A more Robust Tree: Active Topology Maintenance in Reconfiguring Bridged Local Area Networks (STP+)

Mick Seaman, 3Com Corporation

March 6, 1996

1 Summary

This note describes an improvement (called STP+ in this document) to the standard basic Spanning Tree Algorithm and Protocol (STP) for configuring bridged local area networks. This improvement reduces the impact that topology changes in one part of the network may have on the service availability in other parts.

In STP, information propagation races can cause previously Forwarding bridge ports to be put into the Blocking state - even if the network topology local to the bridges turns out to be the same after the new information has fully propagated. After a port is made Blocking twice Forwarding Delay time has to elapse before it can be made Forwarding again. This means there is an unintended and unnecessary denial of service - typically for 30 seconds.

Section 3 provides examples.

It is possible to avoid the problem by careful arrangement of STP parameters - the priorities of Bridges and Bridge Ports, and the Costs associated with Ports. However this procedure is tedious, not widely understood, and is not 'plug and play'.

STP+ is based on the observation that reversion to a previous active topology differs from adoption of a new one - so long as no frames have been forwarded on the new active topology. For a brief interval after the its first receipt of information to transition a Forwarding port to Blocking, any bridge can be sure (to within basic STP probabilities) that no other bridge has yet transitioned a Blocking port to Forwarding. STP+ identifies this interval with a new "Forgetting" state. The port does not forward frames or learn while in the Forgetting state. The Forgetting state is thus identical with Blocking state so far as data traffic is concerned. However, if new information is received which would cause the port to be made Forwarding, a transition to the Forwarding state can take effect immediately. This limits periods of accidental service denial to the duration of the race condition, which is typically very brief.

Section 5 describes the calculation of the duration of the Forgetting state. Section 6 describes the necessary changes to STP.

STP+ requires no additional administrative procedures. Bridges (switches) implementing the suggested improvement are fully compatible and interoperable with 802.1D standard bridges, and can be intermixed freely without topology restrictions. Since there are no changes to BPDUs or basic spanning tree procedures, they should be compatible with all existing bridges that are believed to be .1D compatible - not just those which follow the strict letter of the standard.

Please note: My intention in making information on this Spanning Tree improvement available is to ascertain whether there is interest in P802 in its standardization as an enhancement to 802.1D. I do not wish to imply that it is currently free of patent restrictions. However it is not my personal intention that such patent filings as I am aware of obstruct its potential standardization and subsequent general availability. Mick.

2 Background

This section provides some background for those less familiar with STP and its development.

Media Access Control (MAC) Bridges may be used to connect individual Local Area Networks (LANs) to form a Bridged Local Area Network. These Bridges maintain a simply connected active topology to prevent the duplication or misordering of frames transmitted between stations attached to the Bridged Local Area Network. IEEE Std 802.1D-1990 [1] describes the operation of MAC Bridges in general and the operation of a Spanning Tree Algorithm and Protocol which is used to maintain a fully and simply connected active topology despite the unpredictable addition and removal of Bridges to and from the network.

A Bridge connects to LANs through its Ports. The Spanning Tree Algorithm maintains loop free connectivity of the Bridged Local Area Network by selecting some Bridge Ports to forward frames, and other to block or not forward. Since incorrect selection of a Port to forward frames could lead to loops in the network, which in turn could lead to network overload or protocol malfunction, the Algorithm takes care to avoid such errors. The Algorithm is distributed and its design recognizes propagation delays between Bridges. If information is received by any Bridge that suggests that one of its Ports should block, the transition to a blocking state is actioned immediately, whereas a transition to a forwarding state is delayed, typically by 30 seconds. Thus even temporary receipt of information that indicates blocking could cause a loss of service to stations attached to the network for this period.

The original published description of the Spanning Tree Algorithm [2] (Perlman, 1985) included a waiting period following the receipt of information that would cause a bridge port to block. In this PRE_BACKUP state the port continues to forward and learn from data traffic, while new protocol information can cause an immediate reversion to the FORWARDING state. Thus, in this version of the algorithm, protocol information propagation races do not lead to service denials. However, I know of no implementation of the Algorithm including this mechanism. It has the disadvantages of (a) delaying detection of a genuine loop, which might be caused by the addition of a new network component (Bridge, LAN repeater, or physical link) to the network and (b) necessitating a longer period for transition from a blocking state to a forwarding state - in the event that removal or failure of a component requires spanning tree reconfiguration.

The possibility of a temporary ‘glitch’ in STP propagation, and subsequent denial of service for periods of 30 seconds or so is a recognized problem. It is possible to arrange the parameters of the Spanning Tree Algorithm - the priorities of Bridges and Bridge Ports, and the Costs associated with Ports and connections - to avoid or minimize ‘glitches’. However this procedure is tedious, not widely understood, detracts from the otherwise ‘plug and play’ attributes of the Algorithm and of Bridged Local Area Networks in general, and requires coordination of the administration of the entire network. For this latter reason it is often recommended that the advantages of automatic configuration and loop detection, which the Algorithm provides, be dispensed with when connecting remotely bridged sites or at administrative boundaries in extensive bridged LANs. In these cases it is deemed more important that configuration changes at one site not cause ‘glitches’ and consequent loss of service for a longer period at another site or in another part of the network.

The aim of the enhancement to STP (STP+) described in this note is to extend the applicability and acceptability of autoconfiguring spanning trees as part of increasing the practical use of plug and play solutions.

STP+ does not include a waiting period prior to transition of a Bridge Port into a blocking state. Rather it provides for an improvement to the basic Spanning Tree Algorithm and Protocol to include recognition of an initial period in blocking state during which an immediate return to forwarding is permissible without looping, duplication or misordering of frames. IEEE Std 802.1D-1990 [1] Appendix B describes the calculation method for Spanning Tree Algorithm timer parameters. This note uses and builds on that description (in Section 4).

3 Reconfiguration Examples

Figure 1-a provides a simple example of a situation in which an unnecessary denial of service can occur. The priority order of the bridges is B1 (highest) thru B5 (lowest). All the port Path Costs are identical. Initially B1 is the Root of the spanning tree, and the forwarding path between endstations X (on LAN L2) and Y (on L4) is through B5. The reconfiguration begins with B1 being powered off, and ends with B2 as the new Root. In the final configuration the forwarding path between X and Y is once more through B5. What happens during reconfiguration depends on the precise order of events as determined by processing and transmission delays, and the accuracy of timers. One possible sequence of events is illustrated, with approximate timings based on the default parameters of [1].

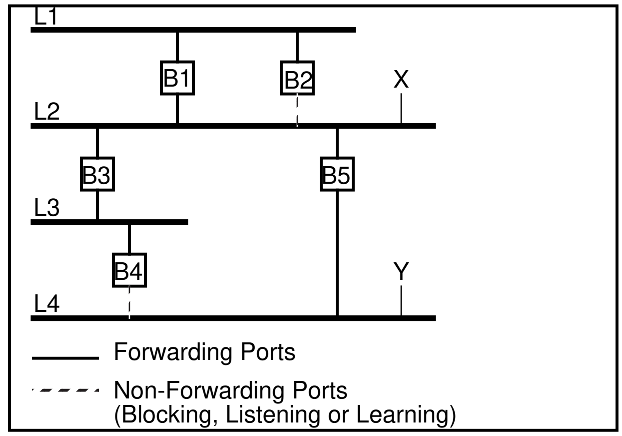


Figure 1-a A hazardous configuration

```
 1:  0.0: B1 power off
 2: 18.3: B4 timeout B1 as Root
            transmit B4 as Root on L3,L4
 3: 18.4: B3 reply B1 is Root on L3
 4: 18.6: B5 reply B1 is Root on L4
 5: 19.1: B2 timeout B1 as Root
            transmit B2 as Root on L1,L2
            B3,B5 ignore
 6: 19.2: B5 timeout B1 as Root
            transmit B5 as Root on L2
            transmit held for L4
            B3 ignores
 7: 19.3: B2 process B5’s BPDU from L2
            reply B2 as Root held on L2
 8: 19.4: B3 timeout B1 as Root
            transmit B3 as Root on L3,L4
 9: 19.5: B4 process B3’s BPDU from L3
            accept B3 as Root
            transmit B3 as Root on L4
10: 19.6: B5 process B3’s BPDU from L2
            accept B3 as Root
            transmit B3 as Root on L4
11: 19.7: B5 process B4’s BPDU from L4
            accept B3 Root,B4 Designated
            make port to L4 Blocking
```

