A UML model for link aggregation management

Mick Seaman

This note is an initial proposal for a UML model for managing link aggregation (LAG), developed as part of reviewing IEEE P802.1AX-Rev/D1.0.[1] While YANG is an obvious target, I hope changes suggested in this note can also be used to simplify the DRNI description and the revised MIB.

1. Summary

Section 2 considers the current organization of management specifications in 802.1 documents in general, and what we should do in the future. YANG augmentation will affect our structuring of management information. The UML(ish)[2] in Figure 2 uses augmentation to simplify LAG configuration, reducing the scope for errors, temporary inconsistencies, and duplication resulting from associating similar parameters with separate interfaces[3]. The mapping of the UML to 1AX-Rev/D1.0 management variables is discussed for basic LAG (Section 3), CSCD (Section 4), and DRNI (Section 5).

2. Management specifications

Specifying management objects[4] in a supposedly management protocol independent clause, separate from the text and state machine specification of what is managed, is cumbersome and error prone. While this seemed to be working when we were dealing with pre-standard management protocols, GDMO, and early MIBs, it has lead to the generation of translation tables that are hard to maintain and review.

A better approach would have been to include the use of all management controls, and updating of counters etc. within the basic operational specification, using the descriptive techniques natural to that specification and making sure each control and counter in that specification could be unambiguously referenced by a clause number and a name.

An SNMP MIB, for example, would then reference that specification directly instead of having an additional layer of translation and repetition[5] .

We may not want to change existing MIBs. Product MIBs bearing greater or lesser resemblance to the standard MIB and the standard's state machines and procedures have already been written and deployed. The aim should be to minimize the amount of work involved in revising standards. However we don't need to add yet another translation table for YANG.

A UML model is incredibly useful in describing the management attributes and operations, not least by making it possible to get the entire picture on one page and thus avoid overloading the natural limitations of human short-term memory. A simple numeric cross-reference direct to text that both defines and uses each management variable should be sufficient, even if naming conventions (prefixes, case stropping, and abbreviation) result in a detailed name change.

Management-protocol-independent-specification has proved to be an illusion[6]. Management frameworks (such as that underlying YANG) make sweeping assumptions about the organization of information[7]. At the same time, simple descriptions of what is to be managed are readily understandable without framework and protocol specific expertise.

We should develop a UML model that: (a) directly references .1AX Clauses 6 and 9, with additional or clarifying text in those clauses as necessary to allow

[1] In general (not particularly in P802.1AX-Rev) the description of management objects is so verbose as to discourage all but the most dedicated of reviewers. A typical managed object definition/declaration takes about 6 lines of text, only part of which is a more or less accurate repetition of what has to be (and is already) said elsewhere in a standard. In contrast C++ can fit the declaration of up to half-a-dozen or so such objects on a single line (many objects being of a common type and supporting the same set of operations, and the name being a sufficient reference to the existing description), and UML can provide an even greater information density - to the point where it is actually possible to see what is going on in a module of significant size.

[2] This has a strong resemblance to the YANG tree, a correspondence which I might usefully explore further.

[3] A previous version of this note analyzed the duplication of information between Aggregator and Aggregation Port interfaces and went through D0.4 clause 7 in detail, mapping it to potential YANG attributes. This analysis has been removed because I have not had time to update it and (for reasons described in this note) I don't think we should maintain a MIB/YANG mapping. It contained information on the scale of the duplicate information problem, but is otherwise superseded by this version.

[4] In most 802.1 standards, dating from 1987.

[5] As, for example, in 7.3 of .1AX/D0.4) where the repetition of "ATTRIBUTE", "APPROPRIATE SYNTAX", and "BEHAVIOR DEFINED AS" overlaps with what is in the MIB, does not have unambiguous references to behavioral specification, and in any case leads to a further 4 page mapping table (Table D.1) enroute to the MIB. In addition to 7.3 we have 5 pages of Table 7-1.

[6] Unique top-down decompositions of a difficult problems are fairy stories. A prominent example is the breakdown of compiler operation into logical and obvious successive stages as presented in introductory text books. The real history followed a more experimental and iterative path.

