LACP Wait To Restore specification

Mick Seaman

The 'Wait To Restore' (WTR) specification in 802.1AX-2014 could lead to excessive frame loss[1] and was removed in the recent corrigendum. A change to the MUX machine is sufficient to add the desired functionality, once some other existing specification issues have been corrected. The usual guarantees of protocol version compatibility are provided[2].

---

1. Summary

Figure 1 is the current, coupled control variant of the MUX machine, once corrected (see 2 below). Figure 2 adds WTR. The layout of these figures follows that of 802.1AX-2014 Figure 6-22 to facilitate comparison. Figure 3 is a further improvement, with Figure 2's functionality and a better division of responsibility between the Mux machine and the Selection Logic (see 2.4). Figure 4 is my final suggestion. It replaces most of the state to state transitions with global transitions, further reducing clutter and avoiding transmission of back to back LACPDUs where only one is wanted. It covers both coupled control and independent control Mux Machines[3].

WTR, as proposed here, works as follows (the reader may find Figure 3 the easiest reference)[4]:

1) The design is predicated on the assumption that attaching/detaching an Aggregation Port to/from an Aggregator involves significantly more system effort (at least in some implementations) than enabling/disabling collection and distribution. The current 802.1AX specification makes a particular point of avoiding detaching/re-attaching [6.4.12 pg 56, PORT_DISABLED][5]. If that were not the case the Selection Logic could provide WTR functionality by using STANDBY and only reattaching the Aggregation Port when it is to be brought into service again.

2) In the current specification the Mux machine sets Actor.Sync TRUE when the Aggregation Port has been attached to the correct Aggregator (in MUX:ATTACHED). Receipt of that Sync by the Actor's Partner allows that Partner (provided it is also correctly attached to an Aggregator) to start collecting received frames and (if its collection and distribution are coupled) to start transmitting distributed frames.

   An Actor transmitting Sync (TRUE) to a Partner that couples collection and distribution and has itself not yet sent Sync (someone has to get the ball rolling) is therefore indicating its willingness to do its best to receive frames as soon as it receives Sync from its Partner (who might start sending data as soon as it receives the Actor's Sync). The Actor will lose data the Partner sends before the Partner's next LACPDU (conveying Partner.Sync) is received and used by the Actor to enable collection[6].

   Equally an Actor transmitting !Sync (i.e. Sync == FALSE) indicates that it is not willing to receive frames in the immediate future. In the current specification this can be because the Aggregation Port has not yet selected an Aggregator, has been made STANDBY, or is not yet attached to the SELECTED Aggregator.

   The proposed WTR specification adds 'waiting to restore' to the above list of reasons for sending !Sync. The Partner's required behavior is the same in all cases, so a Version 1 conformant implementation without WTR will interoperate. If both Actor and Partner implement WTR (possibly with different WTR Timeout values) the link will be used once both have decided to restore connectivity.

3) When port_enabled becomes FALSE (indicating link down) the Receive state machine transitions to the PORT_DISABLED state, setting Partner.Sync FALSE, but holding on to the Partner's ID and Key (unless that ID is received on another port) until the port_enabled become TRUE once more and then for anything up to a further LACP 'Short Timeout'

[1] Diagnosed by Steve Haddock and discussed in the September 2016 interim.

[2] Without error prone procedures of the form 'if he is running a prior version then I'm going to send a variant of this message'.

[3] These could be trivially separated if maintaining the current Figure 6-21, 6-22 separation is thought important for continuity in the standard.

[4] Figure 3 tells the clearer story, but Figure 4 is what you want to implement.

[5] All references in square brackets are to 802.1AX-2014.

[6] When an aggregated link is first being brought into service this is less of an issue than might appear at first. One (at least) of the communicating participants is probably implementing protocol retries at intervals greater than the loss period described, and if both transmit when they (individually) see the link enabled (Collecting and Distributing both TRUE) then the last of the two to begin transmission should see its initial messages get through (providing that its peer has actually enabled Collecting and Distributing by the time its LACPDU is transmitted). The details vary by higher-layer protocol. When a link is being added to an existing aggregate, loss will occur unless some additional conversation distribution protocol makes sure the link is fully useable before distributing data.

period. This allows the Selection Logic to maintain its current Aggregator selection until communication with the Parter is restored. In the current specification, the Mux machine does not distinguish between a change to !Partner.Sync as result of (a) the Partner asserting !Sync and (b) the link going down (!port_enabled). These have to be distinguished to implement WTR, otherwise an Actor might assert !Sync because the Partner has asserted !Sync, and vice versa, resulting a feedback loop that can prevent the link from being used again[7]. The proposed Mux state machine asserts !Actor.Sync (in MUX:ATTACHED_WTR) after !port_enabled until it wishes to attempt to restore connectivity, but does not change Actor.Sync in response to !Partner.Sync if the link remains up (port_enabled).

4) The proposed Mux state machine restarts the WTR time (wtr_while) while !port_enabled, thus requiring port_enabled to be continuously true for the wtr_timeout period before connectivity is restored (by transitioning to MUX:ATTACHED and setting Actor.Sync once more). The point here is to avoid use of an unreliable link, not just to try using the link at wtr_timeout intervals.

   To be more exact, the proposed machine uses a wtr_while timer variable that is decremented by a once per second timer tick (until it reaches zero), so all the conditions and actions are properly expressed in 'C' (and not, e.g., as 'Start wait_while_timer')[8]. This once per second decrement means that wtr_while can be restarted whenever it is not at its initial timeout value and port_enabled becomes false, without specifying a potential spin-lock in the state machine[9]. The state then actually implements a delay of between (wtr_timeout) and (wtr_timeout -1) after port_enabled last becomes true.

2. Current specification issues[10]

The casual reader is warned that this section is a historical record of an analysis of the 802.1AX-2014 text, and might want to skip directly to the proposed Improved Interfaces (see 3 below) assumed by Figure 2 and subsequent figures after reading the first few paragraphs below.

2.1 LACP Selection Logic

The LACP Selection Logic is responsible for assigning Aggregation Ports (each using an individual MAC providing service over a single link) to Aggregator Ports (generally abbreviated to Aggregators). When Link Aggregation was originally developed, care was taken to not overconstrain the selection algorithms[11]. Unfortunately the boundary between Selection Logic and the state machines was informally specified, without all the variables required by state machine logic. Since the existing specification is strictly incorrect[12] it is hard to extend the Mux machine until the deficiencies described below are corrected.

2.2 Selection Logic—Operation

"The Selection Logic is responsible for selecting the Aggregator to be associated with this Aggregation Port." [6.4.3 c)]

The Selection Logic is said to be a state machine [6.4.3 "The state machines are as follows: … c) *Selection Logic* …"

"The Selection Logic is invoked whenever an Aggregation Port is not attached to and has not selected an Aggregator, and executes continuously until it has determined the correct Aggregator for the Aggregation Port." [6.4.14.1 m) on pg 51]. This statement is problematic, the two conditions 'not attached to' and 'not selected' appear to be both required for Selection Logic operation, but this is

[7]This behaviour is the root cause of the poor connectivity diagnosed by Steve. The prior WTR specification implemented hysteresis rather than a true wait after last !port_enabled to port_enabled transition, so connectivity was possible with luck, but the link could oscillate in and out of connectivity if the Actor and Partner did not act exactly in synchronization with the same WTR Timeout values and negligible communication delay.

[8]This is an important general principle since it allows the machines to be coded and run (under the specified state machine conventions) exactly 'as is' without additional interpretation. The latter is a potential source of errors and dispute as to their validity.

[9]Where the conditions for execution of a state remain true after execution an indefinite number of times. The state machine conventions don't say anything about fairness of execution between multiple machines, so under those conditions execution to the exclusion of other machines with executable conditions is possible.

[10]All references in square brackets are to 802.1AX-2014.

[11]The links in any given aggregate can change over time, and uninterrupted service between two LACP capable systems could be provided by a set of links that has entirely changed over time. If a system identified each Aggregator solely by LAG ID (a tuple including both connected System Identifiers and each of their LACP Keys that identify their aggregatable links on each) it is easy to see how such Selection Logic would work. However if a system's ports are often not aggregated, and are managed by port number, it is convenient to use the same port numbers to identify the Aggregators (one for each Aggregation Port, to accommodate the scenario where none are aggregated). In this port numbered case, the successive rerouting of links could lead to a scenario where the connectivity between the Aggregation Ports and Aggregator Ports cannot be explain by the current network configuration. Since the successive rerouting scenario is probably rare, the recommended selection algorithm is history independent (almost).

[12]Of course every one can see 'what the specification was intended to do' and fix their own implementations accordingly, but such fixes are inclined to differ and work for some implementations and not others, so it is even hard to come up with an implementation independent statement of what needs to be fixed.

