
      port->ntt = True;
      break;

   case Lac_tick:
      if (port->hold_while != Expired)
      {
         if (port->hold_while-- == Expiring)
             port->hold_count    = Zero;
      }
      break;

   case Lac_txd:
      port->hold_while = Tx_interval_ticks;
      port->hold_count ++;
      port->ntt = False;
      break;

   default:
      break;
}   }
/*---------------------------------------------------------------------------*/
extern void tx_opportunity(Lac_port *port)
{
      if ((port->ntt) && (port->hold_count < Max_tx_per_interval))
      {
         if (tx_lacpdu(port))
            tx_machine(port, Lac_txd);
      }

   sys_start_timer(&port->tx_scheduler, Lac_tx_scheduling_ticks);
}
/*---------------------------------------------------------------------------*/
```

Preliminary Test Scenarios and Results

Test Scenario 1

```
static void test1()
{
   Lac_system *s1;
   Lac_port   *s1p1, *s1p2, *s1p3, *s1p4;
   Lac_system *s2;
   Lac_port   *s2p1, *s2p2, *s2p3, *s2p4;
   Lan        *lan1, *lan2, *lan3, *lan4, *lan5;

   printf("********* TEST 1 *************\n");

   (void)sys_create_lan(&lan1);
   (void)sys_create_lan(&lan2);

   lact_init_tester(t1);
   lact_init_tester(t2);

   sys_attach_lan_node(lan1, t1);
   sys_attach_lan_node(lan2, t2);

   lac_create_system(707, &s1);

   lac_create_port(s1, 1);
   s1p1 = s1->ports.next;

   lact_user_connect(s1p1);

   lac_create_port(s1, 2);
   s1p2 = s1p1->next;

   lact_user_connect(s1p1);

   lact_lan_attach(lan1, s1p1);
   lact_lan_attach(lan2, s1p2);
   ticks(100 * Lac_ticks);
}
```

Test Results : Scenario 1

```
********* TEST 1 *************
   0.1:TX at 707.1 Actor   1.1:1.707.1 Active   Nervous  Aggregate  In sync
                   Partner 1.1:1.  0.1 Passive  Nervous      .      In sync  Collecting  Distributing

   0.1:TX at 707.2 Actor   1.2:1.707.1 Active   Nervous  Aggregate  In sync
                   Partner 1.2:1.  0.2 Passive  Nervous      .      In sync  Collecting  Distributing

   1.0:TX at 707.1 Actor   1.1:1.707.1 Active   Nervous  Aggregate  In sync
                   Partner 1.1:1.  0.1 Passive  Nervous      .      In sync  Collecting  Distributing

   1.0:TX at 707.2 Actor   1.2:1.707.1 Active   Nervous  Aggregate  In sync
                   Partner 1.2:1.  0.2 Passive  Nervous      .      In sync  Collecting  Distributing

   2.0:TX at 707.1 Actor   1.1:1.707.1 Active   Nervous  Aggregate  In sync
                   Partner 1.1:1.  0.1 Passive  Nervous      .      In sync  Collecting  Distributing

   2.0:TX at 707.2 Actor   1.2:1.707.1 Active   Nervous  Aggregate  In sync
                   Partner 1.2:1.  0.2 Passive  Nervous      .      In sync  Collecting  Distributing

   3.0:TX at 707.1 Actor   1.1:1.707.1 Active   Nervous  Aggregate  In sync  Collecting  Distributing
                   Partner 1.1:1.  0.1 Passive     .         .      In sync  Collecting  Distributing

   3.0:TX at 707.2 Actor   1.2:1.707.1 Active   Nervous  Aggregate  In sync  Collecting  Distributing
                   Partner 1.2:1.  0.2 Passive     .         .      In sync  Collecting  Distributing

  33.0:TX at 707.1 Actor   1.1:1.707.1 Active   Nervous  Aggregate  In sync  Collecting  Distributing
                   Partner 1.1:1.  0.1 Passive     .         .      In sync  Collecting  Distributing

  33.0:TX at 707.2 Actor   1.2:1.707.1 Active   Nervous  Aggregate  In sync  Collecting  Distributing
                   Partner 1.2:1.  0.2 Passive     .         .      In sync  Collecting  Distributing

