LACP Simulation

Mick Seaman

This note is a guide to a simple LACP simulation. A number of simple test cases are included, and the code can be compiled using Visual C++ as a stand alone executable running on NT4.0. The code may be copied, modified, and run for the purpose of commenting on LACP or any other IEEE 802 related development provided that it is understood that (a) no warranty or fitness for any purpose whatsoever is implied (b) it does not represent any 'official' or 'approved' interpretation of the protocol (c) any modifications are clearly identified if any attribution is made (d) I am not supporting it for product implementation – it was created simply to check the protocol design.

The code referred to here can be found on the 802.1 ftp site, along with this document in the directory /docs99/lacp_model. Since this is a development testbed there are many loose ends, this note draws attention to these, hopefully to avoid confusing the casual reader.

Overview and Organization

The model implementation and simulation environment is structured as follows:

1. A 'component interface' provided by LACP to the rest of the environment. It creates 'systems' operating LACP and creates and adds LACP capable ports to those systems.

   Files: lac.h, lac.c

2. The LACP implementation proper, comprising files that implement the state machines and functions described in Clause 43.3 of P802.3ad D3.0

   Files: lac_types.h, lac_machines.h, lac_rx.c, lac_mux.c, lac_tx.c, lac_defaults.h, lac_options.h.

3. A test stub for the hardware specific aggregator functions that LACP controls.

   Files: lac_hw.c

4. A test harness that provides a test environment within which the LACP implementation runs.

   Files: lact.c, lact_lan.h, lact_lan.c

5. An 'operating system' or simulation environment used by the implementation and test harness to provide timer functions, PDU transfer etc.

   Files: sys.h, sys.c

Each of these is described in more detail below as a commentary on the source code. For brevity's sake information that is readily apparent from the source files is often omitted. To avoid having too many forward references the operating environment is described first.

Operating Environment

The simulation uses a very simple operating environment (sys.h, sys.c). This provides for component and state machine scheduling and the flexible interconnection of components. The latter is achieved through instances of a 'node' data structure that comprises pointers to a 'user', a 'provider', and a node 'owner', as well as 'receive' and 'transmit' function pointers. The receive function handles the 'owner's' processing of pdus from the underlying 'provider' node destined for a 'user' node; and the transmit function handles the 'owner's' processing of pdus from the 'user' for the 'provider'.

A 'next' pointer in each node allows the owner to collect nodes together in a multi-node data structure, and a 'pnext' pointer allows the underlying provider to add the owner's nodes to a provider's root node, e.g. a node representing a LAN. Together these support an overall system convention that any component that can be created can be used in the system without later failure due to the limited size of a fixed index array or similar structure. Nodes also contain receive and transmit status functions, as well as separate administrative and operational transmit and receive states.

The operating environment allows protocol components and state machines to schedule themselves, providing timers that call back a supplied function with a supplied argument pointer. No messaging functionality is provided between components or machines; all reception is synchronous with transmission[1]. To avoid calling loops, the system wide convention is that transmission only occurs after a component schedules itself.

The operating environment contains built in support for simple LAN constructs, comprising a single node that represents the LAN. To this further 'mac' nodes can be added, chained through the 'pnext' pointer, and with their 'provider' pointers pointing to the LAN node. A transmit call to one 'mac' node results in reception at all others.

[1] As currently put together this means that the simulation is not capable of representing 'crossing' messages. Explicit reception queuing prior to the LACP is what is required.

LACP Implementation

The LACP Port

The major data structure in the implementation is the LACP port (lac_port in lac_types.h). In addition to operational and administrative versions of LACP information (system, key, port, state) for partner and actor, this records the expected variables for the LACP state machines: port_enabled, lacp_enabled[2], selected, matched, attach (mux_request), attached (mux_state), and ntt.

The selected variable simply takes the values TRUE or FALSE. The STANDBY state is represented by a separate standby boolean variable, relevant only when selected is TRUE[3].

Separate (re)initialize and port_moved variables are not included. The former is treated as an event (a call to the component interface), and the latter is handled as part of receive processing (albeit as part of receive processing for a port other than the port receiving the port_moved signal). Similarly a 'ready' signal is not required, but is supplied by the timing down counter attach_when. This corresponds to the wait_while_timer. In the main timing counters function as multiples of the tick_timer, notionally one second, and the indicated conditions become true once the count reaches zero. The current_while counter indicates CURRENT while it is non-zero, and the periodic_after counter counts up prompting a periodic transmission when it reaches a fast or a slow periodic time count as appropriate.

This implementation has exactly one aggregate port per physical port and the necessary data is held within the LACP port data structure: 'aport' points to the selected aggregate port, and 'alink' allows ports selecting the same aggregate port to be link into a ring. This one to one correspondence between (physical) ports and aggregate ports means that a port can always select an aggregate port - itself. It removes the need for holding actor and partner parameters separately for the aggregate port, and simplifies key management.

[2] In line with the discussion at the interim meeting, the model only provides for the setting of 'lacp_enabled' to be changed when the port is reinitialized. It is anticipated that a real implementation would decide to do this on the basis of auto-negotiation parameters including whether the link is half-duplex or full-duplex. The draft standard does not permit the operation of LACP on half-duplex links.