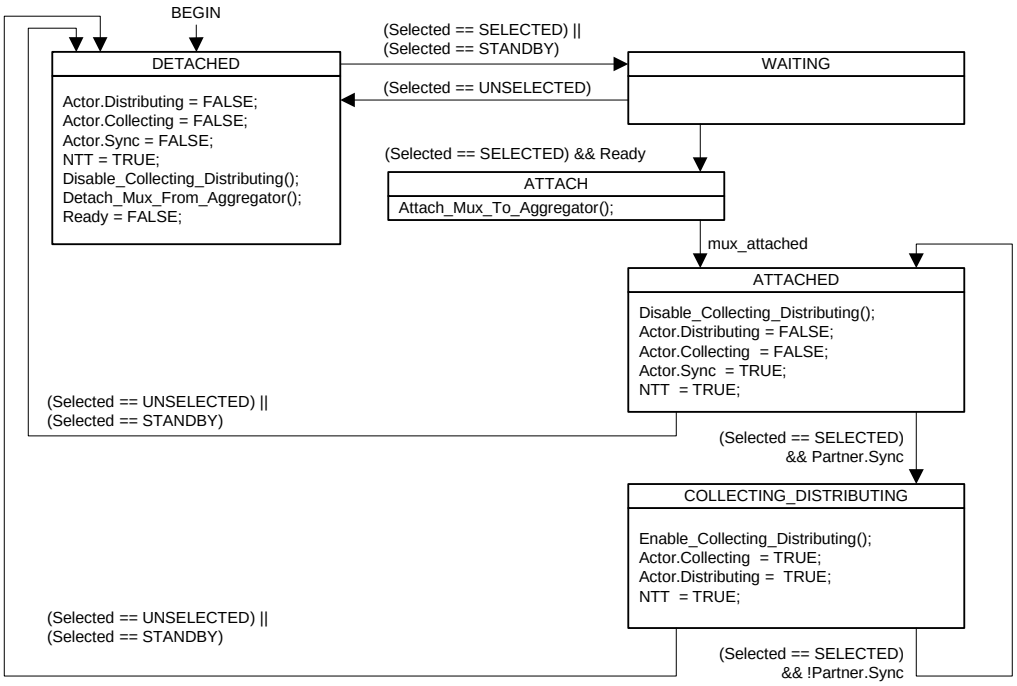

Figure 1—MUX machine state diagram (coupled control, without WTR)

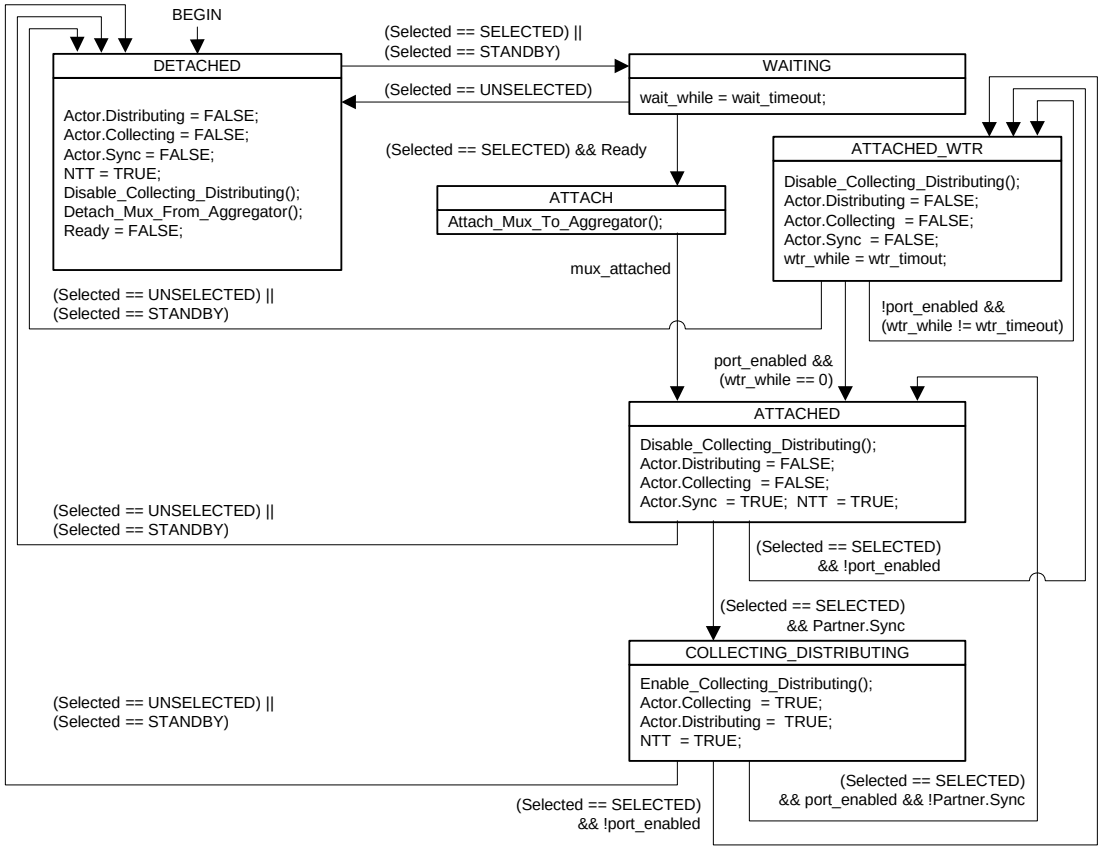

Figure 2—MUX machine state diagram (coupled control, with WTR)

