INTERNATIONAL TELECOMMUNICATION UNION

TELECOMMUNICATION STANDARDIZATION SECTOR

STUDY PERIOD 2013-2016

STUDY GROUP 15

TD 037 Rev.1 (PLEN/15)

English only

Original: English

Question(s): 10/15 1-12 July 2013

TD

Source: Editor G.8121.2/Y.1381.2

Title: Draft new Recommendation ITU-T G.8121.2/Y.1381.2 (for Consent, July 2013)

Abstract

This document contains the latest draft of new Recommendation G.8121.2 for consent. This document has been updated in Hiroshima meeting (January 2013) as wd06r1 and in Geneva as wd15r1_T13-SG15-130701-TD-PLEN-0037!!MSW-E.docx

Update history

Update in Hiroshima meeting (i.e. **wd06r1**)

- WD10 and WD18: Updated clause 1 to 5
- WD19: It was agreed the table for clause 8.8 should be in G.8121 rather than G.8121.2. So far the following text will be removed or modified after the text and table in G.8121 are completed. As to the proposed update to clause 9.2 and 9.4, the updated as proposed with adding "of [ITU-T G.8121].

Update after Hiroshima meeting (Shown by "G.8121 Editor ")

- Filled figure/table numbers
- Updated the description style of Recommendations

Drafting in Geneva, Jul 2013

See wd15r1_T13-SG15-130701-TD-PLEN-0037!!MSW-E.docx

Contact:	Yuji Tochio Fujitsu Japan	Tel: +81-44-754-8829 Fax: +81-44-754-2741 Email: tochio@jp.fujitsu.com
Contact:	Huub van Helvoort Huawei Technologies P.R .China	Tel: +31-20-4300-8108 Fax: +31-20-4300-888 Email: hhelvoort@huawei.com

Draft new Recommendation ITU-T G.8121.2/Y.1381.2

Characteristics of MPLS-TP equipment functional blocks supporting ITU-T G.8113.2/Y.1373.2

Summary

Recommendation ITU-T G.8121.2 specifies both the functional components and the methodology that should be used in order to specify MPLS-TP layer network functionality of network elements based on the protocol neutral constructs defined in ITU-T G.8121 and on the tools defined in ITU-T G.8113.2/Y.1373.2.

<Mandatory material>

Keywords

Atomic functions, equipment functional blocks, MPLS-TP layer network, MPLS-TP.<Optional>

1 Scope

*[CD04 (Correspondence) Proposes to update the scope, see also meeting report]*This Recommendation describes both the functional components and the methodology that should be used in order to describe MPLS-TP layer network functionality of network elements; it does not describe individual MPLS-TP network equipment as such.

This recommendation provides protocol-specific extensions of the protocol-neutral constructs defined in [ITU-T G.8121] to support the OAM tools defined in [ITU-T G.8113.2].

This Recommendation provides a description of the MPLS-TP functional technology using the same methodologies that have been used for other transport technologies (e.g. SDH, OTN and Ethernet)[1]. [copy from com15-R33, c4.10]

This Recommendation, along with [ITU-T G.8121], specifies a library of basic building blocks and a set of rules by which they may be combined in order to describe digital transmission equipment. The library comprises the functional building blocks needed to specify completely the generic functional structure of the MPLS-TP layer network. In order to be compliant with this Recommendation, equipment needs to be describable as an interconnection of a subset of these functional blocks contained within this Recommendation. The interconnections of these blocks should obey the combination rules given.

Not every atomic function defined in this Recommendation is required for every application. Different subsets of atomic functions may be assembled in different ways according to the combination rules given in this Recommendation to provide a variety of different capabilities. Network operators and equipment suppliers may choose which functions must be implemented for each application.

2 References

The following ITU-T Recommendations and other references contain provisions, which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision;

[1] This ITU-T Recommendation is intended to be aligned with the IETF MPLS RFCs normatively referenced by this Recommendation.

users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published.

The reference to a document within this Recommendation does not give it, as a stand-alone document, the status of a Recommendation.

[ITU-T G.805] Recommendation ITU-T G.805 (2000), *Generic functional architecture of transport networks.*

[ITU-T G.806] Recommendation ITU-T G.806 (2004), Characteristics of transport equipment – Description methodology and generic functionality.