[7] The design of particular management protocols may make some structures impracticable, or only achievable through convoluted work-arounds.

an unambiguous reference to be made; and (b) is deliberately designed to make the final translation to YANG as mechanical as possible. This should facilitate review by those who can't or won't plow their way through the YANG code, and help ensure that what is managed is what is in Clauses 6 and 9[8] and not some alternative specification.

Each of the existing 802.1 management specifications reflects the logical details of a standardized operational model.[9] The organization of management variables corresponds directly to the structure of the model: its entities, sublayers, service access points, and internally specified constraints. Management clearly reflects exactly what, and all that, the operational model can be configured to do. No more, no less. This precision and flexibility can have associated costs. For example: a 'bridge port' is the service access point that supports a bridge's MAC Relay Entity while a 'port on a bridge' is the external socket that accepts a physical connector; a unappealing distinction for the users of simple equipment. The YANG notation of augmentation is a step away from the faithful transcription of modelling detail into configuration commands. Interface stacks can, in simple cases, be replaced by a single interface with additional attributes, simplifying the configuration of related variables. The management paradigm changes from one of uniform capability, a single solution for all needs, to one of utilitarianism[10], with the simplest solution for the most common case and a possible cost of introducing alternatives to cover the full set of requirements.[11]

3. UML for basic LAG

Link Aggregation can benefit from a utilitarian approach. In most systems each Aggregation Port is paired with an Aggregator (its 'home Aggregator'), with each member of the pair having the same actor and administratively assigned partner values for System Priority, SystemID, and Key. Figure 1[12] illustrates the default operation of the aggregator Selection Logic. If an Aggregation Port has selected another Aggregator, then its home Aggregator has not been selected by any other Aggregation Port.

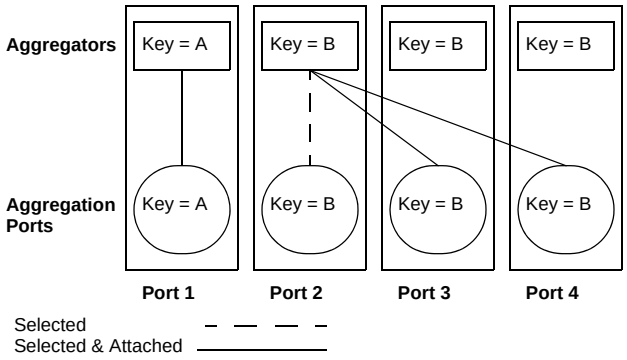

Figure 1—Selection of Aggregators

We can avoid duplicating Aggregator and Aggregation Port information by associating all the LACP configuration information with one of the pair (the Aggregator is the obvious choice), reducing the potential for misconfiguration and the difficulty of synchronizing changes.[13] The **lag** augmentation to **ietf-interfaces.interfaces** provides the management parameters for an Aggregator (port), as shown on the left of UMLish model in Figure 2[14], and includes a reference (my-aggregation-port) to its paired Aggregation port. The **lag-statistics** augmentation adds information on LACP operation to **ietf-interfaces.interfaces.statistics**. Ignore, for the present, the **lag-cscd**, **lag-drni**, and related augmentations. The management parameters for an Aggregation port are shown on the top right of the figure. The **lag-aggregation-port** augmentation adds just a reference to the port's 'home' Aggregator, and no additional statistics are required. All the basic interfaces and statistics parameters are shown for this port. The Aggregator has these same parameters (in the figure they are abbreviated to save space): its lower-layer-if lists the if-indices of the Aggregation Ports currently attached to the Aggregator.

[8]This has been a problem in the past. MIBs have defined management of what those advising the MIB developers have thought the specification should have been, not what is actually in the standard.

[9]While implementations are not overconstrained, their observable external behavior has to match that of the model.

[10]Greatest good for the greatest number, otherwise the tyranny of the majority (see J. S. Mill).

[11]Complex interface stacks provide examples. A bridge port can be a separate interface that uses a lower Ethernet interface, or it can be an augmentation of the latter. Adding link aggregation forces a separation of those two interfaces, and offers a choice as to whether or not the link aggregation parameters are themselves parameters of a separate interface (anticipating some future change, perhaps) or an augmentation of the bridge port.