[3] This differs from, but is functionally equivalent to, the ennumerated variable UNSELECTED, SELECTED, STANDBY deined in the standard.

However attaching to an aggregator may always be subject to delay. LAG information for a port can change while other ports continue to attach to the aggregator on the basis of the previous LAG. These other ports may take time to detach from the aggregator before it can be used again by its 'own' port. Accordingly attachment to (as opposed to selection of) an aggregator is modeled not by the variables 'aport' and 'alink', but by a separate mux 'node' within the LACP port data structure.

System Identifier

For ease of coding and comprehension, this model represents System Identifiers by simple integers, rather than MAC addresses.

Signaling Considerations

In P802.3ad, signaling between state machines is represented purely by shared variables. By 802.3 convention each independent state machine operates simultaneously and infinitely fast. The present model operates strictly by serial execution, and a set of events is caused against each machine by function call. These events and calls are summarized in lac_machines.h.

Receive Machine

The enumerated states of the machine do not include an INITIALIZE state. BEGIN = TRUE is treated as an event as it is subject to an unconditional transition (UCT).

Periodic and Transmit Machines

The actual implementation of the periodic transmission machine does not record NO_PERIODIC, FAST_PERIODIC, SLOW_PERIODIC and PERIODIC_TX states but simply performs the appropriate check on the value of the periodic_after up counter at each timer tick event. The corresponding state nmachine, which is not materially different from that in P802.3ad is given below.

The transmit machine (lac_tx.c) adopts just one of a number of potential approaches to rate limiting the number of LACPDUs transmitted in a given interval. This one enforces a quiet period after some maximum number (3) of LACPDUs have been transmitted. The machine does not transmit immediately after being notified of a 'need to transmit', but schedules transmission after a short interval. This allows a brief period for asynchronous completion of desired changes to the aggregation hardware states, cutting the number of total messages transmitted. Additionally the machine will reschedule itself following a short interval if transmission cannot take place, due to buffer shortages for example.

Churn Machines

These have not been implemented. This is not to deny their utility, simply that their operation does not impact the operation of the basic protocol.

Component Interface

The component interface (lac.h, lac.c) creates 'systems' operating LACP, and creates and adds LACP capable ports to those systems.

A system is represented by a LACP port, number 0, together with a System Identifier and System Priority. Port 0 acts as the anchor point for the ring of ports added to the system.

Test Scenarios

The following test scenarios have been simulated.

1. Single system, single port, initialization scenarios. A LACP actor is connected to a full duplex point-to-point link. The partner port[4] does not implement LACP.
   a) Standard default parameters are used.
   b) The partner administrative setting is Aggregation rather than Individual. Other parameters are standard defaults.
   c) Both the Actor and the Partner are assumed to be Passive (by default only the partner's administrative default is Passive).
   d) LACP is Disabled for the port.
2. Single system, single port scenarios where the full duplex link is connected to a tester that injects 'rogue' LACP PDUs to illustrate LACP's response to such events.
   a) The tester sends a single PDU that does not match the port's own parameters. The port is seen to time out this packet before reverting to administrative defaults and enabling collection and distribution.
   b) A similar scenario, except the rogue PDU's actor is using Long timeouts.
   c) The rogue PDU's Actor claims to have an Individual port and to be in sync and collecting.
3. Two systems, each with a single port, are connected.
   a) Standard default parameters are used.
   b) One actor's administrative setting is Individual rather than Aggregation. Standard default parameters are used for other settings.
   c) Both actor's are administratively set Passive.
   d) Initially one actor only is set Passive, after a while the other is also made Passive.
   e) The two systems are initialized at slightly different times.
   f) One of the connected ports is reinitialized after the simulation has been running for some time.
4. Two systems are connected by two links (two pairs of connected ports).
   a) Standard default parameters are used (each pair of ports on a single systems has the same key value, 1).
   b) One of the ports has the Actor's administrative setting Individual.
   c) The second links is connected after a delay, during which the first link is established.
   d) The second link is connected after a delay and the highest priority port is attached to the second link, this will cause a change in aggregate port .
5. Loopback test between ports on a single system.
   a) One port is connected to another, standard default parameters are used and the keys for the two ports are the same.
   b) Two ports, each with the same key, are connected to two others, whose keys re the same as each other, but differ from the keys of the ports to which they are attached.
   c) Two ports are connected to two others, all retaining the same key.
6. Link failure and replacement tests between two systems, connected by two links. After the two links have been aggregated, one or both of the links are removed and then are reconnected after some time.
   a) The lower priority link is broken, and restored without the attached ports being notified by the MAC via a port disable/enable signal.
   b) The higher priority link is broken and restored, again without explicit MAC notification to LACP.
   c) The lower priority link is broken and restored, LACP is notified by port disable/enable signals.
   d) The higher priority link is broken and restored, LACP is notified by port disable/enable signals

[4] The port attached to the other end of the point to point link.

7. Link failure, addition, and replacement tests. Two systems are connected by one or more links, some of these fail, new links are added, and some or all of the initial links are restored.

a) One initial link. After it fails another link, of lower priority, is added, then the initial link is restored in addition to the added link.

Periodic Transmission State Machine

The model implementation actually uses the following state machine: