Making Waves

Mick Seaman

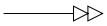
This note describes how the Rapid Spanning Tree Protocol transitions Designated Ports to Forwarding without having to wait for a network round trip delay time, or Forward Delay.

Familiarity with P802.1w Rapid Reconfiguration (D6 and D7) is assumed. The state machine variables used in this description are based on D7, but retain some of the more useful terminology of D6. They are suggested as ballot comment on D7. The diagrammatic conventions are based on those described in “Spanning Tree Visio Standards”.

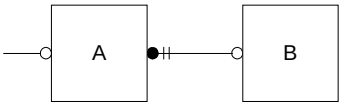
Basic Process

A Bridge ‘A’ with a Designated Port that is not Forwarding, and that is connected to a Bridge ‘B’ by a point-to-point link, sends a BPDU with the ‘Sync Request’ flag set to B. Such a BPDU will be referred to as a ‘Proposal’ and distinguished in diagrams with a double headed arrow:

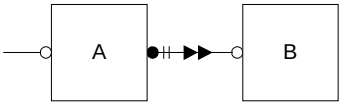
If and when B is ‘Synchronized’ with this ‘Proposal’, B returns a BPDU with the ‘InSync’ flag set to A. Such a BPDU will be referred to as an ‘Agreement’ and shown as a double headed open arrow:



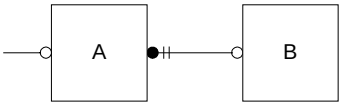
Once a Designated Port on A receives an Agreement that matches its designated priority vector, that port can transition to Forwarding immediately.



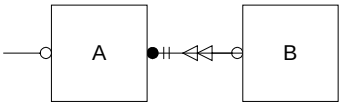
1) The Designated Port A-B is not Forwarding



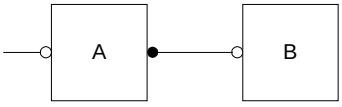
2) A sends a "Proposal" to B



3) B "Synchronizes"



4) B returns an "Agreement" to A



5) A transitions A-B to Forwarding

Synchronization

A Bridge ‘B’ is ‘Synchronized’ with information received on one of its ports, if:

1) The information has been received on a point to point link from a Designated Port, and the receiving port is selected as the Root Port after the received information has been processed, and
2) All other ports (other than the Root Port) on Bridge B are ‘synchronized’.

An individual port (other than the Root Port) on Bridge B is ‘synchronized’ if:

a) It is in the Discarding State, or

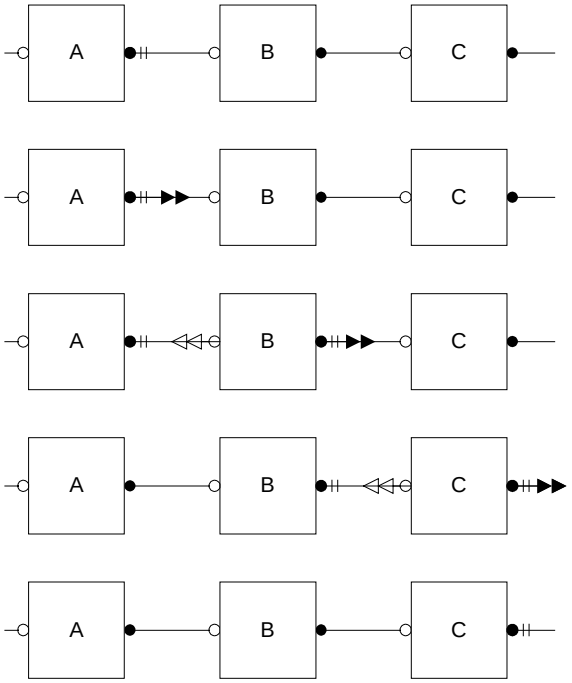
b) It is an operEdge, or

c) It is a Designated Port attached to a point-to-point link, and has received a matching Agreement since the root path priority vector (including the diminutive received port component) last changed.

When a Proposal is received on a Root Port, the Bridge transitions all other non-operEdge ports that are not already synchronized with the root path priority vector implied by the proposal (the message priority plus the port path cost plus the received port identifier) to the Discarding State.

Connectivity Waves

When receipt of a Proposal on a Root Port causes a Designated Port attached to a point-to-point link to transition to Discarding, a further Proposal is sent on that Designated Port. If the Bridge, ‘C’ say, attached to the other end of that link synchronizes and returns an Agreement in its turn, the ‘cut’ in the active topology will move further toward the edge of the network.



This progression of the active topology cut is the ‘wave’ referred in the title of this note.