[12]Taken from the standard.

[13]These changes do not prevent the use of the model with other Selection Logic choices: if fewer Aggregator than Aggregation Ports are required, each surplus Aggregator interface has an admin-status indicating that it is disabled (with the interfaces.description indicating that setting enabled will have no effect on admin-status), with its parameters still supporting its Aggregation Port; if operational practice requires dynamic identification of physical links (which can be plugged into physical ports in no particular order) while each Aggregator has permanently configured attributes (associated with other interface augmentations, such as those for a Bridge Port), the Aggregation Port to Aggregator pairing can be changed.

[14]This figure has been drawn using Framemaker's own graphics to allow Framemaker cross-references to be included..

A UML model for link aggregation management

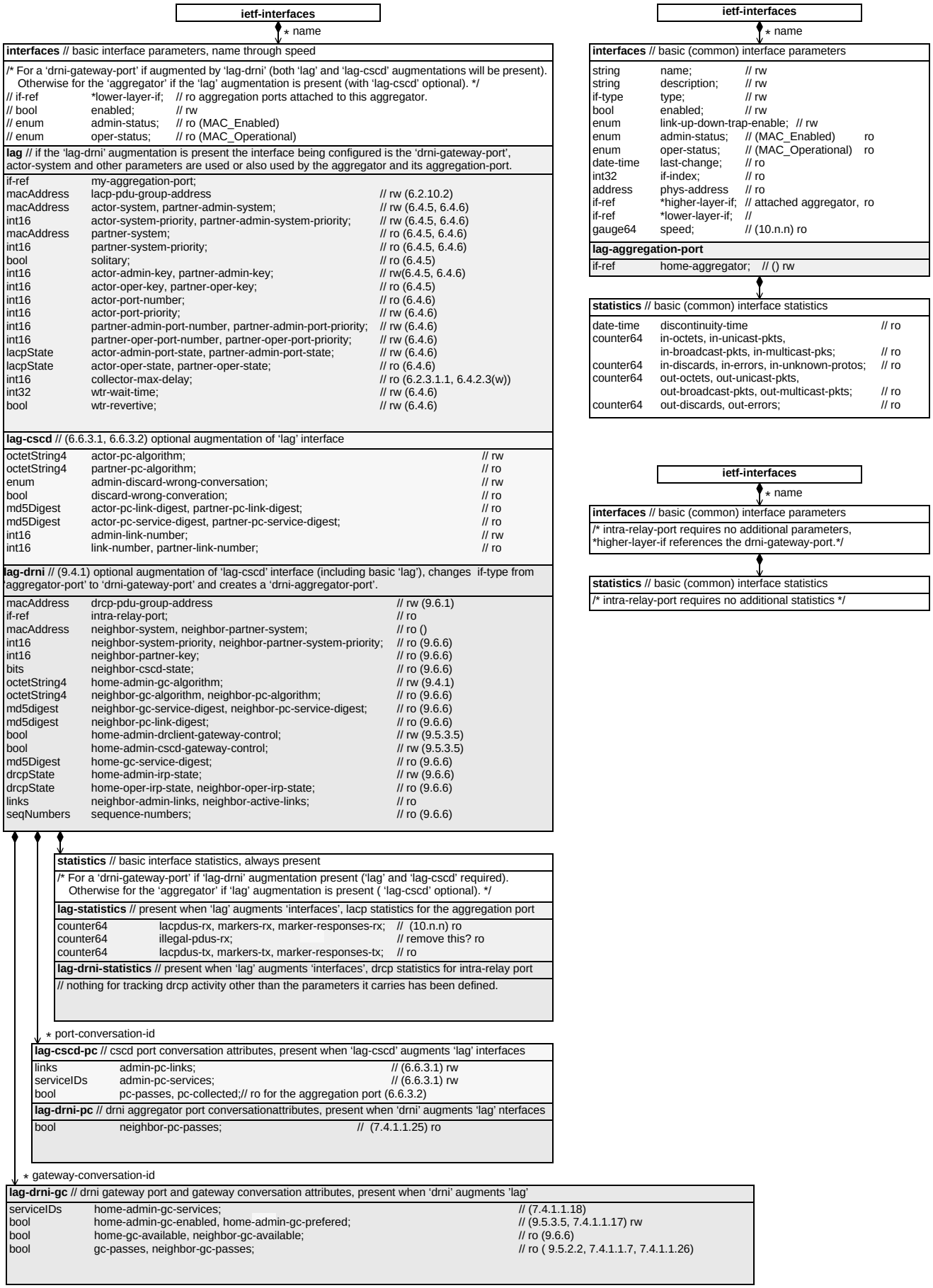

Figure 2—Link Aggregation management (draft)

Basic LAG attributes are specified, with few exceptions, in 1AX clauses 6.4.5 (aggregator variables) and 6.4.6 (aggregation port variables). Some have obvious mappings to **interfaces** attributes, e.g:

Aggregator_Enabled : enabled[15]

Aggregator_Operational : oper-status

Aggregator_MAC_Address : phys-address

Aggregator_Interface_ID : if-index

I have chosen names for **lag** attributes that should make their mapping to the individual variables in 6.4 obvious (changes include removing case-stropping and substituting hyphens instead of underscores, to align with general YANG conventions, and in some cases abbreviation where that does not introduce ambiguity). Some have (for the reasons detailed above) counterparts in both 6.4.5 and 6.4.6 and sometimes but not always the names of these counterparts can differ slightly, e.g. Actor_System (6.4.5) and Actor_Port_System (6.4.6). These differences are unimportant as the distinction between admininistrative and operational values remains clear.

I am not suggesting that an implementation attempt to operate using a single copy of each variable that appears in both 6.4.5 and 6.4.6, only that a remote manager does not need to distinguish them. Any visible differences would be the result of comparing management snapshots taken while the LACP state machines have pending transitions that are immediately actionable, e.g. an aggregation port has received a LACPDU with indicating a change of partner but has not yet been detached from its current aggregator[16]. It is the business of the state machine to deal with such discrepancies, but exposing them to an administrator would only cause confusion.

If an agggregation port has not selected its ‘home’ aggregator, no other aggregation port will select that aggregator unless the aggregation port (interface) has been disabled, so there is no problem with recording LACP information received on a particular aggregation port in the **lag** augmentation. Aggregation port specific parameters such as actor-oper-state and partner-oper-state can always be kept in the ‘home’ **lag** augmentation referenced by **lag-aggregation-port**.home-aggregator, even if the aggregation port is not attached to that aggregator. Further, if another aggregation port has selected the same aggregator, it will necessarily have the same actor-system and partner-system, so there can be no confusion about the values of those parameters.

I have not yet added anything to the UML to correspond to the Aggregation Port Debug information class. This is mainly because I doubt that the equipment end user will ever wish to use YANG to perform the sort of debugging that the equipment vendor should have performed in QA. Protocol implementation errors that turn up in the field are (if present) likely to be very obscure and require vendor diagnosis of a full memory dump.

4. UML for CSCD

Conversation-sensitive collection and distribution capability is managed by the **lag-cscd** augmentation, with attributes matching variables defined in 6.6.3.1 (per aggregator) and/or 6.6.3.2 (per aggregation port). In most cases the correspondence is clear though I have tried to avoid very long names. After reviewing the way that these attribute names are also used with DRNI I have (in most cases) abbreviated ‘port-conversation’ to ‘pc’[17], even where D1.0 simply uses ‘port’ e.g. ‘actor-pc-algorithm’ rather than ‘actor-port-algorithm’. Other points include:

actor-pc-algorithm covers Actor_Port_Algorithm and Actor_Oper_Port_Algorithm. There is no need to manage the latter separately. If the aggregation port is not attached to an aggregator (its home aggregator cannot be enabled) it takes the special value "Unspecified", but there is no need to remind the administrator of that rule. The **lag-aggregation-port**.higher-layer-if provides the attachment status.

