MAC Status Propagation

Mick Seaman

MAC Relays (such as the P802.1aj TPMR) that provide a frame forwarding (sub)layer below and transparent to other bridges and their protocols can significantly degrade service availability. If the relaying sublayer were not present, changes in connectivity would be accompanied by a change in the MAC_Operational status parameter. MAC_Operational provides rapid notification of connectivity failures and prompts the necessary initial protocol behavior to ensure that new connectivity has not caused an instantaneous data loop. If the MAC status is not propagated by the relays, bridge protocols have to rely on periodic transmissions to detect connectivity changes. These take time to cut loops and repair failures caused by changes in the relayed links.

This note describes media independent propagation of MAC status, while allowing connectivity to a relay while one of its links is not operational.

Since the first revision of this note (August 20th, 2006) the ideas described have been incorporated in P802.1aj, with few text changes. This revision is referenced by my ballot comments on P802.1aj D2.2. It aims to improve upon changes made to the original state machines in a prior P802.1aj ballot, and to separate out ideas to be incorporated in P802.1aj from those that may (in time) be useful elsewhere.

1. Organization of this note

The first revision of this note was a discussion document, aimed more at describing and discovering protocol alternatives, than at providing text for ready incorporation in P802.1aj. This revision reverses that perspective, concentrating on providing text and diagrams that can be lifted more or less as is for use in the standard.

Section 2 of this note is suggested as complete replacement text for P802.1aj Clause 23. It is based on text from P802.1aj D2.2 that was in turn a modification of the first revision of this note, with the intent that the ballot comments that led to that modification do not have to be resubmitted by others. While providing replacement text is not the preferred way of making ballot comments, the suggested changes are so extensive that it is necessary to view them as a whole to see whether they work or not.

The main rationale for these changes is that introductory material in a standard should be directed at explaining what is in that standard, only alluding to other possibilities where they might be part of the future evolution of the standard. Since MAC status propagation is now a mandatory part of TPMR operation (which the first revision of this note could not assume) and one specific signaling method has been chosen, a thorough revision was required.

The suggested text in Section 2. has borrowed heavily from experience with other drafts, including those for 802.1D, for amendments to 802.1Q, for parts of 802.1AB, for 802.1AE and for P802.1af. Some of the more significant points in the suggested text are discussed in Section 3.

Alternatives and possible future work are mentionned in Section 4, though I have not seen a case for that work.

This note has been prepared in a fairly short time. While I have endeavoured to proof read it, it is likely that errors remain. I would be grateful for any comments, though am unlikely to be able to respond before the middle of January.

2. MAC status propagation

Individual LANs, each operating its own MAC and media access method specific procedures, can be connected by one or more TPMRs to form a composite LAN that is transparent to other bridges and stations and their configuration protocols. The MAC status protocol (MSP) ensures that changes in the connectivity provided by the composite LAN results in changes in the MAC_Operational status parameter (802.1Q clause 6.6.2) at each of the attached bridges and stations, just as if they were connected to an individual LAN.

MAC_Operational provides rapid notification of connectivity failures and prompts the necessary initial protocol behavior to ensure that new connectivity has not caused an instantaneous data loop. If this MAC status parameter were not propagated by a TPMR, bridge protocols would have to rely on periodic transmissions to detect connectivity changes. On their own these periodic transmissions take longer to detect failures, and cannot detect the possibility of data loops until they have been created, as illustrated by the following examples.

Figure 1 shows a TPMR connecting two Bridge Ports.

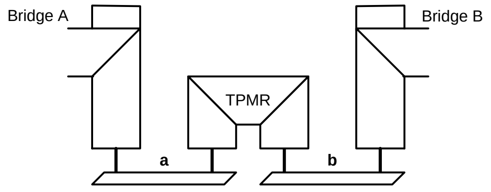


Figure 1—TPMR connecting two Bridge Ports

Assume that Bridge A is the spanning tree Designated Bridge for the LAN that comprises the individual LANs **a** and **b** and the TPMR, and that Bridge B's spanning tree Root Port is shown. If **a** fails and the TPMR does not propagate MAC_Operational, Bridge B will not reselect its Root Port until it has timed out the last BPDU from A.

Figure 2 shows a link that uses two TPMRs, perhaps because LAN **c** uses a non-802 technology together with an appropriate convergence function.

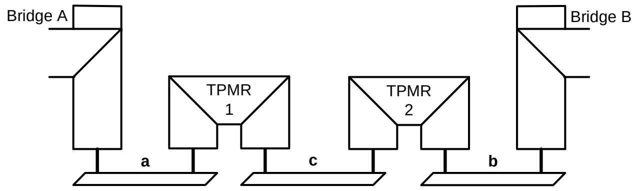


Figure 2—TPMR chain connecting Bridge Ports

Without MAC_Operational propagation, failure of **c** will not be immediately visible to either bridge. Worse, if **c** fails and is restored after a while, both A and B will believe themselves to be Designated Bridge for the composite LAN, and will forward frames until one receives a BPDU from the other, even if a data loop has been created.

When working correctly the MAC Service provides bi-directional connectivity or no connectivity at all. MAC_Operational is TRUE for each of two peers connected to the same LAN if they can communicate, and FALSE for either or both if they cannot. Protocols, such as the spanning tree protocols, that make use of MAC_Operational to detect new connectivity and initialize state machines rely on the last peer that sees MAC_Operational transition TRUE to enforce any necessary behavior after a connectivity change. For example, it is the last of two connected Bridge Ports to be powered on that enforces the necessary delay prior to setting operEdge TRUE (13.24).

NOTE—MAC_Operational being TRUE within a single station does not guarantee connectivity to any peer. Even if connected by a point-to-point LAN, the peer could have just reinitialized. It is clearly not possible, given only the use of the LAN medium for communication, to arrange for two peers to transition MAC_Operational at exactly the same time.

This clause defines media independent propagation of MAC status between TPMRs and to attached stations, as follows. It:

a) Models MAC status propagation within the bridge architecture used to describe a TPMR (2.1)

b) Provides an overview of the MAC Status Protocol (MSP, 2.2)

c) Specifies state machines for MSP operation (2.3–2.9)

d) Specifies the addressing, protocol identification, format, encoding, and validation of MAC Status Protocol Data Units (MSPDUs, 2.13–2.16)

The term *MAC status propagation* (IEEE Std 802.1Q 3.5) describes the overall process of communicating a MAC_Operational parameter value through one or more TPMRs. MSP can use *link status notification*†1 (IEEE Std 802.1x 3.3) between some relays and end stations, and *MAC status notification* (IEEE Std 802.1x 3.4) between others. These two notification methods differ, as follows:

a) Link status notification uses frames to convey information about MAC_Operational. It does not interrupt or prevent other communication between adjacent relays, or between a relay and a bridge. However, it requires both the source and destination of the notification to implement additional protocol. Since it does not prevent communication it cannot, by itself, prevent loops caused by new connectivity.

b) MAC status notification uses a layer management interaction with the local MAC Entity to change MAC_Operational for a peer user of the MAC service provided by an individual LAN. It is equivalent to bringing a link down, and is generally effective for MACs with specific point-to-point support. It also interrupts all other communication, including the use of in-band management to rectify an underlying problem.