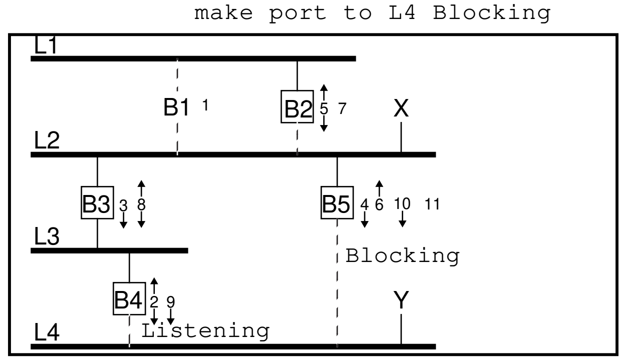


Figure 1-b XY connectivity lost

```
12: 19.8: B4 process B5’s BPDU from L4
            reply B4 as Designated held
13: 20.1: B2 reply B2 as Root on L2
            (previously held)
```

```
14: 20.2: B3 process B2's BPDU from L2
             accept B2 as Root
             transmit held for L3
15: 20.3: B5 process B2's BPDU from L2
             accept B2 as Root
             transmit held for L4
             make port to L4 Listening
16: 20.4: B3 transmit B2 as Root on L3
              (previously held)
17: 20.5: B4 process B3's BPDU from L3
             accept B2 as Root
             transmit B2 as Root on L4
18: 20.6: B5 transmit B2 as Root on L4
              (previously held)
19: 20.7: B5 process B4's BPDU from L4
             reply B5 as Designated held
20: 20.8: B4 process B5's BPDU from L4
             accept B5 as Designated
             make port to L4 Blocking
```

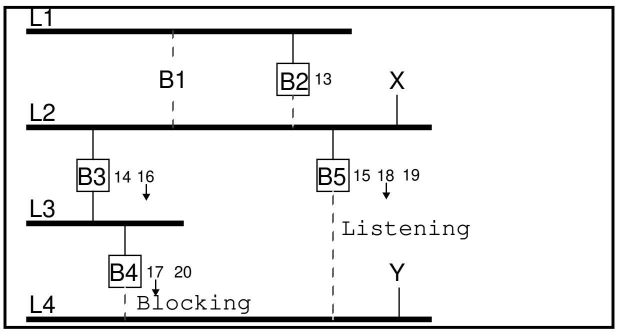


Figure 1-c XY connectivity recovering

```
21: 21.1: B2 transmit B2 as Root on L1,L2
              (Hello Timer Expiry)
22: 21.2: B3 process B2's BPDU from L2
             transmit held for L3
22: 21.3: B5 process B2's BPDU from L2
             transmit held for L4
23: 21.4: B3 transmit B2 as Root on L3
              (previously held)
24: 21.5: B4 process B3's BPDU from L3
25: 21.6: B5 transmit B2 as Root on L4
              (previously held)
26: 21.5: B4 process B5's BPDU from L4
27: 23.1: B2 transmit B2 as Root on L1,L2
              (Hello Timer Expiry)
28: 23.2: B3 process B2's BPDU from L2
             transmit B2 as Root on L3
29: 23.3: B5 process B2's BPDU from L2
             transmit B2 AS Root on L4
30: 23.5: B4 process B3's BPDU from L3
31: 23.5: B4 process B5's BPDU from L4
Events 27 thru 31 repeat at 2 second
intervals until ..
97: 50.3: B5 make port to L4 Forwarding
```

Connectivity between X and Y was lost for 30 seconds.

The priority order of the bridges could have been rearranged to avoid the possibility of temporary denials of service. Figure 2-a gives a nonhazardous alternative.

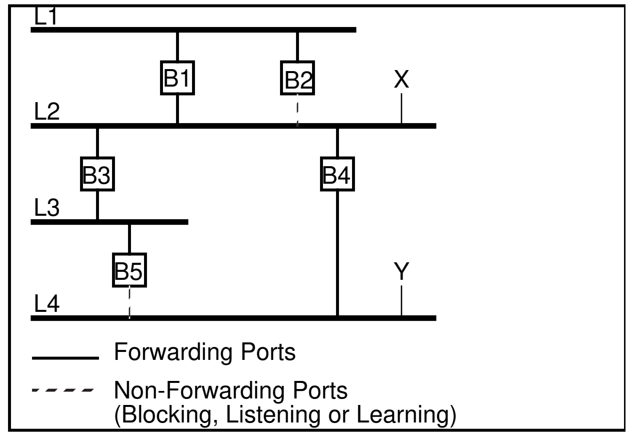


Figure 2-a Hazard-free configuration

B4 will always be a better Designated Bridge for L4 no matter how many bridges are attempting to establish the new topology.