clearly not the case (otherwise the Selection Logic would not be able to make a STANDBY link SELECTED, see 2.6 below, or a failed SELECTED link UNSELECTED in order to make way for a STANDBY, see 1.4). Neither of these cases is addressed by invoking the fact that the operation of the Selection Logic for one Aggregation Port can cause changes to the selection for another.

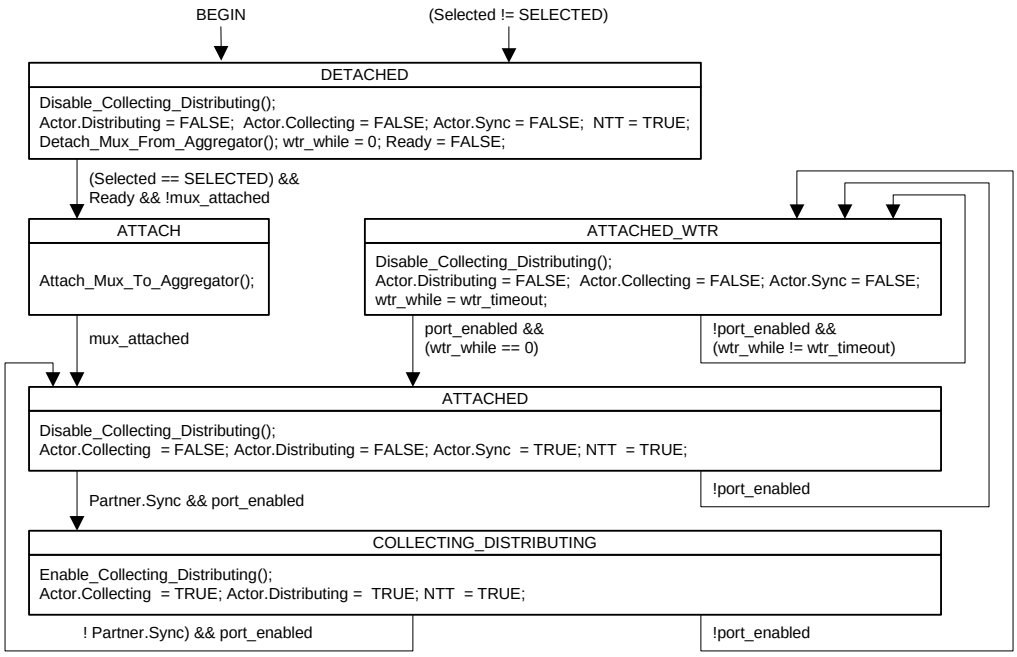

Figure 3—MUX machine state diagram (coupled control, with WTR, redrawn)