[ITU-T G.8101] Recommendation ITU-T G.8101 (2012), Terms and definitions for MPLS transport profile

[ITU-T G.8110.1] Recommendation ITU-T G.8110.1/Y.1370.1 (2011), Architecture of MPLS Transport Profile (MPLS-TP) layer networks.

[ITU-T G.8113.2] Recommendation ITU-T G.8113.2 (2012), Operations, administration and maintenance mechanisms for MPLS-TP networks using the tools defined for MPLS

[ITU-T G.8121] Recommendation ITU-T G.8121 (2012), Characteristics of MPLS-TP network equipment functional blocks

[IETF RFC 4379] IETF RFC 4379 (2006), Detecting Multi-Protocol Label Switched (MPLS) Data Plane Failures

[IETF RFC 5586] IETF RFC 5586 (2009), MPLS Generic Associated Channel.

[IETF RFC 5654] IETF RFC 5654 (2009), Requirements of an MPLS Transport Profile.

[IETF RFC 5718] IETF RFC 5718 (2010), An In-Band Data Communication Network For the MPLS Transport Profile.

[IETF RFC 5860] IETF RFC 5860 (2010), Requirements for OAM in MPLS Transport Networks.

[IETF RFC 5880] IETF RFC 5880 (2010), Bidirectional Forwarding Detection (BFD)

[IETF RFC 5884] IETF RFC 5884 (2010), Bidirectional Forwarding Detection (BFD) for MPLS Label Switched Paths (LSPs)

[IETF RFC 5921] IETF RFC 5921 (2010), A Framework for MPLS in Transport Networks.

[IETF RFC 6215] IETF RFC 6215 (2011), MPLS Transport Profile User-to-Network and Network-to-Network Interfaces

[IETF RFC 6370] IETF RFC 6370 (2011), MPLS Transport Profile (MPLS-TP) Identifiers

[IETF RFC 6374] IETF RFC 6374 (2011), Packet Loss and Delay Measurement for MPLS Networks.

[IETF RFC 6375] IETF RFC 6375 (2011), A Packet Loss and Delay Measurement Profile for MPLS-based Transport Networks

[IETF RFC 6423] IETF RFC 6423 (2011), Using the Generic Associated Channel Label for Pseudowire in the MPLS Transport Profile (MPLS-TP)

[IETF RFC 6426] IETF RFC 6426, MPLS On-Demand Connectivity Verification and Route Tracing.

[IETF RFC 6427] IETF RFC 6427, MPLS Fault Management Operations, Administration, and Maintenance (OAM).

[IETF RFC 6428] IETF RFC 6428, Proactive Connectivity Verification, Continuity Check and Remote Defect Indication for the MPLS Transport Profile.

~~*[to be finalized in next meeting]*~~

3 Definitions

~~<Check in the ITU-T Terms and definitions database on the public website whether the term is already defined in another Recommendation. It may be more consistent to refer to such a definition rather than redefine it>~~~~*[to be finalized in next meeting]*~~

3.1 Terms defined elsewhere:

This Recommendation uses the following terms defined elsewhere:

~~**3.1.1 <Term 1>** [Reference]: <optional quoted definition>~~

3.1.1 access point: [ITU-T G.805]

3.1.2 adapted information: [ITU-T G.805]

3.1.3 characteristic information: [ITU-T G.805]

3.1.4 client/server relationship: [ITU-T G.805]

3.1.5 connection: [ITU-T G.805]

3.1.6 connection point: [ITU-T G.805]

3.1.7 layer network : [ITU-T G.805]

3.1.8 matrix: [ITU-T G.805]

3.1.9 network: [ITU-T G.805]

3.1.10 network connection: [ITU-T G.805]

3.1.11 reference point: [ITU-T G.805]

3.1.12 subnetwork: [ITU-T G.805]

3.1.13 subnetwork connection: [ITU-T G.805]

3.1.14 termination connection point : [ITU-T G.805]

3.1.15 trail: [ITU-T G.805]

3.1.16 trail termination: [ITU-T G.805]

3.1.17 transport: [ITU-T G.805]

3.1.18 transport entity: [ITU-T G.805]

3.1.19 transport processing function: [ITU-T G.805]

3.1.20 unidirectional connection: [ITU-T G.805]

3.1.21 unidirectional trail: [ITU-T G.805]

3.1.22 label: [ITU-T G.8101]

3.1.23 label stack: [ITU-T G.8101]

3.1.2 MPLS label stack: [ITU-T G.8101]

3.1.24 label switched path: [ITU-T G.8101]

3.1.25 Bottom of Stack: [ITU-T G.8101]

3.1.26 Time To Live: [ITU-T G.8101]

3.1.27 Label value: [ITU-T G.8101]

3.1.28 Per-Hop Behaviour: [ITU-T G.8101]

3.1.29 Associated Channel Header: [ITU-T G.8101]

3.1.30 Generic Associated Channel: [ITU-T G.8101]

3.1.31 G-ACh Label: [ITU-T G.8101]

3.1.32 traffic class: [ITU-T G.8101]

3.1.33 Explicitly TC-encoded-PSC LSP: [ITU-T G.8101]

3.1.34 label inferred PHB scheduling class LSP: [ITU-T G.8101]

3.2 Terms defined in this Recommendation

None ~~This Recommendation defines the following terms:~~

~~**3.2.1** **<Term 3>:** <definition>~~

4 Abbreviations and acronyms

~~*[to be finalized in next meeting]*~~

This Recommendation uses the following abbreviations and acronyms:

BFD Bidirectional Forwarding Detection

CCCV Continuity Check and Connectivity Verification

CC/CV Continuity Check and~~or~~ Connectivity Verification ~~[footnote: In RFCs, CC-V is used]~~

DLM Direct Loss Measurement

DSMap Downstream Mapping

FEC Forwarding Equivalence Class

FFS For Further Study

ILM Inferred Loss Measurement

LI Lock Instruct

LKR Lock Report

MTU Maximum Transmit Unit

ODCV On-Demand Connectivity Verification

QTF Querier’s Timestamp Format

Req Request

Resp Response

RPTF Responder's Preferred Timestamp Format

RTF Responder's Timestamp Format

SQI Session Query Interval

TS Timestamp

TSFmt Timestamp Format

BFD Bidirectional Forwarding Detection

CC Continuity Check

CV Continuity Verification

DLM Direct Loss Measurement

DM Delay Measurement

DSMap Downstream Mapping

FEC Forwarding Equivalence Class

GAL Generic Associated Channel Label

G-ACh Generic Associated Channel

ILM Inferred Loss Measurement

LBI Loopback Instruct

LDI Local Down Indication

LKI Lock Instruct

LKR Lock Report

TC Traffic Class

VCCV Virtual Circuit Connectivity Verification

5 Conventions

The diagrammatic convention for connection-oriented layer networks described in this Recommendation is that of [ITU T G.805].

6 Supervision

6.1 Defects

6.1.1 Summary of Entry/Exit conditions for defects

The defect Entry and Exit conditions are based on events. Occurrence or absence of specific events may raise or reset specific defects.

The events used by this recommendation are defined in Table 6-1/G.8121. The Events, unexpCCPeriod, unexpCVPeriod, expCV, CSF-LOS, CSF-FDI and CSF-RDI, that are described in Table 6-1/G.8121 are out of scope of this Recommendation.

[Ed Note – check that all the events in G.8121 are used or no missing events. Events as marked yellow need to be consistency with G.8121 (for AR)]

7 Information flow across reference points

~~7~~ Information flow for MPLS-TP functions is defined in clause 9. A generic description of information flow is defined in clause 7 of [ITU-T G.806].

8 MPLS-TP processes

8.1 G-ACh Process

In the case where OAM packets are encapsulated using a Generic Associated Channel (G-ACh), the G-Ach Process is described in Clause 8.1/G.8121. Encapsulation of OAM packets using IP/UDP or other mechanisms is FFS

8.2 TC/Label processes

See the clause 8.2 in [ITU-T G.8121]

8.3 Queuing process

See the clause 8.3 in [ITU-T G.8121]

8.4 MPLS-TP-specific GFP-F processes

See the clause 8. 4 in [ITU-T G.8121]

8.5 Control Word (CW) processes

See the clause 8.5 in [ITU-T G.8121]

8.6 OAM related Processes used by Server adaptation functions

8.6.1 Selector Process

See the clause 8.6.1in [ITU-T G.8121]

8.6.2 AIS Insert~~ion~~ Process

~~*[Note Updates per C.2373 and/or CD01 below are shown as yellow marked, not diffmark]*~~

The AIS Insert Process generates MT_CI traffic units containing the AIS signal. MI_AIS_Period specifies the period between successive AIS messages, in seconds between 1 and 20. MI_AIS_CoS specifies the priority for AIS messages. MI_Local_Defect specifies whether an alternative path is available – that is, it is set to true when either the server layer does not provide any protection, or when both the working and protect paths have faults The AIS insert process behaviour depends on the aAIS ~~and aSSF~~ consequent action.

Note: it is expected that MI_Local_Defect can be set correctly by the EMF without explicit interaction by the end user. The value can be precomputed as described in [RFC6427].

The AIS Insert Process is described in Clause 8.6.2 in ~~/~~[ITU-T G.8121], and is shown in Figure 8-1~~XX~~

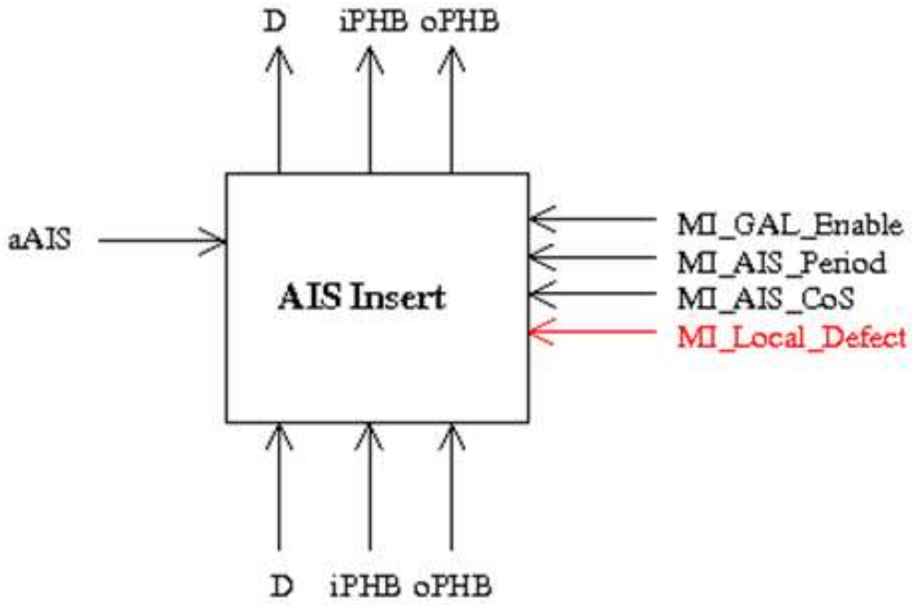


Figure 8-1xx/G.8121.2/Y.1381.2 AIS Insertion process
[Note: Updated per cd01]

Figure 8-xx 2 defines the behaviour of the AIS Insert Process:

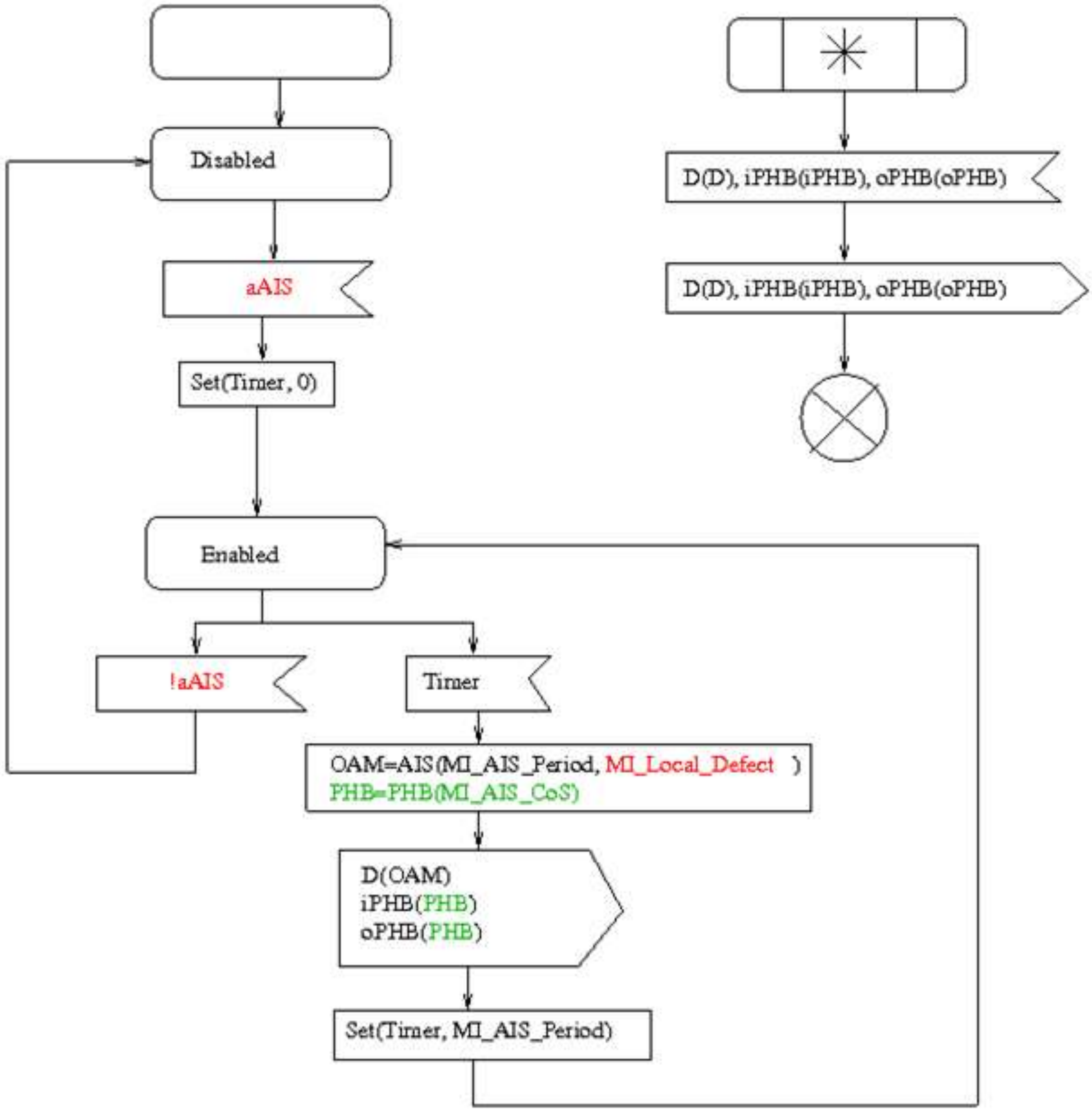


Figure 8-~~xx~~2/G.8121.2/Y.1381.2 AIS Insert behaviour
~~*[Note: Updated per cd01, (L not defied)]*~~

The AIS function creates an AIS frame, by first creating an AIS PDU, and then encapsulating it in a G-ACh and, depending on MI_GAL_Enable, a GAL, as described in Clause 8.1/G.8121. It then inserts it into the data traffic stream. The PHB(CoS) function returns the PHB with the lowest drop precedence within the Class of Service defined by the CoS input parameter.

The AIS PDU is created according to the format described in [RFC 6427]. The fields are filled in follows:

- Vers: set to 1
- Reserved: set to 0

- Message Type: set to AIS
- Flags: The L flag is set to 1 if MI_Local_Defect is true, and is otherwise set to 0. The remaining flags are set to 0.
- Refresh Timer: set to MI_AIS_Period
- Total TLV Length: set to 0

Inclusion of the IF_ID and Global_ID TLVs is FFS.

The PHB(CoS) function returns the PHB with the lowest drop precedence within the Class of Service defined by the CoS input parameter.

8.6.2.1 LCK/AIS ReceptionExtract Process

[Note: To be moved to clause 8.8.x after G.8121 amendment approved]

The LCK/AIS ReceptionExtract Process handles received LKR and AIS packets, and signals the LCK, AIS and SSF defects. The behaviour is shown in Figure 8-xx3.