  63.0:TX at 707.1 Actor   1.1:1.707.1 Active   Nervous  Aggregate  In sync  Collecting  Distributing
                   Partner 1.1:1.  0.1 Passive     .         .      In sync  Collecting  Distributing

  63.0:TX at 707.2 Actor   1.2:1.707.1 Active   Nervous  Aggregate  In sync  Collecting  Distributing
                   Partner 1.2:1.  0.2 Passive     .         .      In sync  Collecting  Distributing

  93.0:TX at 707.1 Actor   1.1:1.707.1 Active   Nervous  Aggregate  In sync  Collecting  Distributing
                   Partner 1.1:1.  0.1 Passive     .         .      In sync  Collecting  Distributing

  93.0:TX at 707.2 Actor   1.2:1.707.1 Active   Nervous  Aggregate  In sync  Collecting  Distributing
                   Partner 1.2:1.  0.2 Passive     .         .      In sync  Collecting  Distributing

test over
```

Test Scenario 2

```
static void test2()
{
   Lac_system *s1;
   Lac_port   *s1p1, *s1p2, *s1p3, *s1p4;
   Lac_system *s2;
   Lac_port   *s2p1, *s2p2, *s2p3, *s2p4;
   Lan        *lan1, *lan2, *lan3, *lan4, *lan5;

   Node        tester1, tester2;
   Node       *t1 = &tester1;
   Node       *t2 = &tester2;

   printf("********* TEST 2 *************\n");

   (void)sys_create_lan(&lan1);
   (void)sys_create_lan(&lan2);

   lact_init_tester(t1);
   lact_init_tester(t2);

   sys_attach_lan_node(lan1, t1);
   sys_attach_lan_node(lan2, t2);

   lac_create_system(707, &s1);

   lac_create_port(s1, 1);
   s1p1 = s1->ports.next;

   lact_user_connect(s1p1);

   lac_create_port(s1, 2);
   s1p2 = s1p1->next;

   lact_user_connect(s1p1);

   lact_lan_attach(lan1, s1p1);
   lact_lan_attach(lan2, s1p2);
   tx_test_pdu(&tester1,
      /** actor’s parameters and state **/
      1 /* port_priority   */,     5 /* port_no   */,
      1 /* system_priority */,   808 /* system_id */,
      5 /* key     */,
      1 /* active  */, 1 /* short_timeout */, 1 /* aggregate    */,
      1 /* in_sync */, 1 /* collecting    */, 1 /* distributing */,

      /** partner’s parameters and state **/
      1 /* port_priority   */,     6 /* port_no   */,
      1 /* system_priority */,   707 /* system_id */,
      5 /* key     */,
      1 /* active  */, 1 /* short_timeout */, 1 /* aggregate    */,
      1 /* in_sync */, 1 /* collecting    */, 1 /* distributing */);

   ticks(100 * Lac_ticks);
}
```

Test Results : Scenario 2

```
********* TEST 2 *************
   0.0:RX at 707.1 Actor   1.5:1.808.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing
                   Partner 1.6:1.707.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing

   0.1:TX at 707.1 Actor   1.1:1.707.1 Active      .     Aggregate     .
                   Partner 1.5:1.808.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing

   0.1:TX at 707.2 Actor   1.2:1.707.1 Active   Nervous  Aggregate  In sync
                   Partner 1.2:1.  0.2 Passive  Nervous      .      In sync  Collecting  Distributing

   1.0:TX at 707.1 Actor   1.1:1.707.1 Active      .     Aggregate     .
                   Partner 1.5:1.808.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing

   1.0:TX at 707.2 Actor   1.2:1.707.1 Active   Nervous  Aggregate  In sync
                   Partner 1.2:1.  0.2 Passive  Nervous      .      In sync  Collecting  Distributing

   2.0:TX at 707.1 Actor   1.1:1.707.1 Active      .     Aggregate  In sync
                   Partner 1.5:1.808.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing

   2.0:TX at 707.2 Actor   1.2:1.707.1 Active   Nervous  Aggregate  In sync
                   Partner 1.2:1.  0.2 Passive  Nervous      .      In sync  Collecting  Distributing