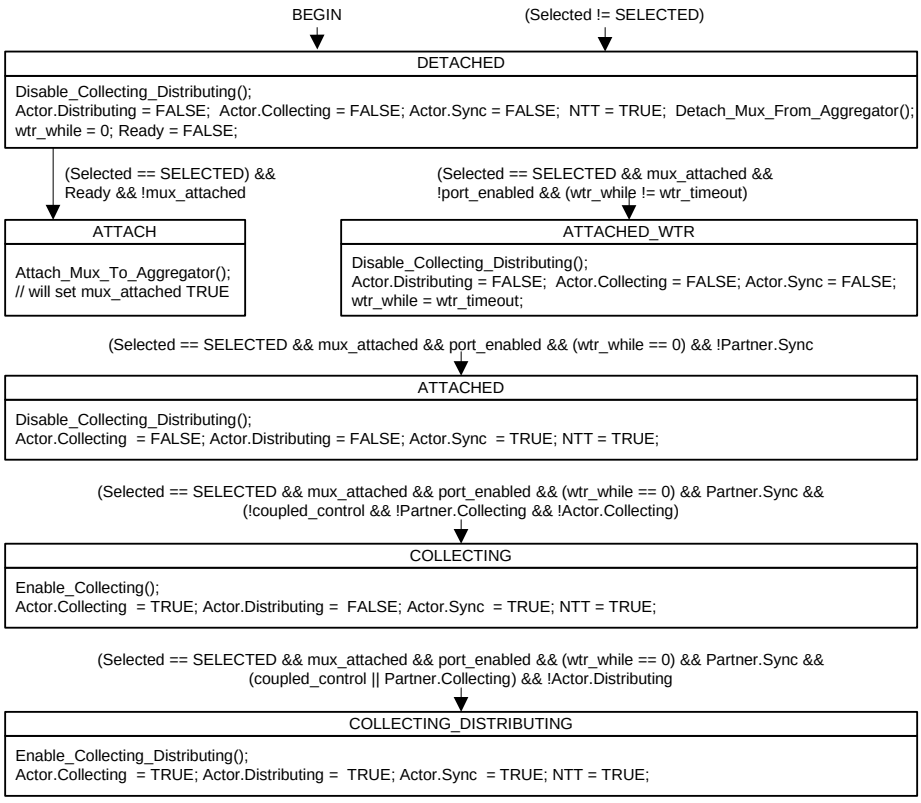

Figure 4—MUX machine state diagram (independent and coupled control, with WTR)

It also begs the question as to what is meant by ‘continuously’, other than advertising that the specification has not identified all conditions under which it should run.

“The value of the Selected variable may be changed by the following:

a) The Receive machine. The Receive machine can set Selected to UNSELECTED at any time if any of the following change: The Partner System ID, the Partner

Key, the Partner_State.Aggregation, the Actor System ID, the Actor Key, or the Actor_State.Aggregation.

b) The Selection Logic, in the process of selecting an Aggregator. The Selection Logic will select an Aggregator when the Mux machine is in the DETACHED state and the value of the Selected variable is UNSELECTED.

c) The Selection Logic, in the process of selecting or deselecting standby links. If the value of the Selected variable is SELECTED or STANDBY, the Selection Logic can change the value to STANDBY or SELECTED." [6.14.5 pg 61]. This text is a more comprehensive description of the possible changes to Selected, though not a complete description of Selection Logic operation[13].

"Where the selection of a new Aggregator by an Aggregation Port, as a result of changes to the selection parameters, results in other Aggregation Ports in the System being required to reselect their Aggregators in turn, this is achieved by setting Selected to UNSELECTED for those other Aggregation Ports that are required to reselect their Aggregators."[6.4.14.1 p)]. The NOTE following explains that the Receive machine does this in certain cases, but does not cover them all.

6.4.14 p), r), and s) discuss Selection Logic functionality without mentioning 'Selection Logic', for example: "An Aggregation Port shall not select an Aggregator …'. This is a mistake, and not just because an individual Aggregation Port can't perform the necessary selection on its own. Leaving out 'Selection Logic' runs the risk of future maintenance missing these bullets.

2.3 Selected signals to multiple machines

The state machine variable Selected is an output of the receive state machine (RX), both an input to and an output from the Selection Logic, and an input to the MUX state machine [Figure 6-17]—a major design error (see below).

Selected can be set to UNSELECTED by the Receive state machine and the Selection Logic can set Selected to UNSELECTED, but only the latter can set it to STANDBY or SELECTED [6.4.8 pg 51, defn. of Selected; 6.4.12 NOTE 1].

"An Aggregation Port is always detached from its prior Aggregator when the LAG ID changes, even if the same Aggregator is selected later; to do otherwise would be to risk misdelivery of frames. Selection of a new Aggregator cannot take place until the Aggregation Port is detached from any prior Aggregator; …" [6.4.14.1 pg 59, NOTE 3]. This observation is crucial, but NOTEs are by definition non-normative and there is nothing normative to back up this statement. The Selection Logic does not, itself, change the value of Partner.Sync. If it were to run in the COLLECTING or DISTRIBUTING MUX machine states and select a different Aggregator it might remain in that state. If the Actor then sends a LACPDU (prompted by the Periodic Transmission machine) it is possible for the Partner to sync up with the new information before sending its next LACPDU with information that allows the first Actor to maintain Partner.Sync while connected to the wrong Aggregator. Note that the specification does not require management changes to the LACP Keys to set UNSELECTED (which it should), but even if it did the Selection Logic could select a new Aggregator and set Selected back to SELECTED before the MUX machine reacts (the state machine conventions require atomic/indivisible execution of the actions in any state execute with respect to any other state in any machine, so there is no weaseling out of this by claiming that Selected is processed by the Selection Logic and the MUX machine simultaneously).

The use of Selected to signal to two machines, the MUX machine and the Selection Logic, one of which can change Selected[14] is the design error referred to above. A cleaner design might have used separate variables, but making such a change to the specification is hardly fair to those who have already designed implementation workarounds based on the current spec. A practical solution is to specify constraints on the operation of the Selection Logic using the Ready variable (see 2.4 below). If the Selection Logic sets SELECTED and Ready (TRUE) it cannot select a different Aggregator to any already chosen (whether it changes Selected or not), until the MUX machine has cleared Ready (to FALSE) again.