In general, a configuration is hazard free if it meets the following conditions. Given any bridge with two or more ports that will be forwarding once the active topology has stabilized, and taking all possible subsets of the bridges in the network, all the spanning trees computed with these subsets make those ports forwarding. For example, in Figure 2-a, both ports of B4 are forwarding if all the bridges other than B1 are removed from the network, or if all other than B3 and B5 are removed, or indeed if any of B1 thru B5 are included.

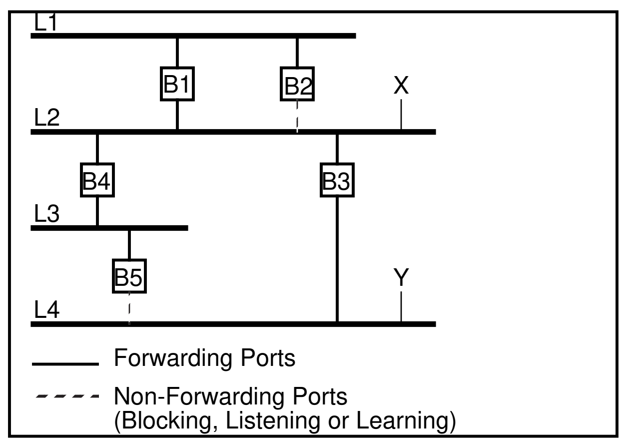


Figure 2-b Alternate hazard-free configuration

The configuration shown in Figure 2-b is also hazard free, though B5's port to L3 can block (consider bridges B3, B4, B5).

Not all configurations can be made hazard free by simply changing bridge priorities. In some Path Costs also need to be set. See, for example, figure 3-a.

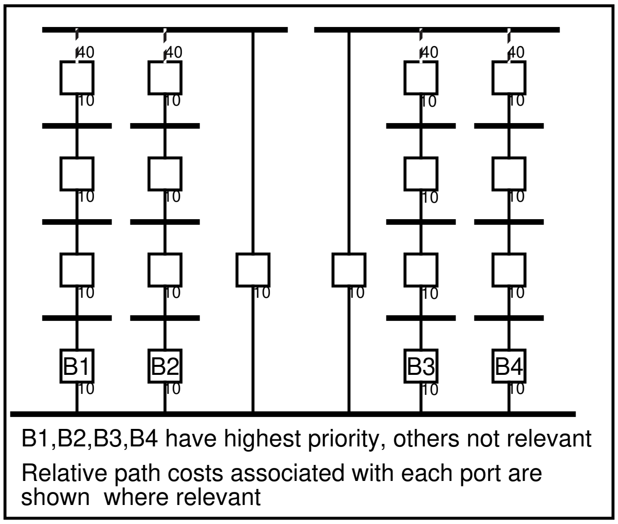


Figure 3-a Hazard elimination by priority and cost assignments

4 Example reconfiguration with STP+

This section uses the first example of Section 3 to show how service denials are minimized by the STP+ enhancement.

The sequence of events previously described remains the same, with two exceptions. At event 11, B5's port to L4 is put into the Forgetting state:

```
11: 19.7: B5 process B4's BPDU from L4
             accept B3 Root,B4 Designated
             make port to L4 Forgetting
```

And at event 15, it reverts directly to the Forwarding state:

```
15: 20.3: B5 process B2's BPDU from L2
             accept B2 as Root
             transmit held for L4
             make port to L4 Listening
```

So the service interruption lasts for only 0.6 second instead of a full 30 seconds, keeping it within the retransmission time limits of almost all higher layer protocols.

5 Calculating the Forgetting Delay

Clause B3.8.2 of IEEE Std 802.1D-1990 [1] describes the calculation of the delay necessary before a bridge adopts a new active topology, i.e. starts forwarding frames on a port which was previously blocking. This delay ensures that there are no longer any frames in the network that were being forwarded on the previous active topology. It is calculated for a worst case scenario where bridges maximally far apart in the network adopt a new active topology following removal of the Root bridge from the network:

2 x fwd_d >= msg_ao + msg_prop + bt_d + life

where:

fwd_d is the STP parameter Forward Delay, and 2 x Forward Delay has to elapse before the port is made forwarding

msg_ao is the maximum Message Age overestimate, i.e. the maximum overestimate of the age of STP information by any bridge in the network

msg_prop is the maximum Message propagation time between bridges in the network

bt_d is the maximum bridge transit delay, the time taken for a data frame to be forwarded through a bridge

life is the maximum frame lifetime in the network of bridges

The STP+ improvement is based on the observation that reverting to the prior active topology is different to adopting a new topology. A bridge can revert immediately to the prior topology if it can be sure (to within the probabilities used by the basic algorithm) that there are no frames in the network that were or are being forwarded on a newer active topology, i.e. that the new topology (or at least one that would require a difference in the forwarding state of the local bridge's ports) has not been adopted by any other bridge.