*port-conversation-id.lag-cscd-pc.admin-pc-services are entries in the Admin_Conv_Service_Map.[18]

*port-conversation-id.lag-cscd-pc.admin-pc-links are entries in the Admin_Conv_Link_Map.

partner-pc-algorithm,
partner-pc-service-digest,
partner-pc-link-digest
I think the D1.0 spec needs to change. Just acting on

[15]Aggregator_Enabled (read-write) is said incorrectly (in .1AX-Rev/D1.0) to map to the read-only MAC_Enabled parameter of the ISS. The value of MAC_Enabled is determined by controls specific to the entity providing the service (11.2 of 802.1AC-2016), and thus takes into account the result of any internal controls that would prevent the service being provided. This would include of course (but not be limited to) the setting of Aggregator_Enabled. The YANG interfaces model provides an optional enabled (read-write) in addition to the required admin-status (read-only). The

[16]Or other aggregation ports that should no longer be attached to this aggregator have not yet been removed.

[17]With DRNI similar variables use ‘gc’ for ‘gateway-conversation’, e.g. neighbor-gc-algorithm..

[18]I amnote sure what the YANG/NETCONF rules are for manipulating such an array of sets, where a specific item (in this case a Service ID, can be present in only one set.

the latest received on any link for these parameters implies ignoring the packet loss that can follow partner misconfiguration. Better to record each received value in the 'home' interface for each aggregation port (making only these values visible to management is sufficient, I have omitted 'oper' from the UML names for brevity) and do not turn on DWC unless all these received values match on an aggregator's 'active' links, i.e. those for which the actor is collecting.

There are no attributes directly corresponding to the following:

Active_LAG_Links. Although this list is used by the state machines, it is not of direct interest to a manager: *lower-layer-if provides a list of the currently attached aggregation ports.

Conversation_Port_Vector: The information that this provides is available in *port-conversation-id.lag-cscd-pc.pc-passes (Port_Oper_Conversation_Mask renamed) for each aggregation port active in the lag. 'pc' is used throughout **lag-cscd-pc** for 'port conversation'.

actor-conversation-link-digest covers both Actor_Conv_Link_Digest and Actor_Oper_Conv_Link_Digest. There is no need to manage the latter separately. If the aggregation port is not attached to an aggregator (its home aggregator cannot be enabled) it takes the special value 'Unspecified', but there is no need to remind the administrator of that rule.

*port-conversation-id.lag-cscd-pc.pc-collected has been added although it only appears in the state machine detail of 6.2.7.1.2 (and in 7.3.2.1.26). It may be needed for DRNI. To be checked.

4.1 CSCD statistics

1AX-Rev/D1.0 doesn't appear to have defined any statistics for CSCD operation. The descriptions of aAggFramesDiscardedonRx (7.3.1.1.26) and aAggFramesWithRxErrors (7.3.1.1.28) makes it hard to tell them apart. They would map to interfaces.statistics.in-discards and in-errors respectively. This is a clear case for being specific in the state machines and their procedures as to which (if any) counters are incremented, rather than describing relying on a separate description of the counted condition. There is no counter that is specific to DWC discarding wrong conversations so it is not possible to tell how much, if any, discarding when links change was due to having DWC enabled.

I don't think anything more needs to be captured for LACP operation with CSCD, but the absence of per conversation or even per service information is surprising. The mapping of conversation IDs to links might be done purely on a SLA basis, but I would have thought that there would be some scenarios in which the bandwidth actually used by a service or services associated with a given conversation ID would be used as a guide for future rebalancing. Such information might be provided by a queuing model, separate from LAG (and .1AX) but aware of the service to link mapping and able to take advantage of link capacity without significant head-of-line blocking.

5. UML for DRNI

The DRNI (Distributed Relay Network Interface) operation of one of a pair of DRNI Systems is managed by the **lag-dnri** augmentation. A given system can participate in more than one pairing—though not with the same neighbour—supporting a single, separate, DRNI gateway port in each DRNI system for each pair.[19] This makes it convenient to associate all the DRNI parameters with the gateway (port) interface. The gateway port also makes use of, or incorporates,[20] a single aggregator as previously described for LAG and CSCD management so can be created as an augmentation of that aggregator (port).

The **lag-dnri** augmentation is applied to only one of the Aggregator's potentially aggregated ports.[21] Most of the necessary configuration parameters are described in 9.4.1 of .1AX-Rev/D1.0 and have an obvious mapping to the UML attributes, points worth noting include:

The basic interface parameters (admin-status, oper-status etc.) and interface statistics now apply to the gateway port, not the aggregator port. The latter is effectively embedded within the management scope of the DRNI port. This is

[19]This makes the term 'DR-sublayer' rather inappropriate for the separate entities support each DRNI gateway port. 'DRNI Entity' would be better cf. 'MACsec Entity' in .1AE, and 'Port-Access Entity' (PAE) and 'Key Agreement Entity' in .1X, even if most DRNI Systems will participate in a single pair and be configured to provide only one gateway port to a single Distributed Relay Client (DRC). It would be possible to change the DRNI design to allow for multiple DRCs, each with its own pair of gateway ports, in a single pair of DRNI Systems but there appears to be no market demand. Optimizations proposed in this note take advantage of the lack of that complication.

[20]As 802.1AC points out the term 'port' can be applied to an interface and to the entity or entities (the interface stack) supporting that interface. In the present case considering management of a DRNI gateway port to encompass its supporting aggregator is far from a stretch as the DRNI operational procedures reach both into the management state of its own supporting aggregator and that of its paired neighbour's aggrgator.

[21]The prior version of this note included UML references to 7.4 of 1AX-Rev/D0.4, I have removed these consistent with suggestion that references should only be to text that specifies LAG operation and not indirected through existing management clauses (and increasing cross-reference table maintenance).

consistent with the DRNI use of both the home and neighbour's aggregator state. The aggregator port statistics can be deduced by summing the gateway and intra-relay port statistics.

There is (I believe) no need for the 9.4.1 DRNI System Identifier[22], or the Distributed Relay Number. The Actor System Identifier is sufficient, with the higher priority of the paired DRNI Systems taking the role associated with Distributed Relay Number if necessary. When the intra-relay connection (IRC) is operable, the Aggregation Ports that could be aggregated to support the gateway port (i.e. all those with a key value matching that of the gateway port) all use this higher priority Actor System Identifier in the LACPDUs they transmit. If the IRC fails, the lower priority system reverts to using its own Actor System Identifier, thus ensuring that its partner is not misled into believing that links to the now separated DRNI Systems can be aggregated.[23]

The neighbor-system and neighbor-system-priority are the received values of the partner's actor-system and actor-system-priority. It's currently unused, other than to restart computations when it changes (but see suggestion above).

The neighbor-port-algorithm is the received value Neighbor_Aggregator_State.Aggregator_Port_ Algorithm.

*gateway-conversation-id.lag-drni-gc.gc-passes refers to the Home_Gateway_Mask bit for the gateway conversation (cf. 7.4), and is true if the referenced conversation passes through the home gateway port.
Similarly gateway...gc.neighbor-gc-passes refers to the neighbor's gateway.

home-admin-drclient-gateway-control matches 'network gateway control' (in AX-Rev/D1.0). The standard mentions the possibility of various types of DR Client, so the name for this control should not limit it to network protocols[24].

...lag-drni-gc.gc-enabled rather than ..lag-drni-gc.gc-enable to match the use of 'enabled' as a read-write control in the basic interfaces parameters. *I am unclear about the relationship between this variable and the gateway 'available' result.*

...lag-drni-pc.neighbor-pc-passes is true if the port conversation identified passes through the neighbour's aggregator i.e. it maps to the relevant bit in the Neighbor-Aggregator_Mask.

...sequence-numbers has been included, *but I believe these could be significantly simplified.*

[22]I guess that part of the perceived need arose from the prior 'third system' model. However if the Distributed Relay Client requires a separate identifier, possibly to indicate reachability to the same distant system in network protocol even if the IRC fails, that is the DRC's business and out of our scope.
[23]So no obvious reference, and I have not supplied one
[24]It took me some time to figure out what 'network control' meant in this context.