[13] The various text excerpts in this section (1.1) illustrate the editor's bane—many commenters would like to include text that addresses their concerns in the document, without regard to the fact that relevant facts are already stated elsewhere in the document and that both their contribution and the existing facts are an incomplete statement of what needs to be said, and might actually conflict. As the document gets larger further commenters are even more disinclined to read it all before commenting. Much of this note is concerned with finding out what the standard actually says.

[14] Or, equivalently, leaving the Selection Logic's updating of Selected to chance when it needs to be seen by the MUX machine)

2.4 Ready specification

The signal Ready, part of the condition for the transition between MUX:WAITING and MUX:ATTACHED, is not initialized in the current machines. It should be cleared in the MUX:DETACHED state. When this is done Ready nicely separates those times when the MUX machine can act on the Selection Logic's choice of Aggregator (when Ready is TRUE) and those times when the Selection Logic is free to choose a new Aggregator (when Ready is FALSE).

"The MUX machine asserts Ready_N TRUE to indicate to the Selection Logic that the wait_while timer has expired …" [6.4.8, pg 51 defn. of Ready_N]—it doesn't.

"The Selection Logic asserts Ready TRUE when the values of Ready_N for all Aggregation Ports that are waiting to attach to a given Aggregator are TRUE." [6.4.8, pg 51 defn. of Ready]. This statement is equivalent to stating that the wait_while timer has run and expired for all such Aggregation Ports. It mandates a delay that is often annoying and quite unnecessary in some circumstances (e.g. if all Aggregation Ports with a given LACP Key are in the MUX:WAITING, or the full capacity of the Aggregator satisfied) there is no point in waiting further. The first of these points is the subject of the NOTE to 6.4.15 d), but NOTEs cannot be normative so it has no effect. A better, comprehensive, solution would have been to put the wait_while timer within the Selection Logic, have the DETACHED state subsume WAITING, and allow the Selection Logic to run the wait_while timer (if it wishes). There seems little point to the present combination of allowing implementation variation in the Selection Logic while at the same time overconstraining it. A smart implementation might record the selection when it has been running for a while, and when powering up next time use that as a prompt to set Ready if all those links have been selected, rather than wait on the off chance that links have been redeployed. 'Waiting' can still be reported, for management purposes

It is probably obvious that "Aggregation Ports that are waiting to attach" don't include those that are STANDBY, but that would be better made explicit if the Selection Logic is to be constrained in this way.

2.5 RX:PORT_DISABLED

"If the Aggregation Port becomes inoperable and the BEGIN variable is not asserted, the state machine enters the PORT_DISABLED state. Partner_Oper_Port_State.Synchronization is set to FALSE. This state allows the current Selection state to remain undisturbed, so that, in the event that the Aggregation Port is still connected to the same Partner and Partner Aggregation Port when it becomes operable again, there will be no disturbance caused to higher layers by unnecessary re-configuration." [6.4.12 pg 12, 2nd para.].

It is unclear what 'allows' and 'no disturbance' mean. Partner_Oper_Port_State.Synchronization (Partner.Sync in the MUX machine [Figure 21]) is set FALSE, so the MUX machine will disable Collecting and Distributing. If this is the only link in the aggregate then we should not be telling the higher layers that it is still up. If it is not and conversations are not redistributed, those previously carried by this port will be lost until LACP times out (thus defeating the recommendation in 6.1.1 NOTE 1 to use LACP timeouts only as a method of last resort for link failure detection, preferring CFM when direct hardware support is not applicable) unless the Selection Logic intervenes. And yet the stated purpose of the RX:PORT_DISABLED state could be read as preventing the Selection Logic from intervening.

The most generous reading of the quoted text is that the state, by not setting Selected UNSELECTED, avoids forcing the MUX machine through its DETACHED and WAITING states (with a mandated wait_while delay in the latter) before the port can become operational again, and that the Selection Logic will find out what has happened in some unspecified way (without the benefit of seeing Selected UNSELECTED) if it wishes to substitute a STANDBY link in the aggregate. The variables currently in the MUX machine don't help, since they don't distinguish a link that has entered ATTACHED from COLLECTING from one that has entered from WAITING.

2.6 Selection Logic—STANDBY option

"The Selection Logic may determine that the link should be operated as a standby link if there are constraints on the simultaneous attachment of Aggregation Ports that have selected the same Aggregator." [6.4.14].