The Network Edge

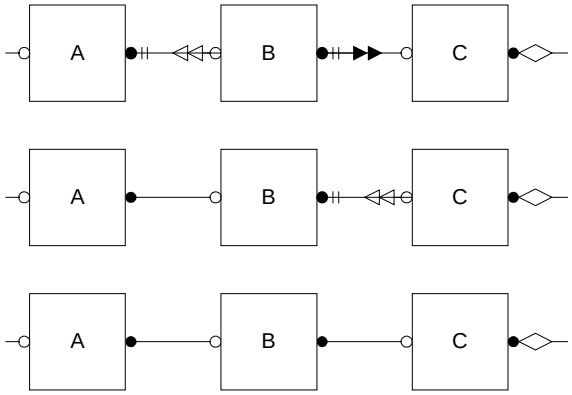
If there were no operEdge ports in the network, the connectivity wave would come to halt at the outermost LANs. The bridge ports attached to those LANs, and more significantly the attached end stations would still experience significant delays in the restoration of network connectivity following reconfiguration.

However if all the ports attached to Bridge ‘C’ above were to be operEdge ports (apart from the Root Port of course), an Agreement can be returned without propagating the cut further.

To allow such scenarios to be easily depicted, it is useful to have a distinguished Visio Format>Line>Begin symbol for an operEdge port, code 38 in the following augmented table..

Port Role	Port State	Visio Code
Designated (operEdge FALSE)	Discarding	36
	Learning	35
	Forwarding	42
Designated (operEdge)	Discarding	n/a
	Learning	n/a
	Forwarding	38
Root Port	Discarding	32
	Learning	31
	Forwarding	41
Alternate	Discarding	25
	Learning	24
	Forwarding	0
Disabled	Disabled	23

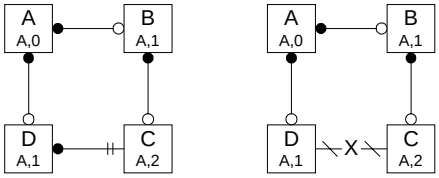
Using this, the last few steps in our preceding example become:



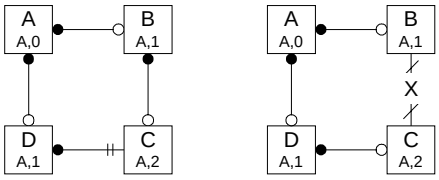
More Waves

In the following four example reconfigurations, four bridges are connected in a ring, and one of the connecting fiber pairs breaks. (the configuration evolves from left to right and top to bottom).

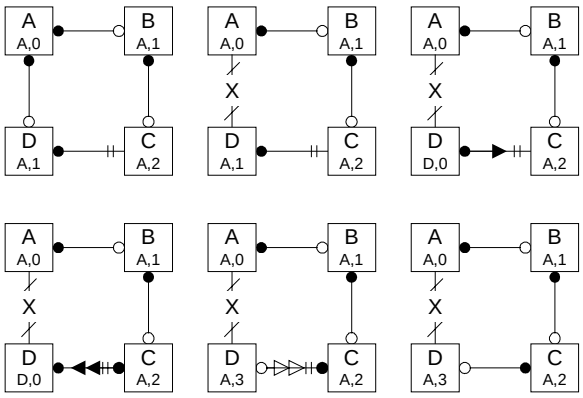
Of course if the break happens to be where the spanning tree has already cut the active topology, no reconfiguration takes place.



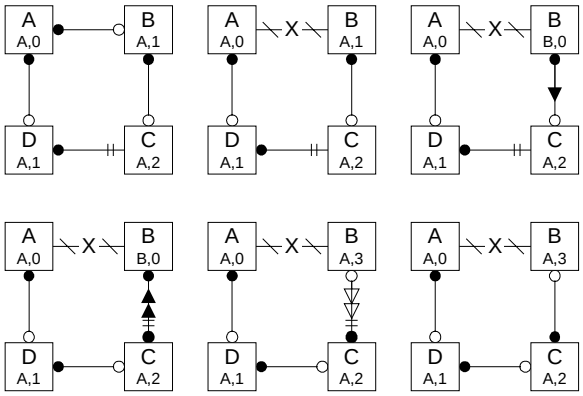
And loss of the other fiber at the bridge that is cutting the active topology is dealt with by that bridge failing over to its alternate root port, without the need for any BPDU transmissions.