   3.0:TX at 707.1 Actor   1.1:1.707.1 Active      .     Aggregate  In sync
                   Partner 1.5:1.808.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing

   3.0:TX at 707.2 Actor   1.2:1.707.1 Active   Nervous  Aggregate  In sync  Collecting  Distributing
                   Partner 1.2:1.  0.2 Passive     .         .      In sync  Collecting  Distributing

   4.0:TX at 707.1 Actor   1.1:1.707.1 Active      .     Aggregate  In sync
                   Partner 1.5:1.808.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing

   5.0:TX at 707.1 Actor   1.1:1.707.1 Active      .     Aggregate  In sync
                   Partner 1.5:1.808.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing

   6.0:TX at 707.1 Actor   1.1:1.707.1 Active      .     Aggregate  In sync
                   Partner 1.5:1.808.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing

   7.0:TX at 707.1 Actor   1.1:1.707.1 Active      .     Aggregate  In sync
                   Partner 1.5:1.808.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing

   8.0:TX at 707.1 Actor   1.1:1.707.1 Active      .     Aggregate  In sync
                   Partner 1.5:1.808.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing
   9.0:TX at 707.1 Actor   1.1:1.707.1 Active      .     Aggregate  In sync
                   Partner 1.5:1.808.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing
```

and so on until:

```
  90.0:TX at 707.1 Actor   1.1:1.707.1 Active   Nervous  Aggregate  In sync
                   Partner 1.5:1.808.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing


  91.0:TX at 707.1 Actor   1.1:1.707.1 Active   Nervous  Aggregate  In sync
                   Partner 1.5:1.808.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing


  92.0:TX at 707.1 Actor   1.1:1.707.1 Active   Nervous  Aggregate  In sync
                   Partner 1.5:1.808.5 Active   Nervous  Aggregate  In sync  Collecting  Distributing


  93.0:TX at 707.1 Actor   1.1:1.707.1 Active   Nervous  Aggregate  In sync  Collecting  Distributing
                   Partner 1.1:1.  0.1 Passive     .        .       In sync  Collecting  Distributing


  93.0:TX at 707.2 Actor   1.2:1.707.1 Active   Nervous  Aggregate  In sync  Collecting  Distributing
                   Partner 1.2:1.  0.2 Passive     .        .       In sync  Collecting  Distributing

test over
```

Test Scenario 3

```
static void test3()
{
   Lac_system *s1;
   Lac_port   *s1p1, *s1p2, *s1p3, *s1p4;
   Lac_system *s2;
   Lac_port   *s2p1, *s2p2, *s2p3, *s2p4;
   Lan        *lan1, *lan2, *lan3, *lan4, *lan5;

   printf("********* TEST 3 *************\n");

   (void)sys_create_lan(&lan3);
   (void)sys_create_lan(&lan4);
   (void)sys_create_lan(&lan5);

   lac_create_system(101, &s1);

   lac_create_port(  s1,1); s1p1 = s1->ports.next;
   lact_user_connect(s1p1);
   lact_lan_attach(  lan3, s1p1);

   lac_create_port(  s1,2); s1p2 = s1p1->next;
   lact_user_connect(s1p2);
   lact_lan_attach(  lan4, s1p2);

   lac_create_system(102, &s2);

   lac_create_port(  s2,3); s2p3 = s2->ports.next;
   lact_user_connect(s2p3);
   lact_lan_attach(  lan3, s2p3);

   lac_create_port(  s2,2); s2p2 = s2p3->next;
   lact_user_connect(s2p2);
   lact_lan_attach(  lan4, s2p2);

   ticks(100 * Lac_ticks);

   lac_create_port(  s1,3); s1p3 = s1p2->next;
   lact_user_connect(s1p3);
   lact_lan_attach(  lan5, s1p3);

   lac_create_port(  s2,1); s2p1 = s2p2->next;
   lact_user_connect(s2p1);
   lact_lan_attach(  lan5, s2p1);

   ticks(100 * Lac_ticks);
}
```

Test Results : Scenario 3

```
********* TEST 3 *************
   0.1:TX at 101.1 Actor   1.1:1.101.1 Active   Nervous  Aggregate  In sync
                   Partner 1.1:1.  0.1 Passive  Nervous     .       In sync  Collecting  Distributing