Where management of a TPMR is permitted through one of its ports, the failure of an individual LAN not in the communication path does not cause MAC status propagation to prevent management connectivity. In Figure 2, for example, the failure of LAN **b** does not prevent connectivity to TPMR 2 from Bridge A. When **b** recovers, MAC_Operational for Bridge A's Port is 'blipped', i.e. taken FALSE for a brief period and allowed to return TRUE, thus meeting the requirement for transition when connectivity changes.While this slows the recognition of newly available connectivity, that is rarely an issue since several seconds hysteresis should be applied to any MAC_Operational status transition to prevent higher layer protocols 'flapping'.

†1The use of the term link status notification in this standard is not to be confused with the term used in SNMP to refer to a type of trap notification.

2.1 Model of operation

The model of operation presented in this clause (2) allows the description of MSP functionality to be consistent with that of the general bridge architecture and operation (IEEE Std 802.1Q clauses 8.2, 8.3). Real implementations of a TPMR may adopt any internal model of operation compatible with the externally visible behavior specified.

The additional entities that model MSP operation, and their relationship to the other processes and entities that model the operation of a bridge are illustrated in Figure 3. They comprise the following:

a) A MAC Status Shim (MSS) for each Port, that allows MSP to control the value of MAC_Operational presented to the Bridge Port connectivity function (IEEE Std 80.21Q clause 8.5.1) and hence to the frame transmission and reception functions of the MAC Relay Entity, Higher Layer Entities, and MSPE.

b) The MAC Status Propagation Entity (MSPE), that implements MSP, controlling each MSS to ensure that frames are not forwarded by the MAC Relay Entity until status propagation is complete, transmitting and receiving frames to support link status notification, and controlling the individual LAN's MAC to provide MAC status notification.

NOTE—In a TPMR the VLAN tagging functions shown in Figure 3 are limited to extracting priority from received tagged frames for use with traffic class queuing. Tags are not added, removed, or changed.

The MSPE uses the LMI to control and receive status from each MSS and individual LAN MAC, allowing generic management requests and indications to be tailored to the requirements of different MACs, as well as providing a way for one MSS to propagate status to another (via the MSPE), even though MAC_Operational for the MSS' upper ISS service access point is FALSE.

Link status notification frames are sent to a standard group address. They are received by the MSPE, but are forwarded by the MAC Relay Entity (like any other frame) in order to communicate without a delay in each TPMR. The MSPE attaches to each Port by a Bridge Port connectivity function (see IEEE Std 802.1Q clause 8.5.1) that allows it to transmit to and receive from the attached individual LAN only, as shown in Figure 3.

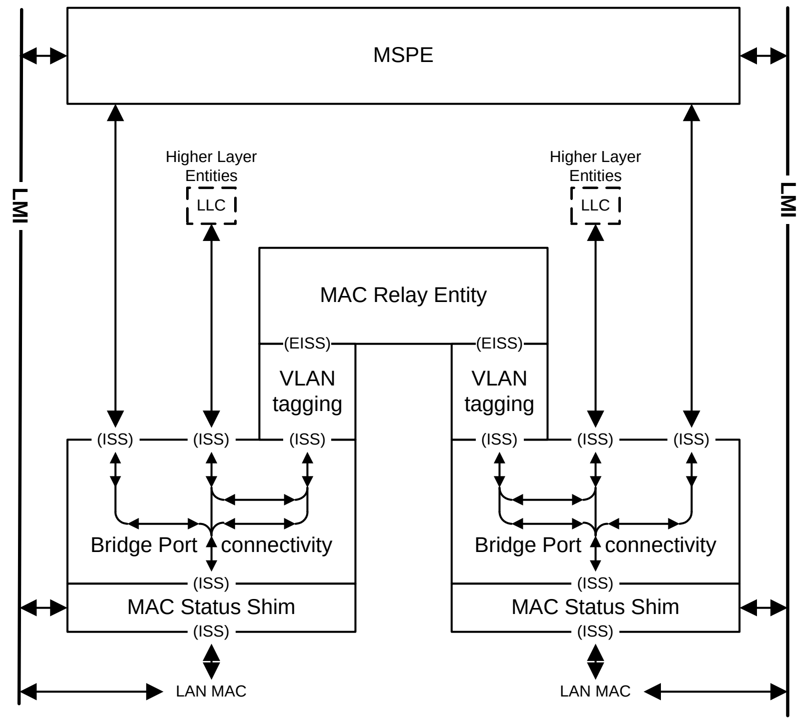


Figure 3—MAC Status Shims and the MAC Status Propagation Entity

2.2 MAC status protocol (MSP) overview

This clause (P802.1aj clause 23.2) provides examples of MSP operation as time sequence diagrams that show the following:

a) Exchange of MSPDUs (MAC Status Protocol Data Units) transmitted and received to support link status notification.

b) The values of MAC_Operational and MAC_Enabled provided by:

   1) each MAC Status Shim MSS, at its upper ISS service access point, to the Bridge Port connectivity function.

   2) each LAN MAC to the MSS, at the latter's lower ISS service access point.

The value of MAC_Operational provided by the MSS is TRUE if, and only if the MSS' MAC_Enabled is TRUE and the value of MAC_Operational provided to the MSS by the LAN MAC is TRUE. The latter can be TRUE only if MAC_Enabled is TRUE for both the LAN MAC and its peer in the other station connected to the LAN. The MSPE can use the LMI to disable the MSS in order to prevent communication until MAC status propagation has occurred, and to disable the LAN MAC in order to provide MAC status notification to a peer service user. This clause uses the symbols defined in Table 1 to show combinations of the MAC status parameter values for each Port.

Throughout this protocol description, the term *LAN* is used in accordance with its identified meaning in this standard, i.e. to refer to individual LANs at the lowest layer in the architecture shown, connecting adjacent TPMRs or a TPMR and an adjacent bridge. A *chain* is a series or part of a series of TPMRs connected by LANs, and a *link* is the connectivity provided by a chain between non-TPMR devices that communicate at a higher (sub) layer and perceive the entire link as a single transparent LAN.

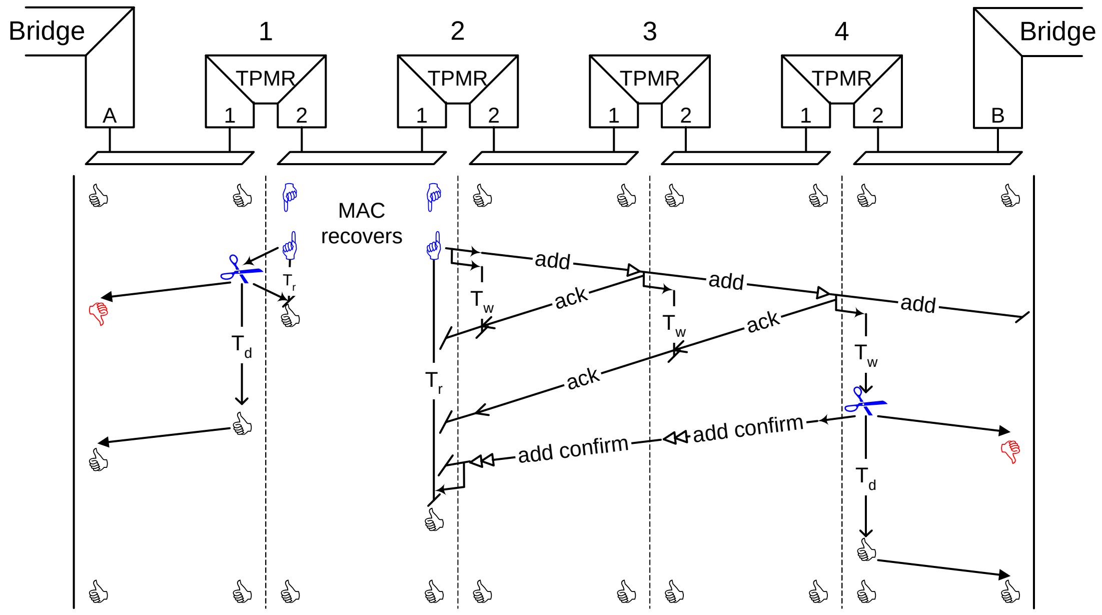
Table 1—Time sequence diagram symbols

		👍	👎	👉	☝	✂
MSS	**MAC_Operational**	T	F	F	F	F
	MAC_Enabled	T	T	F	F	—
LAN MAC	**MAC_Operational**	T	F	F	T	F
	MAC_Enabled	T	T	T	T	F

T = TRUE, F = FALSE, — = either TRUE or FALSE

The number of TPMRs in the following examples is deliberately high, four where one or two might be more common, in order to show all the necessary aspects of protocol behavior.

Signaling the addition of new or restored connectivity is considered first, as it is more difficult than signalling failure—the wide range of options for the latter provide little guidance for protocol design. Figure 4 shows the response to a new or recovered LAN connection in a link between two bridges that do not implement MSP—and thus require MAC status notification.



Timers: Tr : linkNotifyWhen ('retry timer') Tw : macNotifyWhen ('wait timer') Td :macNotifyWhile ('down timer')

Figure 4—Adding connectivity

Before the LAN that connects TPMR 1 to TPMR 2 is MAC_Operational, each MSPE disables its MSS (👉). This allows the MSPE to intercept the new connectivity as the LAN MAC asserts MAC_Operational (☝). TPMR 2 propagates the new status to its other port, transmitting an *add* MSPDU. TPMR 2 does not necessarily know that TPMR 3 implements MSP (or indeed is a TPMR) but uses the default MSP configuration—waiting before resorting to MAC status notification and starting a linkNotifyWhile timer (as do TPMRs 3 and 4) on receipt of the *add* (which is forwarded by the Relay Entity). Each TPMR responds to the *add* with an *ack* that clears the timer, but the bridge at the end of the chain does not (as it does not implement MSP). When TPMR 4's timer expires, its MSPE disables the LAN MAC (✂), ensuring that MAC_Operational becomes FALSE (👎) for the bridge's port, and starts a macNotifyWhile timer. When that timer expires, the MSPE re-enables the LAN MAC so that the bridge port's

MAC_Operational can become TRUE (👍) once more. As well as disabling the LAN MAC for its port 2, TPMR 4 also transmitted an *add confirm* through port 1, and receipt of this *add confirm* by TPMR 2 causes the latter to cancel its retry timer (linkNotifyWhen). The initial value of macNotifyWhile (MacNotifyTime) is sufficient for the *add confirm* to reach the TPMR 2, and for that TPMR's MSPE to enable the MSS, thus ensuring that there is connectivity between the bridges connected by the link when they have both reported MAC_Operational TRUE to their local protocol clients.

In Figure 4 the MSP configuration of TPMR 2's port 2 differs from that for port 1 of TPMR 1. The latter is explicitly configured to use MAC status notification immediately without waiting to see if its peer implements MSP.

Figure 5 illustrates the operation of MSP when the connectivity provided by an individual LAN is lost. The MSPE for each of the TPMR ports directly attached to the failed LAN begins by disabling the MSS for that port (☞) to ensure that connectivity does not 'flap'. Otherwise the protocol proceeds as for connectivity addition, except that *loss* and *loss confirm* are used instead of *add* and *add confirm*, and the MSS' attached to the failed LAN are left disabled. The final state of the link components is the initial state assumed in Figure 4 — both attached bridges have seen MAC_Operational transition to indicate that the connectivity has changed, and the TPMRs in each of the two chains can be reached and managed (if their individual configuration permits) through the attached bridge ports.

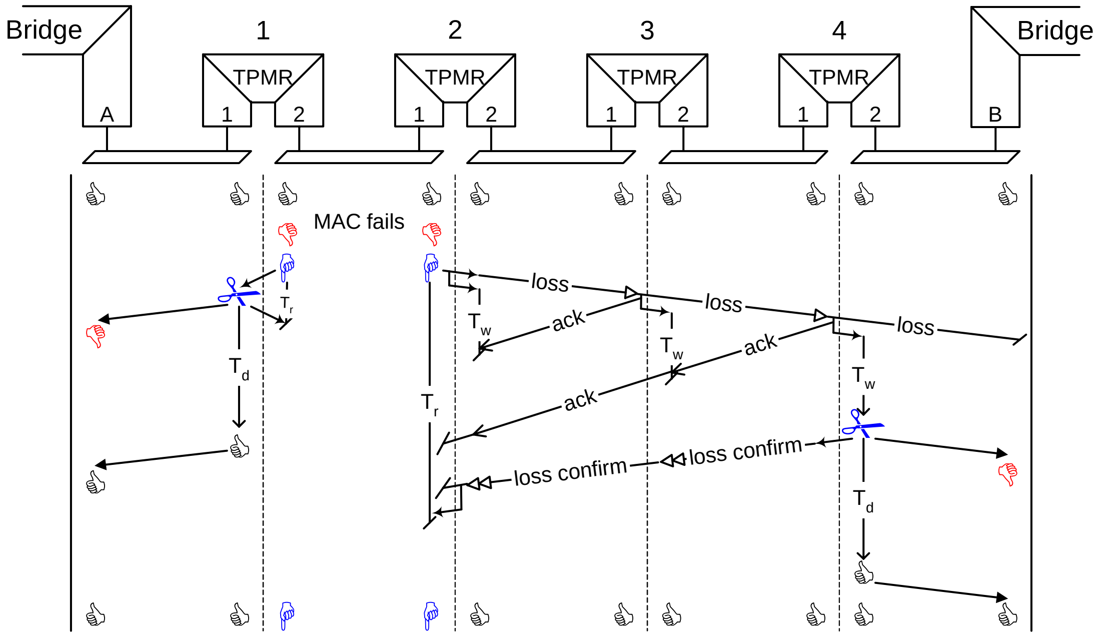


Figure 5—Losing connectivity

In the examples above, if an *add*, or *loss* is lost then the TPMR port that transmitted or relayed that MSPDU will revert to using MAC status notification, as will a TPMR that fails to receive the *ack* from the next (or a subsequent) TPMR in the chain. If the MSPDU is lost on a LAN removed from the TPMR that initiated the link status notification, then that notification will be retried on expiry of the linkNotifyWhen timer which also serves to protect against the loss of an *add confirm* or *loss confirm*.

Loss of a frame due to physical corruption is rare in LAN technologies, and frame losses due to buffer overrun are not expected when connectivity is being added (as the link is not usable prior to the addition) nor when a loss of connectivity is being signaled (as that connectivity loss will have prevented other frames from being added to the link). The loss of an MSPDU is most likely to be due to failure of one of the LANs, or to interruption of a TPMR's relay functionality by another MSS whose MSPE is also waiting for confirmation that a connectivity change that it has detected has been propagated to the end of the link. MSP does not, and cannot, ensure that information about each and every connectivity change reaches both ends of the link. Unless link status notification or MAC status notification is disabled or the individual LANs fail to report MAC_Operational correctly, MSP does ensure that any change in connectivity is accompanied by one or more notifications at each end of the link, and that a continuous period during which MAC_Operational is reported TRUE at both ends of the link provides connectivity from 1 second after the start of the period to 1 second before the end. This guarantee meets the requirements implicit in the initial transmission and retransmission strategies of well designed protocols. Protocol clients of the MAC Service should not use the difference between *loss* and *add* MSPDUs to take different actions on receipt, though the distinction can be useful to a network administrator when investigating connectivity changes.

Figure 4 provides a common example of simultaneous change. TPMR 2 is powered on and initializes both its ports. When the LAN MAC becomes operational, each port attempts to send a notification through the other, but cannot as they are both waiting for the connectivity addition to be confirmed. However, in this eventuality, each port can provide the other with the confirmation required, because the peer system for each of the individual LANs will also have seen the initial MAC_Operational transition and will have initiated a notification towards its end of the link.

NOTE—The state machine transition direct from SNM:LINK_NOTIFICATION to SNM:MAC_NOTIFYING supports this behavior (see Figure 11).

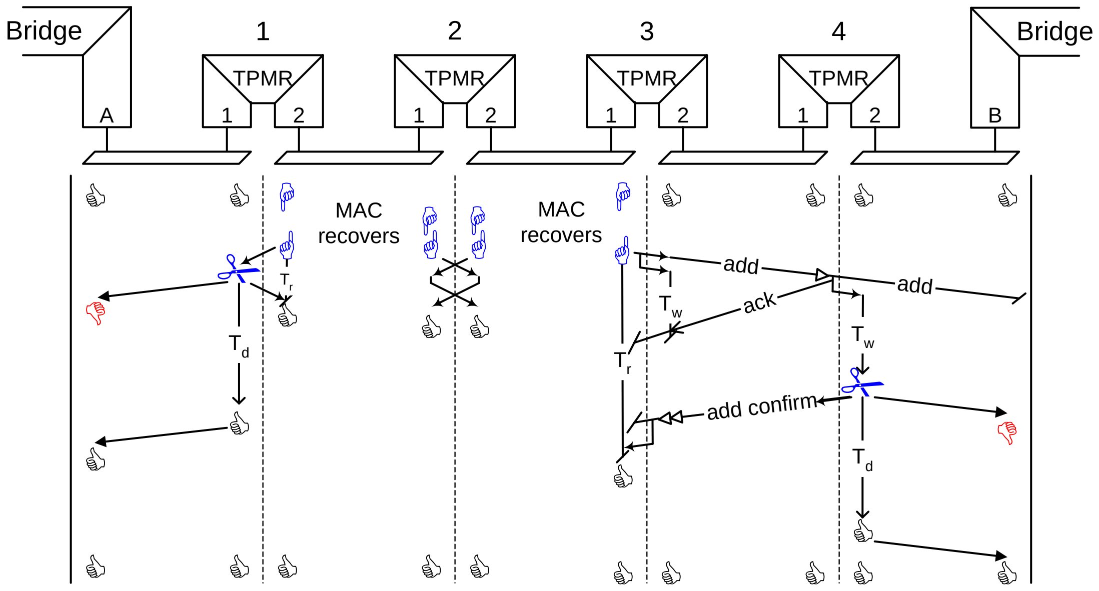


Figure 6—TPMR recovery

If the individual LAN that recovers (or loses) connectivity is at the end of the link, MSP ensures that the other end of the link is also aware of the status change as illustrated in Figure 7. This figures also shows the effect on MSP of including two non-standard relays in the chain. Provided they (and their attached LANs) never fail, MSP can continue to operate as intended.

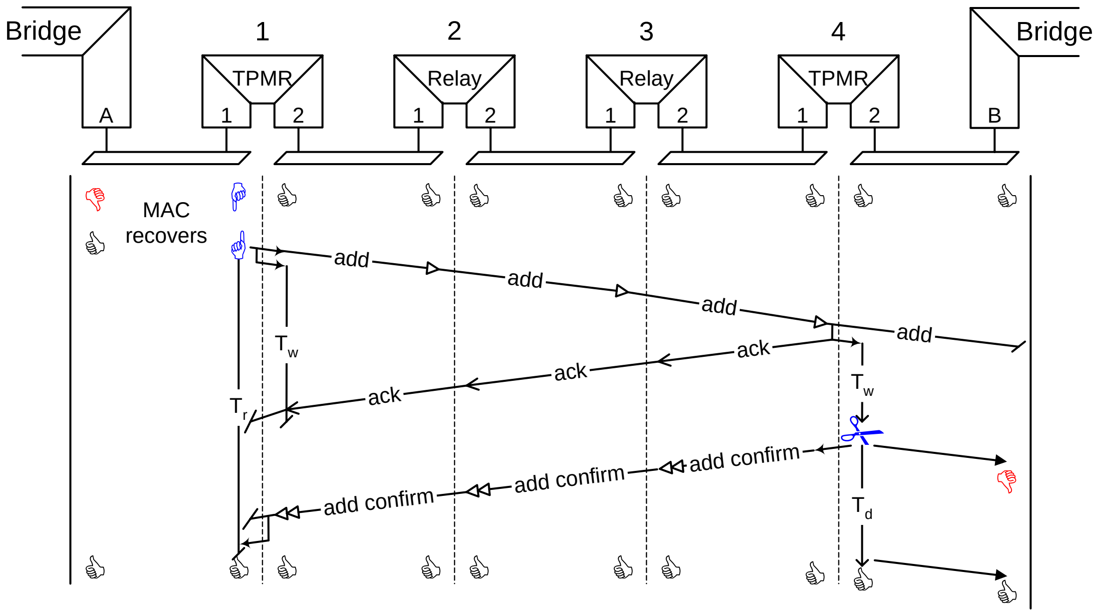


Figure 7—Notification from one end of the link to the other

If TPMR 4 is configured to use MAC status notification immediately, it does not return an *ack* as the *add confirm* can be sent almost immediately (see Figure 8).