Using the worst case described in [1] B3.8.2: *msg_ao* is the time difference between two Bridges in the network recognizing the need for a new active topology, and *msg_prop* is the time taken for new protocol information from the later of the pair to recognize the new topology to reach the other. This permits the later bridge to be sure (on recognizing the new topology) that the earlier still has a period of *bt_d + life* (or *2 x fwd_d - (msg_ao + msg_prop)*) to run before it attempts to use the new active topology. So this later bridge can transition a port directly back to forwarding at any time during this period without increasing the risk of looping frames.

For the parameters derived in [1] Appendix B and used in B3.8.2:

fwd_d = 15 seconds
msg_ao = 6 seconds
msg_prop = 14 seconds
bt_d = 1 second
life = 7.5 seconds

leaving an interval of 8.5 seconds in which direct reversion is permitted.

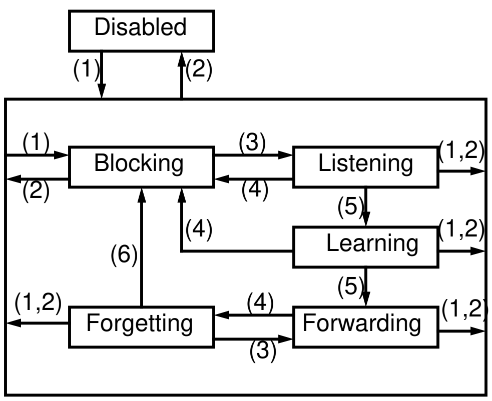
A practical approach might relate the Forgetting Delay to the hello time. It is not necessary to be precise about the Forgetting Delay and I would suggest that it be in the range one to two hello times. This is somewhat less than the upper bound calculate but should deal with protocol information races in almost all practical situations.

It should be noted that the above reasoning makes the normal simplifying assumption that a single event - the addition or failure of a bridge, link, repeater, or power source - has caused the network to reconfigure. STP does not guarantee the maintenance of a loop free topology in the general case of a set of events spaced over time. In particular the injection of continuously improving protocol information into the network over the course of a reconfiguration could cause loops to develop with some probability depending on the precise behavior of the bridge implementations. However, the single event assumption is believed to cover all practical cases with the exception of deliberate denial of service attacks or faulty implementation. I don't know of any attempts to deploy STP on such a scale that it would be a normal occurence for two bridges to be independently failing or powering up within a 60 second window. In this respect the scaling pretensions of STP are considerably less than those of major router protocols.

6 Changes to .1D

This section describes the major changes to .1D to introduce the Forgetting State of STP+. In practice I believe it would be best to make this implementation optional, at least for some transition period. For the sake of clarity I have left out such considerations.

Figure 4-3 of 802.1D is changed to add the Forgetting State.



(1) Port enabled, by management or initialization
(2) Port disabled, by management or failure
(3) Algorithm selects as Designated or Root Port
(4) Algorithm selects as not Designated or Root Port
(5) Protocol timer expiry (Forwarding Timer)
(6) Protocol timer expiry (Forgettiing Timer)

Section 4.4 is updated, and a new Section 4.4.5 Forgetting inserted before the existing 4.4.5 Disabled.

A new Section 4.5.6.4 Forgetting Timer is added specifying a timer per port with a timeout value of twice Hello Time (4.5.3.5), i.e. twice the value being used by the current Root.

Update sections 4.6 and 4.8 to correspond to the code changes following.