The 'may' indicates an option that is included in the PICS (SLM11). The practical effect of STANDBY (as opposed to UNSELECTED) is that a STANDBY link

can be in the MUX:WAITING state with the wait_while timer already expired, so there might be no delay in substituting the STANDBY link for a failed link. However to make this substitution under the stated constraints would mean that the Selection Logic would have to detect the failure (see 1.4 above) and make the failed link UNSELECTED, so its MUX machine can transition to DETACHED and detach it to make room for the STANDBY link.

It is not clear from the quoted 6.4.14 text whether the use of STANDBY is intended to be restricted to the case of constrained aggregations, or indeed whether any configuration that calls for the use of STANDBY is such a constraint. It is also unclear why this is an option. If there is a constraint (whether as a result of lack of hardware capability or of explicit configuration), the implementation difficulty associated with moving a link from MUX:DETACHED (where it would remain if UNSELECTED) to MUX:WAITING is trivial. In complex cases (two or more links in STANDBY, with the Selection Logic deciding which to make SELECTED in the event of a failure of a currently SELECTED link) ticking the box for the option provides the user with very little information.

The use of ‘should’, a conformance term indicating a recommended option, in the quoted text is also problematic. The whole sentence would be better using ‘can’, with the equipment supplier providing supplementary information if the Selection Logic has been designed to be particularly clever or flexible.

If the failed link is to serve as the new STANDBY (after subsequent recovery) it might then transition to WAITING, starting its wait_while timer. There is nothing in the current specification that says that this timer should not be running if port_enabled is FALSE. Of course the Selection Logic has to avoid making the failed link SELECTED again (following the current specification, see 1.3 above) that would inhibit the transition of the previous STANDBY link to ATTACHED

The 6.4.14 text quoted above is repeated, with a little elegant variation in 6.4.14.1 k) on pg 59.

2.7 Attaching and Detaching

“On entry to the ATTACHED state, the Mux machine initiates the process of attaching the Aggregation Port to the selected Aggregator. Once the attachment process has completed, … A change in the Selected variable to UNSELECTED or to STANDBY causes the state machine to enter the DETACHED state. The process of detaching the Aggregation Port from the Aggregator is started. Once the detachment process is completed, …” [6.4.15 pg 62, describing the operation of the MUX machine].

This text acknowledges the fact that the attaching/detaching operations can take some time. However waiting for an operation to complete half way through state execution is runs counter to the spirit of, and quite possibly the letter of, the state machine conventions. Each state executes atomically/indivisibly with respect to any other state for any state machine in the system, so waiting part of the way through state execution for an operation to complete means that no other state in any machine can execute, including those of any machine that is responsible for executing the operation in progress.

The motivation for having the process of detaching the Aggregation Port from the Aggregator complete before execution of DETACHED completes might have been to ensure that detaching completes before a transition to WAITING [see Figure 6-21 or Figure 6-22], but that can be guaranteed by requiring that the Selection Logic not set Selected to SELECTED or STANDBY before the Mux is detached. Introducing an additional variable mux_attached to formalise that linkage between the MUX machine, the implementation dependent aggregator function, and the Selection Logic, would provide the necessary clarity.

I note in passing that the ordering of operations in the MUX:DETACHED state is odd. If the Partner system is using the coupled control MUX machine [Figure 6-22, pg 63] and is in the ATTACHED state, having Actor.Sync TRUE incents that Partner to move to COLLECTING_DISTRIBUTING. If detaching the Mux is going to take some time, Actor.Sync should be made FALSE first. I suspect that the current DETACHED state execution ordering was chosen so that Actor.Sync could serve as a proxy for the proposed mux_attached variable. It has the further oddity that Collecting and Distributing are disabled after the Mux has been detached, by which time there is nothing to or from which frames could be collected or distributed.

2.8 Frame loss

It is worth noting that even the independent Mux machine [Figure 6-21 pg 61] makes little effort to avoid frame loss when a link is being deliberately remove from an aggregator (as opposed to failing). It would not be difficult to arrange for a delay (terminated by a short timer, or a round trip of the marker protocol) between signalling Actor.Collecting

FALSE and Disable_Collecting(), both currently in MUX:ATTACHED. On the other hand, such a change would be entirely interoperable with the standard state machines and protocols, and the latter deal with the common case of link failure where it is impossible to avoid loss.

3. Improved Interfaces

3.1 Selection Logic interfaces

The following variables are maintained for each Aggregation Port (link), and are used to communicate between the Selection Logic and the other state machines and system resources (loosely called ‘hardware’ from this point on):

— Selected (UNSELECTED, STANDBY, SELECTED) as in the current specification.

— Selected_Aggregator, the Aggregator last chosen for this Aggregation Port by the Selection Logic if Selected is STANDBY or SELECTED, unspecified if Selected is UNSELECTED. Updated by the Selection Logic and not changed by any other machine. Read by the hardware when the MUX machine calls Attach_Mux_To_Aggregator().