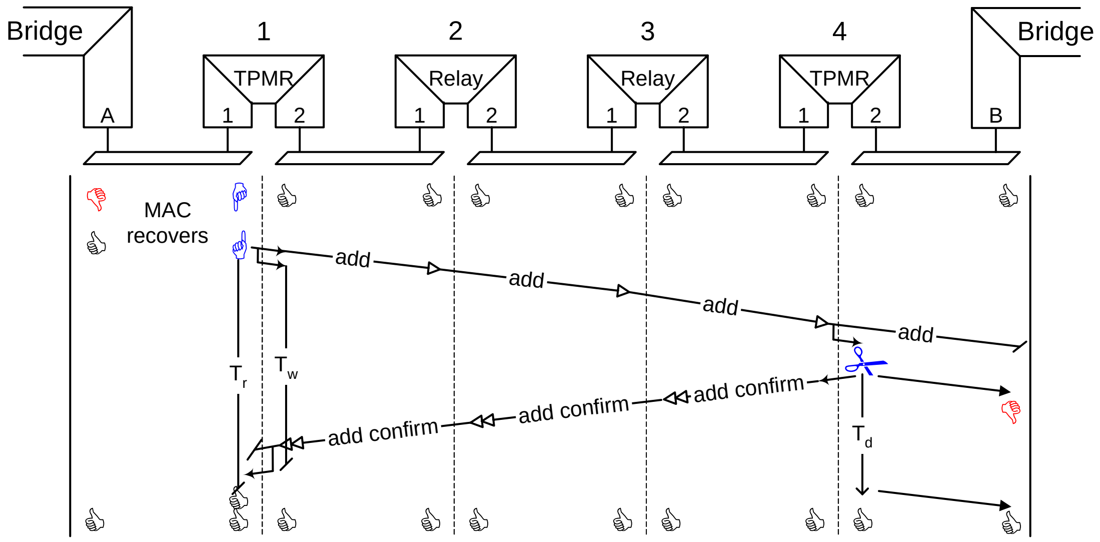


Figure 8—Immediate MAC status notification at the end of a link

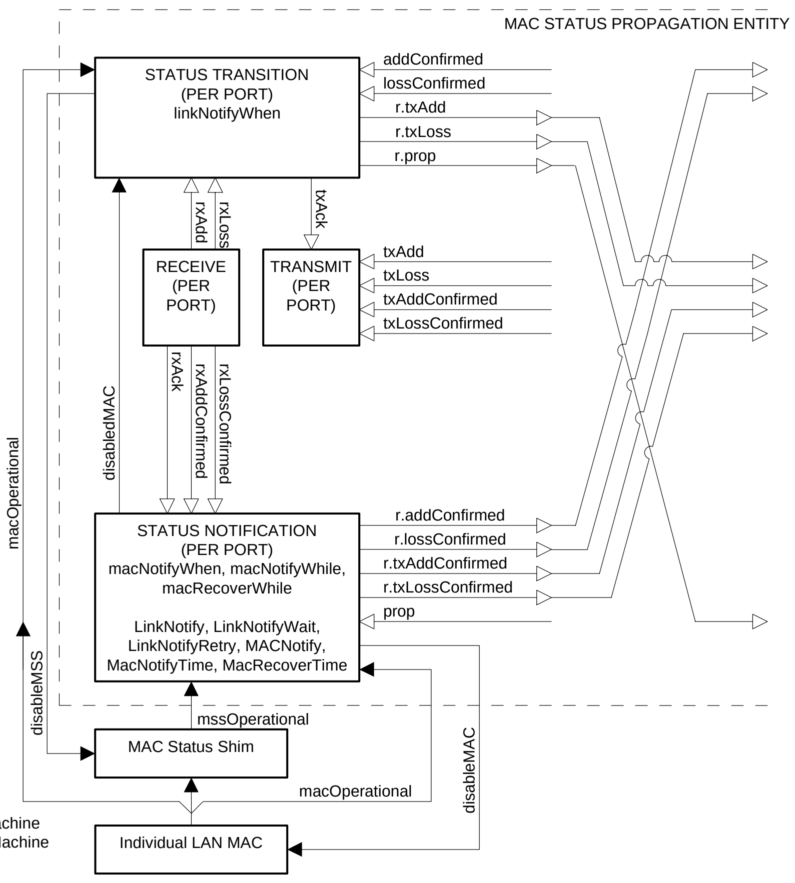
2.3 MAC status protocol state machines

The operation of the MAC Status Propagation Entity (MSPE) is represented by an instance of each of the following for each Port of the TPMR:

a) A Status Transition state machine (STM, 2.8)
b) A Status Notification state machine (SNM, 2.9)
c) A Receive Process (2.10)
d) A Transmit Process (2.11)

Figure 9 shows the state machine variables that are used to communicate between these machines and processes, and that support management control over their operation. Variables prefixed with ‘r.’ are those of the corresponding state machines of the other Port of the TPMR.

The notational conventions used in Figure 9 and in the specification of the state machine are identical to those used in the specification of MSTP (IEEE Std 802.1Q Clause 13) and RSTP and are defined in clause 17.6 of IEEE Std 802.1D.



ABBREVIATIONS:

STM:	Status Transition Machine
SNM:	Status Notification Machine

NOTATION:
A variable is shown within the machine that uses it, or between machines with an arrow head style that indicates how it is used to communicate:
——▶Not changed by the target machine.
——▷Set (or cleared) by the originating machine, cleared (or set) by the target machine.

Figure 9—MSPE machine overview

In Figure 9, the prefix ‘r.’ identifies a variable of the other port’s corresponding state machines. Port 1’s STM monitors MAC_Operational transitions for its own LAN, and tells Port 2’s Transmit process to send *add* or *loss* link notifications and Port 2’s SNM to monitor the progress of status notification, using MAC status notification if necessary. Port 2’s SNM receives the *ack* that indicates that it should wait for link status notification to complete, and the *add confirm* or *add confirm* that indicates that completion. Whether link status or MAC status notification is used, Port 2’s SNM tells Port 1’s STM when the notification has been confirmed, so the latter does not have to retry the notification. Port 1’s SNM provides the same service to Port 2’s STM.

A *loss* or *add* notification that is received from Port 1’s own LAN is handled by its STM, which propagates the notification to Port 2’s SNM (in the same way as a local MAC_Operational transition) while transmitting an *ack* on Port 1’s LAN. Similarly Port 2’s STM transmits an *ack* for a link status notification received on its own LAN, and propagates the notification to Port 1’s SNM.

2.4 State machine timers

Timers are implemented by variables that are decremented on each timer tick, with timer expiry occurring when they reach zero.

2.4.1 linkNotifyWhen

Causes a link status notification to be sent on each expiry until the original status transition is confirmed.

2.4.2 linkNotifyWhile

Started when a change is first propagated through the Port, on expiry allows MAC status notification.

2.4.3 macNotifyWhile

Sets the time for which the MAC is disabled for MAC status propagation.

2.4.4 macRecoverWhile

Sets the time for which the MAC is permitted to be non-operational, after being disabled, before the link is reported as lost.

2.5 MSP performance parameters

These parameters are not modified by the operation of MSP but are treated as constants by the state machines. They may be managed independently for each TPMR Port—default values and permissible ranges are specified in Table 2.

2.5.1 LinkNotify

TRUE if the port uses link status notification to propagate MAC status, and will wait to allow link status notification to succeed before using MAC status notification.

NOTE—If LinkNotify is FALSE, the TPMR may still forward *loss* and *add* notifications transmitted by other TPMRs prior to using MAC status notification, but the TPMR will not originate link status notifications.

2.5.2 LinkNotifyWait

The initial value of the linkNotifyWhile timer.

2.5.3 LinkNotifyRetry

The initial value of the linkNotifyWhen timer.

2.5.4 MACNotify

TRUE if the port uses MAC status notification.

2.5.5 MACNotifyTime

The initial value of the macNotifyWhile timer.

2.5.6 MACRecoverTime

The initial value of the macRecoverWhile timer.

Table 2—MSP performance parameters

Parameter	Recommended or default value	Permitted range
LinkNotify	TRUE	TRUE or FALSE
LinkNotifyWait	0.4 s	0.2 – 1.0 s
LinkNotifyRetry	1.0 s	0.1 – 1.0 s
MACNotify	TRUE	TRUE or FALSE
MACNotifyTime	0.2 s	0.01 – 0.5 s
MACRecoverTime	0.1 s	0.02 – 0.5 s

2.6 State machine variables

2.6.1 BEGIN

This is a Boolean variable controlled by the system initialization process. A value of TRUE causes all TPMR state machines to continuously execute their initial state. A value of FALSE allows all state machines to perform transitions out of their initial state, in accordance with the relevant state machine definitions.

2.6.2 addConfirmed

Set by the other Port's SNM to tell STM that the addition has been confirmed. Cleared by STM.

2.6.3 disableMAC

Set by SNM to instruct the individual LAN MAC (via an LMI) to disable itself in a way that will cause MAC_Operational to be FALSE for the peer user of the MAC Service provided by that LAN.

NOTE— The state machines do not assume that a client of the MAC can tell whether the MAC is not operational because that client has disabled it, or whether some other client has disabled it.

2.6.4 disabledMAC

Set by the SNM when it has set disableMAC and for MACRecoverTime after disableMAC has been reset, so that the STM does not conclude that a loss notification should be sent through the other TPMR port.

2.6.5 lossConfirmed

Similar to addConfirmed but confirms a loss.

2.6.6 macOperational

The value of MAC_Operational for the individual LAN MAC.

2.6.7 mssOperational

The value of MAC_Operational provided by the MSS to the TPMR Port's bridge transmit and receive function.

2.6.8 prop

Set by the other Port's STM to Add or Loss to notify SNM that a change is being propagated through the Port. Reset by SNM to None.

2.6.9 rxAck

Set by the Receive process to tell SNM that an acknowledgment has been received. Cleared by SNM.

2.6.10 rxAdd

Set by the Receive process to tell STM that an *add* notification is being propagated through the relay. Cleared by STM.

2.6.11 rxAddConfirm

Set by the Receive process to tell SNM that an *add confirm* has been received. Cleared by SNM.

2.6.12 rxLoss

Similar to rxAdd, but for a *loss*.

2.6.13 rxLossConfirm

Similar to rxAddConfirm, but for a *loss confirm*.

2.6.14 txAck

Set by the STM to instruct the Transmit process to send an acknowledgment. Cleared by the Transmit process.

2.6.15 txAdd

Set by the other Port's STM to causes transmission of an *add* notification. Cleared by the Transmit process.

2.6.16 txAddConfirm

Set by the other Port's SNM to causes transmission of an *add confirm* (through this Port) confirming that a received *add* message has been acted upon. Cleared by the Transmit process.

2.6.17 txLoss

Similar to a txAdd but causes a *loss* message to be transmitted.

2.6.18 txLossConfirm

Similar to a txAddConfirm but causes a *loss confirm* message to be transmitted.

2.7 State machine procedures

No procedures are defined beyond those represented in the state machines.

2.8 Status Transition state machine

The Status Transition state machine shall implement the function specified by the state diagram in Figure 10 and the attendant definitions contained in 2.4 through 2.7.

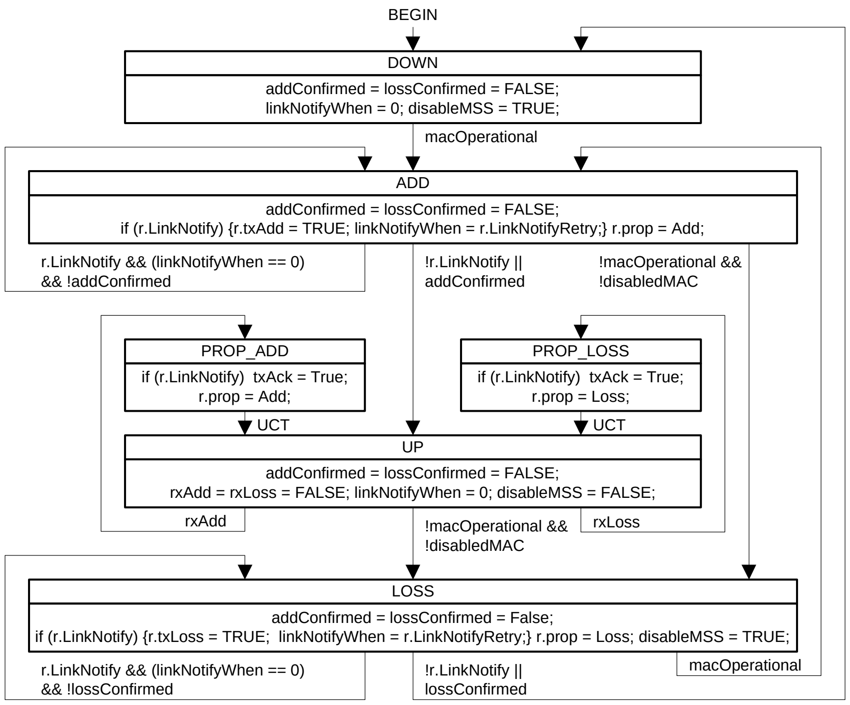

BEGIN
DOWN
addConfirmed = lossConfirmed = FALSE;
linkNotifyWhen = 0; disableMSS = TRUE;
macOperational
ADD
addConfirmed = lossConfirmed = FALSE;
if (r.LinkNotify) {r.txAdd = TRUE; linkNotifyWhen = r.LinkNotifyRetry;} r.prop = Add;
r.LinkNotify && (linkNotifyWhen == 0) && !addConfirmed
!r.LinkNotify || addConfirmed
!macOperational && !disabledMAC
PROP_ADD
if (r.LinkNotify) txAck = True;
r.prop = Add;
PROP_LOSS
if (r.LinkNotify) txAck = True;
r.prop = Loss;
UCT
UCT
UP
addConfirmed = lossConfirmed = FALSE;
rxAdd = rxLoss = FALSE; linkNotifyWhen = 0; disableMSS = FALSE;
rxAdd
!macOperational && !disabledMAC
rxLoss
LOSS
addConfirmed = lossConfirmed = False;
if (r.LinkNotify) {r.txLoss = TRUE; linkNotifyWhen = r.LinkNotifyRetry;} r.prop = Loss; disableMSS = TRUE;
r.LinkNotify && (linkNotifyWhen == 0) && !lossConfirmed
!r.LinkNotify || lossConfirmed
macOperational

Figure 10—Status Transition state machine

2.9 Status Notification state machine

The Status Notification state machine shall implement the function specified by the state diagram in Figure 11 and the attendant definitions contained in 2.4 through 2.7.