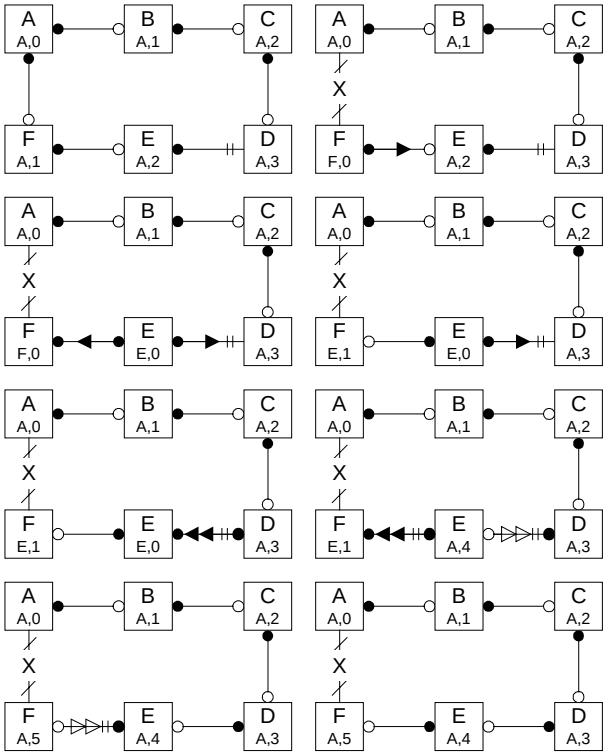
A break in a fiber connection to the Root Bridge, does require the exchange of Proposal and Agreement BPDUs to complete the rapid reconfiguration. In one case ….



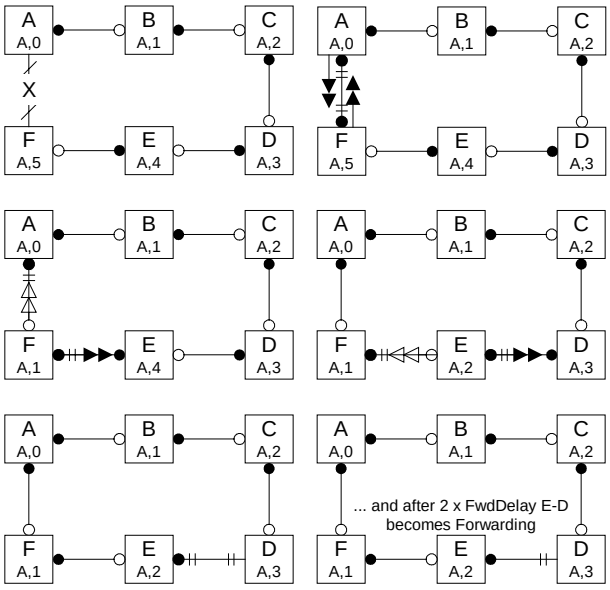
And in the other ….



In none of our ‘ring of four’ reconfiguration examples, does a ‘cut’ have to be moved from one side of a bridge to another. A more interesting scenario can be created with a ring of six bridges, with the fiber cut occurring at the Root Bridge.

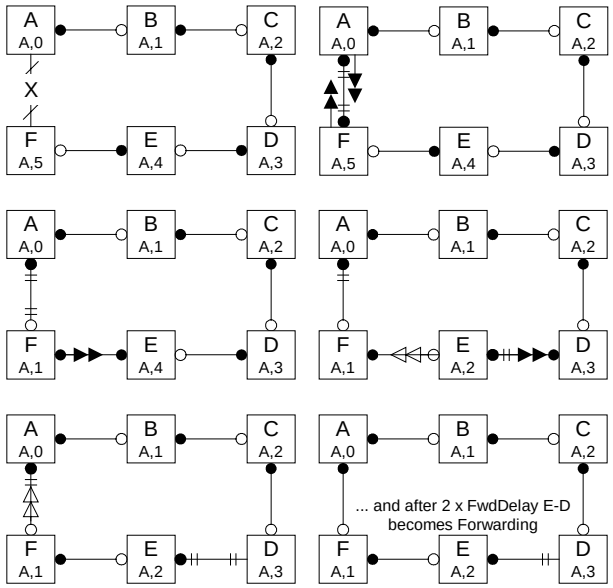


If the link A-F is restored, the network evolves as follows.



A Race

In these network scenarios there are actually two competing processes that allow a Bridge to achieve synchronization and return an Agreement to a previous proposal. The Agreement may be returned after Designated Ports have transitioned to Discarding, or after Agreements have been received on those ports. There is no requirement for a Designated Port to be complete the transition to Discarding before a Proposal is sent. So it is possible, though unlikely that the prior 'ring of six" restoration scenario proceed as follows.



The proposed standard deliberately does not contain any mechanism to increase the likelihood of the reconfiguration proceeding in this way.

Synchronization Process

The synchronization process within each Bridge proceeds as follows[1].

1. If the current spanning tree priority vector for a port is updated, the **synced**[2], **agreed**[3], and **proposing**[4] flags for the port are reset[5].
2. On receipt of a Proposal carried in a better or repeated[6] BPDU the **proposed**[7] flag is set[8]. If the received BPDU is not a Proposal, the **proposed** flag is reset, and it is also reset if the spanning tree priority vector is generated by this Bridge rather than being received from another.
3. If the **proposed** flag is set, the receiving port is the Root Port, and the Bridge is not already synchronized with the received spanning tree priority vector[9], the **sync**[10] flag is set[11] for all ports and the **proposed** flag is reset[12].
4. If the **sync** flag is set for a Designated Port and the **synced** flag is not set, the port is transitioned to Discarding[13].
5. If a Designated Port is not or will not soon be Forwarding (as is obviously the case if it has just been set to transition to Discarding) and the **proposing** flag is not set for the port, it is set[14], and the **newInfo** flag is also set[15] which will cause a Proposal to be sent through this Port.
6. If or once a Port has transitioned to Discarding, its **sync** flag is reset and its **synced** flag is set[16].
7. Once the **synced** flag is set for all ports other than the Root Port, **synced** will be set[17] for the Root Port and an Agreement will be sent through that Port.
8. On receipt of an Agreement, the **agreed**[18] flag is set and the **proposing** flag is reset.
9. If or once a Designated Port has the **agreed** flag set, its **sync** flag is reset and its **synced** flag is set[19], and it is transitioned to Forwarding[20].

---

[1] Each variable change is annotated with <state machine>:<state name>. The following acronyms are used for the state machines. PI : Port Information, PR: Port Role Transitions. All the footnotes as to what happens in each state machine below refers to what the state machines should contain, as suggested below, not to D7.

[2] **synced** is called **agree** in D7. I suggest the change so that these variables consistently follow the pattern where the command 'act' connotes a request to which the response is 'acted'.

[3] **agreed** in D7.

[4] **proposing** is called **propose** in D7. I suggest the name change because the variable is actually used not to record the fact that a proposal is about to be made, but that one has been made – is in progress so to speak – so that newInfo is not asserted repeatedly.

[5] **synced**, **agreed** and **proposing** are reset in PI:BETTER and PI:UPDATE.

[6] As defined by the result of the rcvdBpdu() procedure, rcvdMsg == BetterDesignatedMsg or rcvdMsg == RepeatedDesignatedMsg respectively.

[7] **proposed** is called **proposed** in D7 and **syncReq** in D6.

[8] **proposed** becomes recordProposed() in PI:BETTER and PI:REPEAT, and is reset in PI:UPDATE.

[9] In which case **synced** will be set for the Root Port.

[10] **sync** is called **agreePort** in D7.

[11] PR:AGREE_BRIDGE setAgreeBridge() in D7. I suggest renaming the state and the procedure to PR:ROOT_PROPOSED and syncBridge().

[12] If the bridge is already synchronized, proposed is reset and an Agreement is sent through the Root Port as in step 7.

[13] PR:DESIGNATED_LISTEN.

[14] PR:REQUEST_AGREEMENT in D7. I suggest renaming this state to PR:DESIGNATED_PROPOSE.

[15] To cause a BPDU to be sent.

[16] Both in PR:BLOCKED_PORT and in PR:DESIGNATED_AGREED (D7 name). I suggest renaming the latter of these to PR:DESIGNATED_SYNCED

[17] PR:ROOT_AGREED

[18] PI:CONFIRM.

[19] PR:DESIGNATED_SYNCED.

[20] PR:DESIGNATED_LEARN followed by PR:DESIGNATED_FORWARD as necessary.

Synchronization States

A principal purpose of this note is to improve upon and correct some of the changes that have been made to the P802.1w state machines from D5 through D7 in this area. The original set of variables names were hard to grasp[21], and this has led to incorrect changes being made in the face of both real and apparent bugs.

Here I propose to describe the purpose of each state in the machines that relates to the bridge to bridge synchronization process, as partitioned into its proposal, internal synchronization, and agreement components. The D7 conditions that lead to entry into the state and the actions in the state are presented using further revised variable names[22], and a set of corrections are suggested using those names.

All transitions in the Port Role Transition, Topology Change, and Port Transmit state machines should be qualified by:

```
&& selected && !updtInfo
```[23]

otherwise there is a risk that inconsistent data is sent. In the transmit state machine, for example, a Designated Port that has received a 'better' PDU should not transmit until the role has been updated and, if the role becomes Alternate Port, should not transmit at all. Equally a port that has become a Designated Port should not transmit until the information associated with the port has been updated by PI:UPDATE.

PI:DISABLED

This state initializes some of the variables related to the synchronization process. Its currently defined actions are in line with comments below that suggest that the responsibility of clearing agreed and proposed should be done by the Port Information state machine rather than by the Port Role Transition state machine. No modifications appear to be required to PI:DISABLED D7, which currently reads:

```
rcvdBpdu = rcvdRSTP = rcvdSTP = FALSE;
portInfo = myPtInfo();
updtInfo = FALSE;
agreed = proposed = FALSE;
infoAge = 0; infoIs = Disabled;
reselect = TRUE; selected = FALSE;
```

[21] For which the present author accepts full responsibility.
[22] As introduced in footnotes above.
[23] And not by "&& !selected" as shown in D7.

PI:AGED

This state is unrelated to the synchronization mechanisms. Its actions should remain as in D7. However the transition into this state from PI:CURRENT should be qualified by **selected**[24], and should become:

```
selected && (infoIs == Received) &&
(infoAge == 0) && !updtInfo &&
!rcvdBpdu
```

and the transition from PI:AGED should also be qualified by **selected** rather than !**reselect**[25] and become:

```
selected && updtInfo
```

[24] Indicating that port role selection has completed and produced a result since the information last changed on this port and is not the result of a long delayed response by this machine to something that happened a while ago.
[25] !reselect simply means that the port role selection process does not need to be restarted from the beginning, not that it has completed

PI:BETTER

This state processes a received BPDU with a different priority vector than the current priority vector for the port. The received priority vector will replace the current priority vector for the port, either because it has a higher priority, or because the BPDU has been transmitted by the designated port for the attached LAN.

Once the received priority vector becomes the current priority vector, any prior agreement between the port role, port state, and the spanning tree information for the receiving bridge and its neighbour is suspect. Both the **synced**[26] and the **agreed** flags need to be reset. Moreover any proposal the port has recently made to its neighbour will have carried out of date priority vector information, so the **proposing** flag also needs to be reset.

PI:BETTER D7 reads:

```
portInfo = recordStpInfo();
updtInfoAge();
agreed = synced = FALSE;
proposed = recordProposed();
infoIs = Received; reselect = TRUE;
selected = FALSE;
```

PI:BETTER should read:

```
portInfo = recordStpInfo();
updtInfoAge();
agreed = proposing = synced = FALSE;
proposed = recordProposed();
infoIs = Received; reselect = TRUE;
selected = FALSE;
```

The transition to PI:CURRENT should become unconditional since qualification of transitions from PI:CURRENT to PI:AGED and PI:UPDATE suggested elsewhere in this note ensure that information is up to date before those occur:

```
UCT
```

The prior qualification of the transition by **!reselect**[27] does not ensure consistent up to date information in the PI:CURRENT state. Replacing it with **selected** doesn't help because that would mean each received BPDU would have to be processed right through role selection, whereas if two BPDUs are received with the latter being 'better than' the former the first of the two can be discarded without a bridge wide recomputation. This helps if, for some reason, a bridge with a very large number of ports comes under heavy protocol processing load.

[26] If the synced flag should be set because the port is Discarding, the Port Roles Transition machine should see to that.

[27] Because !reselect simply means that the port role selection process does not need to be restarted from the beginning, not that it has completed. This by the way is entirely my fault, since I proposed an inadequate change to D6.

PI:UPDATE

This state updates BPDU the current priority vector for the port,[28] following role selection by selectRoles(). The designated priority vector calculated from the root path priority vector for the bridge will replace the current priority vector for the port.

Once the received priority vector becomes the current priority vector, any prior agreement between the port role, port state, and the spanning tree information for the receiving bridge and its neighbour is suspect. Both the **synced**[29] and the **agreed** flags need to be reset. Any proposal the port has recently made to its neighbour will have carried out of date priority vector information, as will any proposal a neighbour has made to this port, so both the **proposing** and **proposed** flags need to be reset.

PI:UPDATE D7 reads:

```
portInfo = myPtInfo();
updtInfo = FALSE;
agreed = proposed = FALSE;
infoIs = Mine; newInfo = TRUE;
```

PI:UPDATE should read:

```
portInfo = myPtInfo();
updtInfo = FALSE;
agreed = synced = FALSE;
proposed = proposing = FALSE;
infoIs = Mine; newInfo = TRUE;
```

The transition from PI:CURRENT should be qualified by **selected**:

```
selected && updtInfo
```

PI:REPEAT

This state records any new proposal attached to existing spanning tree information. It is likely to be executed as part of the propagation of the wave of connectivity. I believe it reads correctly in D7:

```
proposed = recordProposed();
updtInfoAge();
```

PI:CONFIRM

This state records the receipt of an Agreement from a Root Port. Its name should be changed to PI:AGREEMENT in line with the general terminology of D7, and of this note. I believe the state's actions read correctly in D7:

```
agreed = TRUE; proposing = FALSE;
```

[28] This will only occur if the port has been selected as a Designated Port, but that fact is not known to the state machine.

[29] If the synced flag should be set because the port is Discarding, the Port Roles Transition machine should see to that.

PR:INIT_PORT

This state initializes some of the variables related to the synchronization process. PR:INIT_PORT D7 currently reads:

```
role = DisabledPort;
synced = TRUE; sync = FALSE;
reRoot = FALSE;
rrWhile = 0; rbWhile = 0;
```

Its currently defined actions are in line with comments below that suggest that the responsibility of clearing agreed and proposed should be done by the Port Information state machine[30] rather than by the Port Role Transition state machine.

A pessimist, guarding against loops being created by ports being Disabled by management action and being immediately enabled again, could equally well maintain that the synced, sync, and reRoot variables should be set to ensure that the Port become Discarding initially to clear these conditions. In addition fdWhile should be initialized in this state since BLOCK_PORT may not be entered once selectedRole is calculate to be DesignatedPort.