   0.1:RX at 102.3 Actor   1.1:1.101.1 Active   Nervous  Aggregate  In sync
                   Partner 1.1:1.  0.1 Passive  Nervous     .       In sync  Collecting  Distributing

   0.1:TX at 101.2 Actor   1.2:1.101.1 Active   Nervous  Aggregate  In sync
                   Partner 1.2:1.  0.2 Passive  Nervous     .       In sync  Collecting  Distributing

   0.1:RX at 102.2 Actor   1.2:1.101.1 Active   Nervous  Aggregate  In sync
                   Partner 1.2:1.  0.2 Passive  Nervous     .       In sync  Collecting  Distributing

   0.1:TX at 102.3 Actor   1.3:1.102.1 Active      .     Aggregate     .
                   Partner 1.1:1.101.1 Active   Nervous  Aggregate  In sync

   0.1:RX at 101.1 Actor   1.3:1.102.1 Active      .     Aggregate     .
                   Partner 1.1:1.101.1 Active   Nervous  Aggregate  In sync

   0.1:TX at 102.2 Actor   1.2:1.102.1 Active      .     Aggregate     .
                   Partner 1.2:1.101.1 Active   Nervous  Aggregate  In sync

   0.1:RX at 101.2 Actor   1.2:1.102.1 Active      .     Aggregate     .
                   Partner 1.2:1.101.1 Active   Nervous  Aggregate  In sync

   0.2:TX at 101.1 Actor   1.1:1.101.1 Active      .     Aggregate     .
                   Partner 1.3:1.102.1 Active      .     Aggregate     .

   0.2:RX at 102.3 Actor   1.1:1.101.1 Active      .     Aggregate     .
                   Partner 1.3:1.102.1 Active      .     Aggregate     .

   0.2:TX at 101.2 Actor   1.2:1.101.1 Active      .     Aggregate     .
                   Partner 1.2:1.102.1 Active      .     Aggregate     .

   0.2:RX at 102.2 Actor   1.2:1.101.1 Active      .     Aggregate     .
                   Partner 1.2:1.102.1 Active      .     Aggregate     .

   1.0:TX at 101.1 Actor   1.1:1.101.1 Active      .     Aggregate     .
                   Partner 1.3:1.102.1 Active      .     Aggregate     .

   1.0:RX at 102.3 Actor   1.1:1.101.1 Active      .     Aggregate     .
                   Partner 1.3:1.102.1 Active      .     Aggregate     .
```

```
1.0:TX at 101.2 Actor   1.2:1.101.1 Active      .     Aggregate     .
                Partner 1.2:1.102.1 Active      .     Aggregate     .

1.0:RX at 102.2 Actor   1.2:1.101.1 Active      .     Aggregate     .
                Partner 1.2:1.102.1 Active      .     Aggregate     .

1.0:TX at 102.3 Actor   1.3:1.102.1 Active      .     Aggregate     .
                Partner 1.1:1.101.1 Active      .     Aggregate     .

1.0:RX at 101.1 Actor   1.3:1.102.1 Active      .     Aggregate     .
                Partner 1.1:1.101.1 Active      .     Aggregate     .

1.0:TX at 102.2 Actor   1.2:1.102.1 Active      .     Aggregate     .
                Partner 1.2:1.101.1 Active      .     Aggregate     .

1.0:RX at 101.2 Actor   1.2:1.102.1 Active      .     Aggregate     .
                Partner 1.2:1.101.1 Active      .     Aggregate     .

2.0:TX at 101.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync
                Partner 1.2:1.102.1 Active      .     Aggregate     .

2.0:RX at 102.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync
                Partner 1.2:1.102.1 Active      .     Aggregate     .

2.0:TX at 102.3 Actor   1.3:1.102.1 Active      .     Aggregate  In sync
                Partner 1.1:1.101.1 Active      .     Aggregate     .

2.0:RX at 101.1 Actor   1.3:1.102.1 Active      .     Aggregate  In sync
                Partner 1.1:1.101.1 Active      .     Aggregate     .