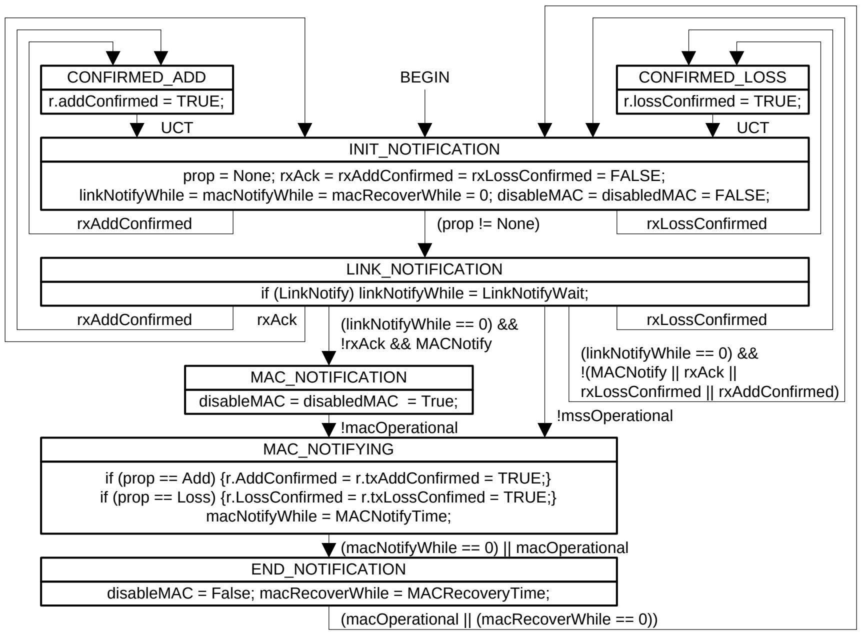

CONFIRMED_ADD
r.addConfirmed = TRUE;
UCT
BEGIN
CONFIRMED_LOSS
r.lossConfirmed = TRUE;
UCT
INIT_NOTIFICATION
prop = None; rxAck = rxAddConfirmed = rxLossConfirmed = FALSE;
linkNotifyWhile = macNotifyWhile = macRecoverWhile = 0; disableMAC = disabledMAC = FALSE;
rxAddConfirmed
(prop != None)
rxLossConfirmed
LINK_NOTIFICATION
if (LinkNotify) linkNotifyWhile = LinkNotifyWait;
rxAddConfirmed
rxAck
(linkNotifyWhile == 0) && !rxAck && MACNotify
rxLossConfirmed
(linkNotifyWhile == 0) && !(MACNotify || rxAck || rxLossConfirmed || rxAddConfirmed)
MAC_NOTIFICATION
disableMAC = disabledMAC = True;
!macOperational
!mssOperational
MAC_NOTIFYING
if (prop == Add) {r.AddConfirmed = r.txAddConfirmed = TRUE;}
if (prop == Loss) {r.LossConfirmed = r.txLossConfimed = TRUE;}
macNotifyWhile = MACNotifyTime;
(macNotifyWhile == 0) || macOperational
END_NOTIFICATION
disableMAC = False; macRecoverWhile = MACRecoveryTime;
(macOperational || (macRecoverWhile == 0))

Figure 11—Status Notification state machine

2.10 Receive Process

The Receive Process shall receive and validate MSPDUs as specified in 2.16.

2.11 Transmit Process

The Transmit Process shall transmit and encode MSPDUs as specified in 2.13–2.15. If the Transmit Process is instructed to transmit an MSPDU before it has had the opportunity to transmit a prior MSPDU, that prior MSPDU shall be discarded and not transmitted. If MAC_Operational status provide by the MSS for the port is FALSE, the MSPDU shall be discarded and is not transmitted.

2.12 Management of MSP

An implementation of MSP in a TPMR:

a) May allow the performance parameters (2.5) to be read by management.

b) May allow the performance parameters (2.5) to be modified by management.

c) May maintain each of the following counts for one or both ports of the TPMR:

— acksTransmitted: The number of *acks* transmitted by the port's Transmit Process as a consequence of txAck being set.

— addNotificationsTransmitted: The number of *adds* transmitted by the port's Transmit Process as a consequence of txAdd being set.

— addConfirmationsTransmitted: The number of *add confirms* transmitted by the port's Transmit Process as a consequence of txAddConfirm being set.

— lossNotificationsTransmitted: The number of *loss's* transmitted by the port's Transmit Process as a consequence of txLoss being set.

— lossConfirmationsTransmitted: The number of *loss confirms* transmitted by the port's Transmit Process as a consequence of txLossConfirm being set.

— acksReceived: The number of *acks* received by the port's Transmit Process.

— addNotificationsReceived: The number of *adds* received by the port's Receive Process.

— addConfirmationsReceived: The number of *add confirms* received by the port's Receive Process.

— lossNotificationsReceived: The number of *loss's* received by the port's Receive Process.

— lossConfirmationsReceived: The number of *loss confirms* received by the port's Receive Process.

— addEvents: The number of transitions to STM:ADD directly from STM:DOWN or STM:LOSS.

— lossEvents: The number of transitions to STM:LOSS directly from STM:UP or STM:ADD.

— macStatusNotifications: The number of transitions to SNM:MAC_NOTIFICATION.

2.13 MSPDU transmission, addressing, and protocol identification

MAC Status Protocol Data Units (MSPDUs) are transmitted and received using the service provided by an LLC entity that uses, in turn, a single instance of the MAC Service provided at an MSAP. In a TPMR the MSPE transmit and receives the MSPDUs, and the MSAP is provided by the bridge port transmit and receive function as illustrated in Figure 3. Each MSPDU is transmitted as a single MAC service request, and received as a single MAC service indication, with the following parameters:

a) destination address (2.13.1)
b) source address (2.13.2)
c) MSDU
d) priority (2.13.3)

The MSDU of each request and indication comprises an number of octets that provide Ethertype protocol identification (2.13.4) followed by the MSPDU proper (2.15).

NOTE 1—For the purposes of this standard, the term "LLC entity" includes entities that support protocol discrimination using the Ethertype field as specified in IEEE Std 802a-2003.

NOTE 2—The complete format of an MSP frame 'on the wire' or 'through the air' depends not only on the MSPDU format, as specified in this clause, but also on the media access method dependent procedures used to support the MAC Service.

2.13.1 Destination MAC Address

The destination address for each MAC service request used to transmit an MSPDU shall be the group address identified in Table 3.

Table 3—MSPDU group destination address

Address assignment	Address value
IEEE Std 802.1X PAE group address*	01-80-C2-00-00-03

*This addresses was originally assigned by IEEE Std 802.1X-2001, and was identified as an S-VLAN component Reserved Address (IEEE Std 802.1Q Table 8-2) by the IEEE Std 802.1ad amendment to IEEE Std 802.1Q-2005. This standard further identifies it as the 'nearest non-TPMR bridge address', to be filtered by all relay functions above the sub-layer specified for TPMRs.

2.13.2 Source MAC Address

The source address for each MAC service request used to transmit an MSPDU shall be an individual address associated with the MSAP at which the request is made.

Table 4—MSP Ethernet Type assignment

Assignment	Value
MAC Status Protocol Ethernet Type	wx-yz[*]

[*]The Ethertype value denoted by wx-yz in this table will be assigned at Sponsor ballot.

2.13.3 Priority

The priority associated with each MAC Service request should be the default associated with the MSAP. Transmitted MSPDUs are not Virtual LAN (VLAN) tagged or priority tagged. All TPMRs and other bridges shall be capable of receiving MSPDUs that are untagged, priority tagged, S-VLAN tagged, or C-VLAN tagged. The VID of a tagged frame is ignored on receipt.

NOTE—While a TPMR transmits frames without a VLAN tag, this standard specifies that a TPMR be capable of receiving a tagged frame in order to ease participation in the protocol by end stations.

2.13.4 Ethertype use and encoding

All MSPDUs are identified by the Ethertype specified in Table 4.