Add the forgetting state capability to the pICS as an optional item.

```
/***************************************************************************
* STP+ Code Changes
* NO WARRANTY IMPLICIT OR IMPLIED AS TO ACCURACY OR COMPLETENESS
***************************************************************************/
/***************************************************************************
* DEFINED CONSTANTS
***************************************************************************/
/** port states **/
#defined Disabled   0                               /* (4.4.6) */
#defined Listening  1                               /* (4.4.2) */
#defined Learning   2                               /* (4.4.3) */
#defined Forwarding 3                               /* (4.4.4) */
#defined Blocking   4                               /* (4.4.1) */
#defined Forgetting 5                               /* (4.4.5) */

/***************************************************************************
* STATIC STORAGE ALLOCATION
***************************************************************************/
Timer forgetting_timer[All_ports];

/***************************************************************************
* CODE
***************************************************************************/

make_forwarding(port_no)                                 /* (4.6.12) */
Int port_no;
{
   if (port_info[port_no].state == Blocking)
   {
      set_port_state(port_no, Listening);
      start_forward_delay_timer(port_no);
   }
   else if (port_info[port_no].state == Forgetting)
   {
      set_port_state(port_no, Forwarding);
      stop_forgetting_timer(port_no);
   }
}

make_blocking(port_no)
Int port_no;
{
   switch (port_info[port_no].state)
   {
      case Disabled: break;
      case Listening:
           topology_change_detection();
           port_info[port_no].state = Blocking;
           break;
      case Learning:
           port_info[port_no].state = Blocking;
           break;
      case Forwarding:
           start_forgetting_timer(port_no);
           port_info[port_no].state = Forgetting;
           break;
      case Blocking: break;
      case Forgetting: break;
      default: break;
   }
}
```

```
forgetting_timer_expiry(port_no)    /* (4.7.9) */
Int port_no;
{
   set_port_state(port_no, Blocking);
}

initialize_port(port_no)
Int port_no;
{
    become_designated_port(port_no);                    /* (4.8.1.4.1)  */
    set_port_state(port_no, Blocking);                  /* (4.8.1.4.2)  */
    port_info[port_no].topology_change_acknowledge = False;
                                                        /* (4.8.1.4.3)  */
    port_info[port_no].config_pending = False;          /* (4.8.1.4.4)  */
    stop_message_age_timer(port_no);                    /* (4.8.1.4.5)  */
    stop_forward_delay_timer(port_no);                  /* (4.8.1.4.6)  */
    stop_hold_timer(port_no);                           /* (4.8.1.4.7)  */
    stop_forgetting_timer(port_no);                     /* (4.8.1.4.8)  */
}

disable_port(port_no)
Int port_no;
{
   Boolean root;
   root = root_bridge();
   become_designated_port(port_no);                     /* (4.8.3.1)    */
   set_port_state(port_no, Disabled)                    /* (4.8.3.2)    */
   port_info[port_no].topology_change_acknowledge = False;/* (4.8.3.3)    */
   port_info[port_no].config_pending = False;           /* (4.8.3.4)    */
   stop_message_timer(port_no);                         /* (4.8.3.5)    */
   stop_forward_delay_timer(port_no);                   /* (4.8.3.6)    */
   stop_forgetting_timer(port_no);                      /* (4.8.3.6)    */
   configuration_update();
   port_state_selection();                              /* (4.8.3.7)    */
   if ((root_bridge()) && (!root))                      /* (4.8.3.8)    */
   {
      bridge_info.max_age = bridge_info.bridge_max_age  /* (4.8.3.8.1) */
      bridge_info. = bridge_info.bridge
      bridge_info. = bridge_info.bridge
      topology_change_detection();                      /* (4.8.3.8.2) */
      stop_tcn_timer();                                 /* (4.8.3.8.3) */
      config_bpdu_generation();                         /* (4.8.3.8.4) */
      start_hello_timer();
   }
}
```

```
/** pseudo-implementation-specific timer running support **/
tick()
{
   Int port_no;
   if (hello_timer_expired())
   {  hello_timer_expiry();
   }
   if (tcn_timer_expired())
   {  hello_timer_expiry();
   }
   if (topology_change_timer_expired())
   {  topology_change_timer_expiry();
   }
   for (port_no = One; port_no <= No_of_ports; port_no++)
   {
      if (forward_delay_timer_expired(port_no))
      {  forward_delay_timer_expiry(port_no);
      }
      if (message_age_timer_expired(port_no))
      {  message_age_timer_expiry(port_no);
      }
      if hold_timer_expired(port_no))
      {  hold_timer_expiry(port_no);
      }
      if forgetting_timer_expired(port_no))
      {  forgetting_timer_expiry(port_no);
      }
   }
}

/* where */

start_forgetting_timer(port_no)
Int port_no;
{  forgetting_timer.value = (Time) Zero;
   forgetting_timer.active = True;
}

stop_forgetting_timer(port_no)
Int port_no;
{  forgetting_timer.active = False;
}

Boolean forgetting_timer_expired(port_no)
Int port_no;
{  if (forgetting_timer[port_no].active &&
       (++forgetting_timer[port_no].value >= (2*bridge_info.hello_time))
   {   forgetting_timer[port_no].active = False;
       return(true);
   }
}
```