2.0:TX at 102.2 Actor   1.2:1.102.1 Active      .     Aggregate  In sync  Collecting
                Partner 1.2:1.101.1 Active      .     Aggregate  In sync

2.0:RX at 101.2 Actor   1.2:1.102.1 Active      .     Aggregate  In sync  Collecting
                Partner 1.2:1.101.1 Active      .     Aggregate  In sync

2.1:TX at 101.1 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting
                Partner 1.3:1.102.1 Active      .     Aggregate  In sync

2.1:RX at 102.3 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting
                Partner 1.3:1.102.1 Active      .     Aggregate  In sync
2.1:TX at 101.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                Partner 1.2:1.102.1 Active      .     Aggregate  In sync  Collecting

2.1:RX at 102.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                Partner 1.2:1.102.1 Active      .     Aggregate  In sync  Collecting
```

```
  2.1:TX at 102.3 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting

  2.1:RX at 101.1 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting

 31.0:TX at 101.1 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

 31.0:RX at 102.3 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

 31.0:TX at 101.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.102.1 Active      .     Aggregate  In sync  Collecting

 31.0:RX at 102.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.102.1 Active      .     Aggregate  In sync  Collecting

 31.0:TX at 102.3 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

 31.0:RX at 101.1 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

 31.0:TX at 102.2 Actor   1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

 31.0:RX at 101.2 Actor   1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

 61.0:TX at 101.1 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

 61.0:RX at 102.3 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

 61.0:TX at 101.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
 61.0:RX at 102.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

 61.0:TX at 102.3 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

 61.0:RX at 101.1 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
 61.0:TX at 102.2 Actor   1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

 61.0:RX at 101.2 Actor   1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

 91.0:TX at 101.1 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

 91.0:RX at 102.3 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

 91.0:TX at 101.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

 91.0:RX at 102.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

 91.0:TX at 102.3 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

 91.0:RX at 101.1 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

 91.0:TX at 102.2 Actor   1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

 91.0:RX at 101.2 Actor   1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

100.1:TX at 102.3 Actor   1.3:1.102.1 Active      .     Aggregate     .
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

100.1:RX at 101.1 Actor   1.3:1.102.1 Active      .     Aggregate     .
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
100.1:TX at 101.3 Actor   1.3:1.101.1 Active   Nervous  Aggregate  In sync
                  Partner 1.3:1.  0.3 Passive  Nervous      .      In sync  Collecting  Distributing

100.1:RX at 102.1 Actor   1.3:1.101.1 Active   Nervous  Aggregate  In sync
                  Partner 1.3:1.  0.3 Passive  Nervous      .      In sync  Collecting  Distributing

100.1:TX at 102.1 Actor   1.1:1.102.1 Active      .     Aggregate     .
                  Partner 1.3:1.101.1 Active   Nervous  Aggregate  In sync

100.1:RX at 101.3 Actor   1.1:1.102.1 Active      .     Aggregate     .
                  Partner 1.3:1.101.1 Active   Nervous  Aggregate  In sync
```

```
100.2:TX at 101.1 Actor   1.1:1.101.1 Active      .     Aggregate  In sync
                  Partner 1.3:1.102.1 Active      .     Aggregate     .


100.2:RX at 102.3 Actor   1.1:1.101.1 Active      .     Aggregate  In sync
                  Partner 1.3:1.102.1 Active      .     Aggregate     .

100.2:TX at 101.3 Actor   1.3:1.101.1 Active      .     Aggregate     .
                  Partner 1.1:1.102.1 Active      .     Aggregate     .

100.2:RX at 102.1 Actor   1.3:1.101.1 Active      .     Aggregate     .
                  Partner 1.1:1.102.1 Active      .     Aggregate     .

101.0:TX at 101.3 Actor   1.3:1.101.1 Active      .     Aggregate     .
                  Partner 1.1:1.102.1 Active      .     Aggregate     .

101.0:RX at 102.1 Actor   1.3:1.101.1 Active      .     Aggregate     .
                  Partner 1.1:1.102.1 Active      .     Aggregate     .

101.0:TX at 102.1 Actor   1.1:1.102.1 Active      .     Aggregate     .
                  Partner 1.3:1.101.1 Active      .     Aggregate     .