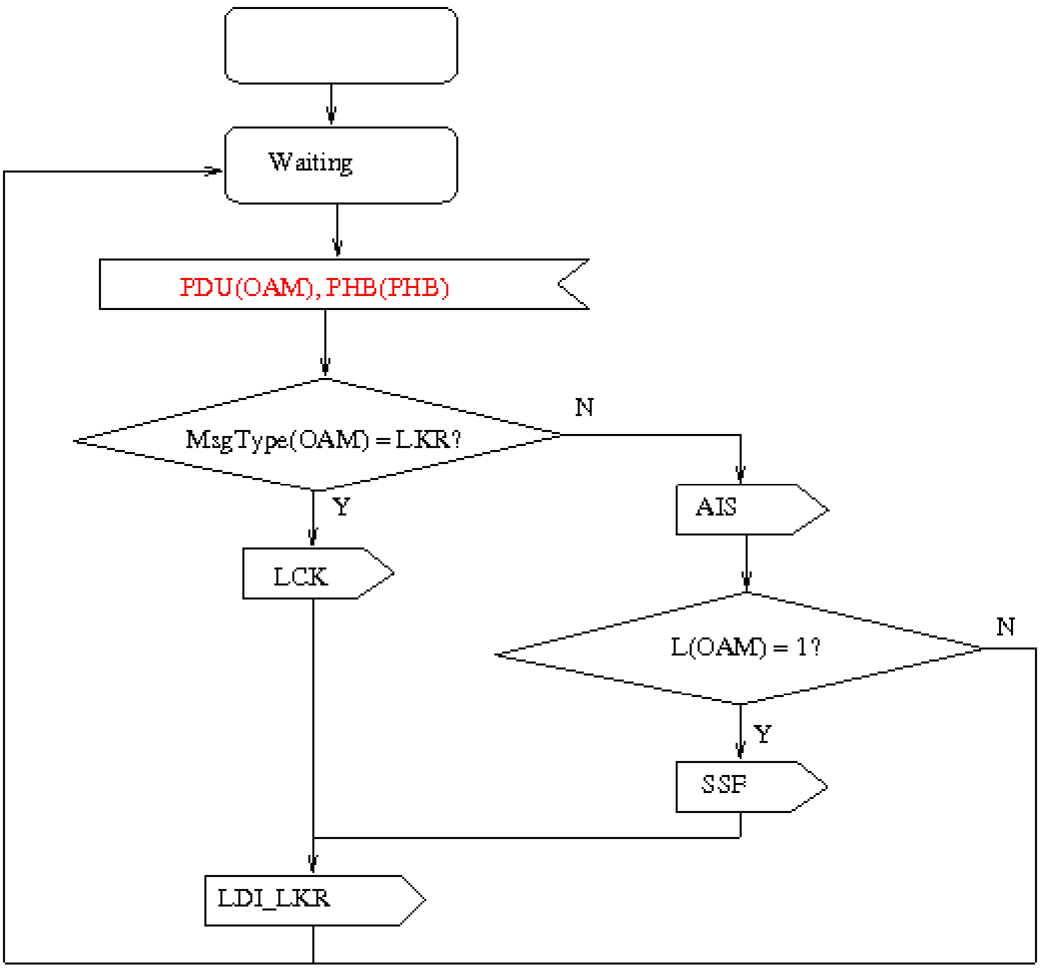


Figure 8-xx3/G.8121.2/Y.1381.2 LCR/AIS ReceptionExtract behaviour

8.6.3 LCK Generation Process

The LCKR Generation Process generates MT_CI traffic units containing the Lock signal, i.e. containing LKR messages. MI_LCK_Period specifies the period between successive LKR messages, in seconds between 1 and 20. MI_LCK_CoS specifies the priority for LKR messages.

Note: IETFEEE uses "LKR" (Lock Report) equivalently to the ITU-T use of "LCK".

The LCKR Generation Process is described in Clause 8.6.3/G.8121 and its behaviour is shown in Figure 8-xx4.

[Note: Usage LCK and LKR message should be clarified???]