Where an individual LAN MAC supports direct encoding of Ethertypes (as does IEEE Std 802.3-2002, for example) the LLC entity shall encode the MSP Ethertype as the first two octets of the MPDU. Otherwise (for IEEE Std 802.5, for example) the MSP Ethertype shall be encoded in the initial octets of the MPDU according to the procedures specified in IEEE Std 802-2001 for Subnetwork Access Protocols (SNAP).

NOTE —The SNAP discriminator comprises the octets AA-AA-03-00-00-00 prepended to the PAE Ethertype.

2.14 Representation and encoding of octets

All MSPDUs consist of an integral number of octets, numbered starting from 1 and increasing in the order that they are put into a MAC frame. The bits in each octet are numbered from 1 to 8, where 1 is the low-order bit. When consecutive octets are used to encode a binary number, the lower numbered octet contains the more significant bits of the binary number.

When the encoding of (an element of) an EAPOL PDU is represented using a diagram in this clause, the following representations are used:

a) Octet 1 is shown toward the top of the page, higher numbered octets being toward the bottom.
b) Where more than one octet appears on a given line, octets are shown with the lowest numbered octet to the left, higher numbered octets being to the right.
c) Within an octet, bits are shown with bit 8 to the left and bit 1 to the right.

2.15 MSPDU structure

The MSPDU comprises the octets following the MSP Ethertype. All MSPDUs comprise a Protocol Version (2.15.1) and a Packet Type (2.15.2).

	Octet number
Protocol Version (2.15.1)	**1**
Packet Type (2.15.2)	**2**

Figure 12—MSPDU structure

2.15.1 Protocol Version

The MSP Protocol Version is encoded in all MSPDUs as a single octet, representing an unsigned binary number. Its value identifies the version of MSP supported by originator of the MSPDU. An implementation conforming to this specification shall encode the value 0000 0000 in this field. All other values are reserved.

NOTE—For TPMRs modeled by this clause's state machine specification in this clause, the originator is the TPMR port that acts on the txAdd, txLoss, txAck, txAddConfirm, or txLossConfirm variables that prompted transmission of the MSPDU. TPMRs that relay an MSPDU do not change its Protocol Version.

2.15.2 Packet Type

The MSP Packet Type is encoded as a single octet, representing an unsigned binary number. Table 5 lists the Packet Types specified by this standard, and the state machine variables set to indicate reception and transmission of MSPDUs of that type. All other possible values of the Packet Type field are reserved and shall not be used.

Table 5—MSP Packet Types

Packet Type	Value	Transmission	Reception
MSP-Add	0	txAdd (2.6.15)	rxAdd (2.6.10)
MSP-Loss	1	txLoss (2.6.17)	rxLoss (2.6.12)
MSP-Add Confirmed	2	txAddConfirm (2.6.16)	rxAddConfirm (2.6.11)
MSP-Loss Confirmed	3	txLossConfirm (2.6.18)	rxLossConfirm (2.6.13)
MSP-Ack	4	txAck (2.6.14)	rxAck (2.6.9)

2.16 Validation of received MSPDUs

To ensure that backward compatibility is maintained for future versions of this protocol, the validation and protocol version handling for all MSPDUs, follows general rules developed for this and other protocols. A received MSPDU shall be processed as specified by Table 5 if and only if:

a) The destination MAC address is the group address specified (2.13.1); and

b) The MSPDU is identified by the MSP Ethernet Type encoded as specified in 2.13.4; and

c) The received MSPDU contains at least two octets, i.e. at least the Protocol Version and Packet Type; and

d) The Packet Type is one of the values specified in Table 5.

Otherwise the received EAPOL PDU shall be discarded. No other checks shall be applied to received MSPDUs, in particular the value of the Protocol Version is not checked and MSPDUs of length greater than the minimum of two octets are accepted as valid.

2.17 Other MSP participants

An end station or non-TPMR bridge attached to the end of a TPMR link can participate in link status notification, avoiding the need for the last TPMR in the chain to use MAC status notification and thus speeding the transition of the link to an operational state. Such a participant receives MSPDUs, but acts only on a received *loss* or *add*, notifying its protocol clients of the change in connectivity, and responding immediately by transmitting an *add confirm* or *loss confirm* as appropriate. There is no need for such a participant to transmit an *add*, *loss*, or *ack*, or act upon a received *ack*, *add confirm* or *loss confirm*, or to initiate MAC status notification.

3. Suggested changes

The suggested replacement text in Section 2 of this note incorporates many detailed changes, both technical and editorial. Some are aimed at making future development easier (if, for example, we feel the need for a formal Protocol Design Requirements and Protocol Support requirements subclauses as was done for MSTP they can be added as 23.1 and 23.2 without reworking the proposed changes). The following covers some of the more significant points.

The proposed text is meant to be stylistically and organizationally compatible with the rest of 802.1Q, taking the existing clause 13 as a model.

The acronyms for the protocol entities, the protocol itself, and for the PDUs sent have been changed (back) to MSPE, MSP, and MSPDU respectively. These acronyms do not clash with any others that are in widespread use in this technical area (and avoid 'MSPP' deliberately). They are consistent with the fact that this amendment does not (and should not) change earlier clauses of the document to replace the term 'MAC status parameters' with anything else (this term is used outside 802.1D/802.1Q).

The protocol overview has been improved to make the interactions with each port's MSS and with the underlying MAC explicit. The overview is explained in terms of the existing MAC status parameters—MAC_Operational and MAC_Enabled for each of these interface stack components.

The number of cooperating state machines has been reduced, mainly as a consequence of Panos' earlier ballot comment which showed that two of the signal variables needed to be set by two machines. So machines have been multiplied out, which actually had the effect of reducing the total number of states and transitions (albeit very slightly) which is a good test of whether that multiplication should have been done. Some undesirable behavior has been removed (in the case when the two ports of the TPMR come up together). The kludge of distinguishing variables by quoting them, as in 'Add' has been removed, I don't believe that is portable C.

The control variables for the status notification behavior have been separated for the timer values, and ranges suggested for the latter.

Management statistics have been added.

The PDU formats now deal properly with LANs that have to use LLC SNAP to convey an Ethertype, and contains a proper set of validation rules.

4. Alternatives and futures

The original note on this subject aj-seaman-status-propagation-0806-01.pdf considered a number of different signaling methods for the protocol including extension to CFM or the use of MAC specific methods. The description of the protocol provided the flexibility to accommodate these, and a wider ranging of timing expectations for signaling notifications, acknowledgements, and confirmations, possibly using different mechanisms for each. This note is specific about one approach, but that does not mean that a relay at the level of a TPMR could not use different methods for different media, though there are advantages to having the notifications and confirmations continue to follow the specific form proposed in Section 2 —to provide the widest interoperability.

The original note also discussed the idea of applying status propagation to virtual links, even to multi-point links. While that idea remains valid (and could be part of providing a proper connection oriented service) it should not be allowed to interfere with or miscellaneously extend the P802.1aj project. How and when to revert to lower layer signaling when communicating a connection break or 'reset' on a virtual link is a rather different subject from using MAC status notification, and the presentation in Section 2 makes full use of being in a two port rather than a multi-port environment. Before any work on extensions were to take place there would have to be a proper problem statement and the details of the state machines would have to change to fit that problem statement. Similarly there is much additional work to be done if loss signaling were to be combined with repair. While that could be interesting work it goes some way beyond Section 2, and needs someone with a lot of time to produce a convincing proposal (if there is a case to be made) including detailed protocol and state machines.