101.0:RX at 101.3 Actor   1.1:1.102.1 Active      .     Aggregate     .
                  Partner 1.3:1.101.1 Active      .     Aggregate     .

102.0:TX at 102.3 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync

102.0:RX at 101.1 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync

102.0:TX at 102.1 Actor   1.1:1.102.1 Active      .     Aggregate  In sync
                  Partner 1.3:1.101.1 Active      .     Aggregate     .
102.0:RX at 101.3 Actor   1.1:1.102.1 Active      .     Aggregate  In sync
                  Partner 1.3:1.101.1 Active      .     Aggregate     .

102.1:TX at 101.1 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.3:1.102.1 Active      .     Aggregate  In sync  Collecting

102.1:RX at 102.3 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.3:1.102.1 Active      .     Aggregate  In sync  Collecting

121.0:TX at 101.1 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.3:1.102.1 Active      .     Aggregate  In sync  Collecting

121.0:RX at 102.3 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.3:1.102.1 Active      .     Aggregate  In sync  Collecting
```

```
121.0:TX at 101.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing


121.0:RX at 102.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

121.0:TX at 102.3 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

121.0:RX at 101.1 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

121.0:TX at 102.2 Actor   1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

121.0:RX at 101.2 Actor   1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

131.0:TX at 101.3 Actor   1.3:1.101.1 Active      .     Aggregate     .
                  Partner 1.1:1.102.1 Active      .     Aggregate  In sync

131.0:RX at 102.1 Actor   1.3:1.101.1 Active      .     Aggregate     .
                  Partner 1.1:1.102.1 Active      .     Aggregate  In sync

131.0:TX at 102.1 Actor   1.1:1.102.1 Active      .     Aggregate  In sync
                  Partner 1.3:1.101.1 Active      .     Aggregate     .

131.0:RX at 101.3 Actor   1.1:1.102.1 Active      .     Aggregate  In sync
                  Partner 1.3:1.101.1 Active      .     Aggregate     .
151.0:TX at 101.1 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

151.0:RX at 102.3 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

151.0:TX at 101.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

151.0:RX at 102.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

151.0:TX at 102.3 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

151.0:RX at 101.1 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
```

```
151.0:TX at 102.2 Actor   1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

151.0:RX at 101.2 Actor   1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

161.0:TX at 101.3 Actor   1.3:1.101.1 Active      .     Aggregate     .
                  Partner 1.1:1.102.1 Active      .     Aggregate  In sync

161.0:RX at 102.1 Actor   1.3:1.101.1 Active      .     Aggregate     .
                  Partner 1.1:1.102.1 Active      .     Aggregate  In sync

161.0:TX at 102.1 Actor   1.1:1.102.1 Active      .     Aggregate  In sync
                  Partner 1.3:1.101.1 Active      .     Aggregate     .

161.0:RX at 101.3 Actor   1.1:1.102.1 Active      .     Aggregate  In sync
                  Partner 1.3:1.101.1 Active      .     Aggregate     .

181.0:TX at 101.1 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

181.0:RX at 102.3 Actor   1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

181.0:TX at 101.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
181.0:RX at 102.2 Actor   1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing

181.0:TX at 102.3 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

181.0:RX at 101.1 Actor   1.3:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.1:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

181.0:TX at 102.2 Actor   1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

181.0:RX at 101.2 Actor   1.2:1.102.1 Active      .     Aggregate  In sync  Collecting  Distributing
                  Partner 1.2:1.101.1 Active      .     Aggregate  In sync  Collecting  Distributing

191.0:TX at 101.3 Actor   1.3:1.101.1 Active      .     Aggregate     .
                  Partner 1.1:1.102.1 Active      .     Aggregate  In sync

191.0:RX at 102.1 Actor   1.3:1.101.1 Active      .     Aggregate     .
                  Partner 1.1:1.102.1 Active      .     Aggregate  In sync
```

```
 191.0:TX at 102.1 Actor   1.1:1.102.1 Active      .     Aggregate  In sync
                   Partner 1.3:1.101.1 Active      .     Aggregate     .

 191.0:RX at 101.3 Actor   1.1:1.102.1 Active      .     Aggregate  In sync
                   Partner 1.3:1.101.1 Active      .     Aggregate     .

test over
```