With these changes PR:INIT_PORT D7 should read:

```
role = DisabledPort;
synced = FALSE; sync = TRUE;
reRoot = TRUE;
rrWhile = FwdDelay; rbWhile = 0;
fdWhile = FwdDelay
```

[30] With the exception of clearing **proposed** on the Root Port when the bridge has been synchronization and an Agreement has been returned (or is about to be returned).

PR:BLOCK_PORT

This state removes a port that was previously a Root Port or a Designated Port from the active topology.

In D7, this state resets both **agreed** and **proposed**. While it is true that these cannot be true when a port is blocked, clearing these variables here can never be a complete solution, as they are tied to spanning tree priority vectors that may change even though a port's role does not change. I suggest that clearing **synced**, **agreed**, and **proposed** be left to the Port Information state machine (PI), and removed from the Port Role Transition state machine. This approach does mean that changing the root path priority vector (the spanning tree priority information for the bridge derived from the root port) does require **synced**, **agreed**, **proposing**, and **proposed** to be cleared for all ports, however this is accomplished by PI:UPDATE for ports that become Designated and by PI:BETTER for Alternate, Backup, and Root Ports. There is a possibility that **proposed** remains set for an Alternate Port, but that is no bad thing, since that will record the last BPDU sent by the Designated Bridge for the attached LAN, and if the port transitions to a Root Port, the bridge will immediately act on the recent proposal.

PR:BLOCK_PORT D7 reads:

```
role = selectedRole;
agreed = proposing = FALSE;
fdWhile = FwdDelay;[31]
learn = forward = FALSE;
```

PR:BLOCK_PORT should read:

```
role = selectedRole;
learn = forward = FALSE;
```

[31] This is superfluous since fdWhile = FwdDelay is the first action in the BLOCKED_PORT state.

PR:BLOCKED_PORT

This state maintains an Alternate, Backup, or Root Port in the Discarding state once it has transitioned to Discarding. The **synced** flag is set – the port necessarily being synced with the spanning priority vector once it is Discarding, and the **sync** command is cleared. The port is neither **proposing** to its neighbour that it should become Forwarding, nor has it's neighbour **agreed**. However, as in the discussion of PR:BLOCK_PORT above, clearing these variables is better left to PI:BETTER and PI:UPDATE (here they are redundant).

PR:BLOCKED_PORT D7 reads:

```
fdWhile = FwdDelay;
synced = TRUE; rrWhile = 0;
sync = proposed = agreed = reRoot =
FALSE;
```

PR:BLOCKED_PORT should read:

```
fdWhile = FwdDelay;
synced = TRUE; rrWhile = 0;
sync = reRoot = FALSE;
```

The transition that causes PR:BLOCKED port to be re-entered in D7 reads:

```
(fdWhile != FwdDelay) ||
proposed || sync ||
agreed || reRoot || !synced
```

should read:

```
(fdWhile != FwdDelay) ||
sync || reRoot || !synced
```

i.e. proposed and agreed are not required.

PR:BACKUP_PORT

This state is unrelated to the synchronization mechanisms. Its transitions and actions should remain as in D7.

PR:ROOT_PORT

This state maintains a Root Port in its steady state, all other Root Port states dealing with transition conditions or processing. Although proposing and agreed should never be set in this state, that can be accomplished, as suggested above and as partially implemented in D7, by the Port Information state machine.

PR:ROOT_PORT D7 reads:

```
role = RootPort;
proposing = agreed = FALSE;
rrWhile = FwdDelay;
synced = allSynced[32];
```

PR:ROOT_PORT should read:

```
role = RootPort;
rrWhile = FwdDelay;
```

with the synced = allSynced assignment being moved to PR: ROOT_AGREED.

The transition condition from PR:ROOT_PORT to itself reads (in D7):

```
(agreed || proposing || (rrWhile !=
FwdDelay)) && !proposed
```

This condition should read:

```
(rrWhile != FwdDelay)
```

The inclusion of !proposed in the original looks strange. This was intended (I believe) purely to disambiguate which event should fire amongst competing possibilities. In D7 it is neither necessary or sufficient and the completed state diagram needs checking again to fully disambiguate competing transitions.

[32] agree = allAgreed in the original, which is confusing because in the original allAgreed means that **agree** is TRUE for all ports, not that **agreed** is TRUE for all ports.

PR:AGREE_BRIDGE (PR:ROOT_PROPOSED)

This state processes a received Proposal and initiates the synchronization process for the bridge. Setting the **sync** command variable for this port, the Root Port, records the lack of synchronization of the Bridge. It will not be reset until **synced** is TRUE for all the other bridge ports. Since the **sync** variable is also set for these ports they will have received the instruction to get synced.