— Ready. Set TRUE by Selection Logic to indicate to the MUX machine that the Aggregation Port is to be attached to the Selected_Aggregator (if Selected is SELECTED). When Ready is TRUE the Selection Logic can change Selected to STANDBY or UNSELECTED, but cannot change it to SELECTED again or change Selected_Aggregator. Ready is cleared by the Mux machine to indicate to the Selection Logic that it can change Selected to SELECTED and/or change Selected_Aggregator.

— port_enabled, defined as in the current specification.

3.2 Mux hardware interfaces

The following procedures and variable(s) are used/maintained by each Aggregation Port (link), and are used to communicate between the Mux state machines and the hardware:

— Detach_Mux_From_Aggregator() can be called at any time. If the Aggregation Port is not connected to an Aggregator, or the process of detaching it has already begun, it has no effect. Otherwise the process of detaching the Aggregation Port is started, mux_attached is cleared (becomes False) and the procedure returns.

— Attach_Mux_To_Aggregator() starts the process of attaching the Aggregation Port to the Selected_Aggregator. Subsequently (when that process is complete) mux_attached is set TRUE. This procedure cannot be repeated if there has been no intervening call to Detach_Mux_From_Aggregator() (or its effects are undefined if so repeated). Similarly the results of changing Selected_Aggregator after it has been called are undefined. If Attach_Mux_To_Aggregator() is called after Detach_Mux_From_Aggregator() but before the process of detaching is complete, the attachment request will be honoured (with the requested Selected_Aggregator) as soon as possible.

— mux_attached. See the definition of the procedures above. Note that a single variable cannot be used to describe both the definite states resulting after attaching or detaching are complete if there is an intervening hiatus while the process of attaching or detaching is in progress. mux_attached means that the Aggregation Port has been attached to the Selected_Aggregator and frames are distributed from and collected to that Aggregator (if enabled) and not to any other Aggregator. !mux_attached does not mean that the Aggregation Port is detached (or will not become attached shortly).

4. 802.1AX-2014 WTR issues

4.1 WTR feedback loop

As Steve Haddock observed in the interim, part of the loss problem was due to undesirable feedback. An LACP Actor (Alice, say) could be making decisions about the setting of the Sync variable she sends in the protocol (Actor.Sync or, to avoid taking one side’s point of view when each thinks of him or herself as ‘the actor’ and the other as ‘the partner’, Alice.Sync) that depend on the value of Pete.Sync. Then of course Pete.Sync depends on Alice.Sync, and so on.

Looking at this issue more broadly, 1AX-2014 describes the operation of the wtr_timer in terms of Actor_Oper_Port_State.Distributing, not only in terms of its purpose (2014-6.6.2.5 "This timer is used to prevent frequent distribution changes ..") but also in terms of how it is controlled (2014-6.6.2.2 "..sets the WTR_timer when .... Distributing changes..".

4.2 WTR goals

The purpose of the wtr timer isn’t stated with any great precision (2014-6.6.2.5 "prevent frequent distribution changes due to an intermittent defect". What sort of defect? Apparently any sort of defect that causes frequent changes. The circularity is clear: as soon as one participant tries to make distribution changes (particularly when both directions of a conversation

are to be mapped to the same link, with the possibility of Alice discarding traffic that Pete has put on the ‘wrong’ link) we are in trouble. Each participant uses its own, independent WTR Timer as part of the process of updating a conversation mask (2014-6.6.2.4 updateConversationMask on page 80).

There is a reason why each participant would like its own detection mechanisms for flaky links (assuming that those are what is meant by ‘intermittent defect’ in 2014-6.6.2.5). If the link is ‘composite’ or ‘multi-hop’, perhaps traversing one or more TPMRs or even a connection across a service provider network, one end might see link down (!portEnabled) events that are invisible to the other. Unfortunately, in this case, the link might be temporarily disrupted without either of the LACP participants being aware of that fact. Rather than add information to LACP which would allow one participant to signal to another “I think that this link is flaky even if you don’t, so I am going to do/doing the following.."[15] a better solution is to use CFM (or some alternative, already designed, link monitoring protocol) so that the state of the entire link can be represented by portEnabled.

If 2014-6.6.2.5’s ‘intermittent defect’ is intended to cover not just link unreliability, but extends to undesirable behavior exhibited by a LACP partner (or even some defect in the actor’s own protocol stack), we are headed down the old illusory path of trying to figure how out how to run a protocol when the partner doesn’t follow the protocol.

[15]Attempting to modify LACP to do this “almost for free” is very enticing. I made several bad attempts to do this, which must have been frustrating to Steve.