This state does not reset the **synced** variable either for the Root Port or for the other ports. If the spanning tree priority vector for any of these ports has changed since synced was last set, the port information machine will reset these variables (in PI:BETTER or PI:UPDATE).

A better name for PR:AGREE_BRIDGE would be PR:ROOT_PROPOSED, and the procedure setAgreeBridge[33] should be called setSyncBridge. With these changes PR:ROOT_PROPOSED in D7 reads:

```
setSyncBridge();
```

It requires the addition of:

```
proposed = FALSE;
```

as explained below.

The transition condition into PR:ROOT_PROPOSED in D7 is:

```
proposed && !sync && !synced
```

This condition is intended to have the effect that:

a) if synchronization is already underway, it will not be restarted

b) if a prior Proposal carrying the same spanning tree priority vector has already resulted in synchronization, the full procedure including transitioning ports to Discarding will not be repeated.

The logic for b) is fine, and indeed if **sync** is reasserted for ports for which **synced** is already TRUE, there having been no change in the spanning tree priority vector for the port, there will be no need for any further Discarding or transmission of Proposals by that port. This is fortunate since the logic for a) is flawed, since sync could still be TRUE at the Root Port while it has been cleared at another port, but that other port cleared it while syncing on a prior spanning tree vector. In that case synced would have been set FALSE but the suppression of the further sync to the port would not have given it a chance to become synced, and the bridge as a whole would not synchronize and return an Agreement.

The transition condition into PR:ROOT_PROPOSED should read:

```
proposed && !synced
```

while clearing proposed at an early stage (see above) suppresses repeated entry to the state.

[33] In D7 setAgreeBridge sets agreePort not agree for all ports, while setSyncBridge will set sync for all ports.

PR:ROOT_AGREED

This state returns one, and only one[34], Agreement to each Proposal once the bridge is synchronized. It will also transmit an unsolicited Agreement if the bridge synchronizes before a Proposal is received. It clears the **proposed** and **sync** variables, and sets **newInfo** to request the transmission of a BPDU.

PR:ROOT_AGREED in D7 reads:

```
proposed = sync = FALSE;
newInfo = TRUE;
```

but should read:

```
proposed = sync = FALSE;
synced = allSynced;
newInfo = TRUE;
```

with the synced = allSynced assignment having been moved here from PR:ROOT_PORT (see above). This is necessary so that synced is set correctly before the transmit machine can read newInfo.

The transition condition into PR:ROOT_AGREED in D7 is:

```
(proposed || !synced) &&
(allSynced || synced)
```

which is possibly more easily understood as its logical equivalent:

```
(proposed && allSynced) ||
(proposed && synced) ||
(!synced && allSynced)
```

It could be claimed that the second of these is redundant since synced should never be TRUE for a ROOT_PORT unless allSynced is TRUE. With this revision the transition condition into PR:ROOT_AGREED should read:

```
(proposed && allSynced)||
(!synced && allSynced)
```

PR:REROOT, PR:REROOTED, PR:ROOT_LEARN, PR:ROOT_FORWARD

These states are unrelated to the synchronization mechanisms. Their transitions and actions should remain as in D7, with the exception of the inclusion of !proposed in some of the transitions. This was intended (I believe) purely to disambiguate which event should fire amongst competing possibilities. In D7 it is neither necessary or sufficient and the completed state diagram needs checking again to fully disambiguate competing transitions.

[34] If regular HelloTime BPDUs were sent on the Root Port as well as Designated Ports, which I think is a very good idea on point-to-point links, then this statement should be read as limiting the number of Agreements generated asynchronously to that repeated transmission process.

PR:DESIGNATED_PORT

This state maintains a Designated Port in its current role, all other Designated Port states deal with transition conditions and their processing. Although **proposed** should never be set in this state, that can be accomplished, as suggested above, by the Port Information state machine.

PR:DESIGNATED_PORT D7 reads:

```
role = DesignatedPort;
proposed = FALSE;
```

PR:DESIGNATED_PORT should read:

```
role = DesignatedPort;
```

PR:REQUEST_AGREEMENT (PR:DESIGNATED_PROPOSE)

This state sends a Proposal to a neighbouring bridge if the Designated Port is not Forwarding, has not yet made a Proposal, and has not already received an Agreement. In D7 it is called REQUEST_AGREEMENT; a better name would be DESIGNATED_PROPOSE. It reads:

```
proposing = TRUE;
newInfo = TRUE;
```

and should remain unchanged.

The transition condition into PR: DESIGNATED_PROPOSE in D7 is:

```
(!forward) && (!synced && !proposing)
```

This is incorrect and probably the result of some variable name confusion (in D7 the variable referred to in this note as **synced** is called 'agree'). Unfortunately **synced** can become TRUE just because the port is Discarding, which will leave some Designated Ports transitioning to Forwarding only by way of Forward Delay timer expiry. A further correction is required to stop an operEdge port sending a Proposal as it transitions from Disabled.

The transition condition into PR: DESIGNATED_PROPOSE should be:

```
(!forward) &&
(!agreed && !proposing && !operEdge)
```

PR:DESIGNATED_AGREED (PR:DESIGNATED_ SYNCED)

This state is inaccurately named PR:DESIGNATED_AGREED in D7, since it can be entered when 'agreed' is not true and no Agreement has been received. It should PR:DESIGNATED_SYNCED. This reflects its purpose which is to set **synced** TRUE and **sync** FALSE, to indicate the its neighbour and/or the port has been synchronized with the spanning tree priority vector. This has happened if the port state has transitioned to Discarding, if an Agreement has been received, if the port is an **operEdge,** or if the port has already been **synced** but **sync** has been reasserted. In D7 the state actions read:

```
synced = TRUE; rrWhile = 0;
sync = proposing = FALSE;
```

They should read:

```
rrWhile = 0;
synced = TRUE;
sync = FALSE;
```

since setting **proposing** FALSE is a mistake, possibly related to the confusion between **synced** and **agreed** in the transition into PR:DESIGNATED_PROPOSE (see above). The Port Information machine will reset **proposing** before any further Proposal needs to be sent, as mentioned several times previously in this note.

The transition condition into PR: DESIGNATED_SYNCED in D7 is:

```
(!learning && !forwarding && !synced) ||
(agreed && !synced) ||
(operEdge && !synced) ||
(proposing && synced) ||
(sync && synced)
```

all of which are correct apart from the, fourth line. This condition should read:

```
(!learning && !forwarding && !synced) ||
(agreed && !synced) ||
(operEdge && !synced) ||
(sync && synced)
```

PR:RETIRED_ROOT (PR:DESIGNATED_RETIRED)

This state resets the reRoot command to the port once rrWhile has reached zero, either through time expiry or because the port has been synced (see PR:DESIGNATED_SYNCED). In D7 the state actions are:

```
reRoot = FALSE;
```

and should remain unchanged.

The transition condition into PR: DESIGNATED_LISTEN in D7 is:

```
(rrWhile == 0) && reRoot
```

and should remain unchanged.

PR:DESIGNATED_LISTEN

This state transitions a Designated Port to the Discarding state if the port is not an operEdge and either a sync command has been received and the port is not already synced or the port is a recent root and the bridge is being re-rooted. In D7 the state actions are:

```
learn = forward = FALSE;
fdWhile = FwdDelay;
```

and should remain unchanged.

The transition condition into PR: DESIGNATED_LISTEN in D7 is:

```
(((sync && !synced)||
 (reRoot && (rrWhile !=0)
 ) && !operEdge
) && (learn || forward)
```

This logic should remain unchanged, but could be more simply written:

```
(   (sync   && !synced)
 || (reRoot && (rrWhile !=0))
)
&& !operEdge
&& (learn || forward)
```

PR:DESIGNATED_LEARN

This state transitions a Designated Port to the Learning state if the port is Discarding and:

a) the port is an operEdge; or

b) the port is either not a recent root, or the bridge is not being re-rooted; and either:

the fdWhile timer has expired; or

an Agreement has been received.

In D7 the state actions are:

```
learn = TRUE;
fdWhile = FwdDelay;
```

and should remain unchanged.

The transition condition into PR: DESIGNATED_LEARN in D7 is:

```
((((rrWhile == 0) || !reRoot) &&
((fdWhile == 0) ||(agreed &&
forceVersion >= 2))) &&
!sync) || operEdge) && !learn
```

This is unnecessarily complicated by the inclusion of (forceVersion >= 2). This check would be better done when processing the BPDU to see if **agreed** should be set at all. Removing this, and supplying a missing bracket, D7 says:

```
((   ((rrWhile == 0) || !reRoot)
  && ((fdWhile == 0) || agreed))
  &&  !sync
 )
 || operEdge
)
&& !learn
```

I think this should be rather more simply put:

```
  ((fdWhile == 0) || agreed || operEdge)
&&((rrWhile == 0) || !reRoot)
&& !sync
&& !learn
```

PR:DESIGNATED_FORWARD

This state transitions a Designated Port to the Forwarding state if the port is Learning and the other conditions are as for PR:DESIGNATED_LEARN.

c) the port is an operEdge; or

d) the port is either not a recent root, or the bridge is not being re-rooted; and either:

the fdWhile timer has expired; or

an Agreement has been received.

In D7 the state actions are:

```
forward = TRUE;
fdWhile = FwdDelay;
```

and should remain unchanged.

Similar comments as those on PR: DESIGNATED_LEARN apply to the transition conditions. These should read:

```
  ((fdWhile == 0) || agreed || operEdge)
&&((rrWhile == 0) || !reRoot)
&& !sync
&& (learn && !forward)
```