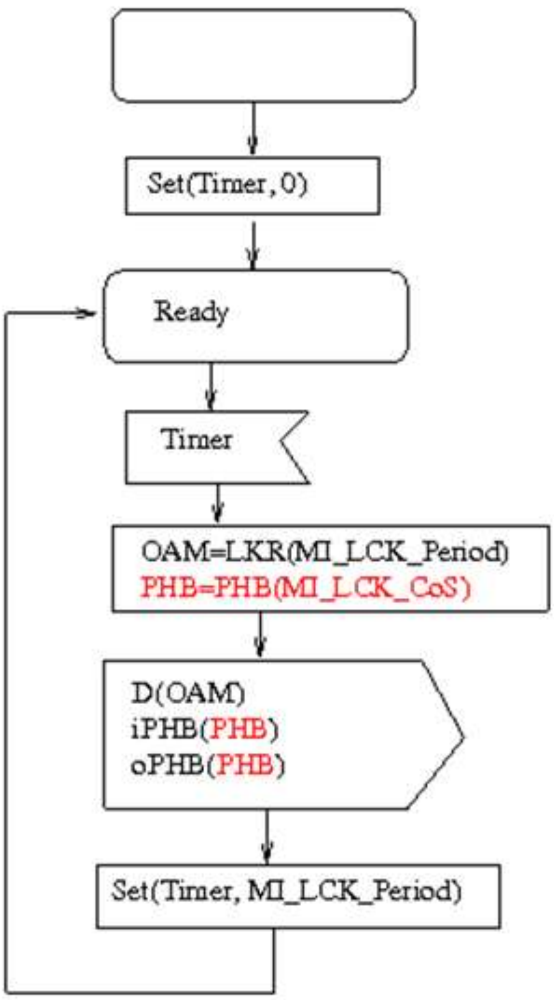


Figure 8-xx4/G.8121.2/Y.1381.2 - LCK Generation behaviour [Annex III, C.2025]

The LKR function creates an LKR frame, by first creating an LKR PDU, and then encapsulating it in a G-ACh and, depending on MI_GAL_Enable, a GAL, as described in Clause 8.1/G.8121.

The LKR PDU is created according to the format described in [RFC 6427]. The fields are filled in as follows:

- Vers: set to 1
- Reserved: set to 0

- Message Type: set to LKR
- Flags: set to 0
- Refresh Timer: set to MI_LCK_Period
- Total TLV Length: set to 0

Inclusion of the IF_ID and Global_ID TLVs is FFS.

The PHB(CoS) function returns the PHB with the lowest drop precedence within the Class of Service defined by the CoS input parameter.

8.7 OAM related Processes used by adaptation functions

8.7.1 MCC/SCC Mapping Insert and De-mapping Process

See the clause 8.7.1in [ITU-T G.8121]

8.7.2 APS Insert and ExtractProcess

See the clause 8.7.2 in [ITU-T G.8121]

8.7.3 CSF Insert and Extract Process

See the clause 8.7.3 in [ITU-T G.8121]

8.8 Pro-active and on-demand OAM related Processes

~~[Note: In reviewing WD19, it was pointed out the table below should be in G.8121 rather than G.8121.2. So far the following text will be removed or modified after the text and table in G.8121 isare completed.]~~

As described in Clause 8.8/G.8121, there are 6 processes for pro-active and on-demand OAM:

- Proactive OAM Source Control
- Proactive OAM Sink Contol
- On-demand OAM Source Control
- On-demand OAM Sink Control
- OAM PDU Generation
- OAM PDU Reception

Each of these consists of a number of protocol-specific sub-processes, as described in G.8121. Appendix I provides tThe table thatbelow showsindicates the relationship between processes and sub-processes and indicates where these (sub-)processes are implemented to the termination functions (MT_TT, MTDe_TT, and MTDi_TT). ~~described in this document:~~

Table 8-1/G.8121.2/Y.1381.2 OAM Process and subprocesses

8.8.1.1.1 Process	8.8.1.1.2 Sub-processes
8.8.1.1.3 Proactive OAM Source Control	8.8.1.1.4 CCCV Generation 8.8.1.1.5 LI Source Control
8.8.1.1.6 Proactive OAM Sink Control	8.8.1.1.7 CCCV Reception 8.8.1.1.8 LCK/AIS Reception 8.8.1.1.9 LI Sink Control 8.8.1.1.10 Proactive PM Control 8.8.1.1.11 PM Responder
8.8.1.1.12 On-demand OAM Source Control	
8.8.1.1.13 On-demand OAM Sink Control	8.9 On-demand CV Control 8.9.1.1.1 MIP On-demand CV Responder 8.9.1.1.2 MEP On-demand CV Responder 8.9.1.1.3 On-demand PM Control 8.9.1.1.4 PM Responder
8.9.1.1.5 OAM PDU Generation	8.9.1.1.6 On-demand CV Request Generation 8.10 On-demand CV Response Generation 8.10.1.1.1 OAM Mux 8.10.1.1.2 LI Generation PM Mux 8.10.1.1.3 PM Generation
8.10.1.1.4 OAM PDU Reception	8.10.1.1.5 Session Demux 8.10.1.1.6 On-demand CV Reception 8.10.1.1.7 OAM Demux 8.10.1.1.8 LI Reception PM Demux 8.10.1.1.9 PM Reception

[Ed Note: the highlighted entries are not present in the existing G.8121.2 text but are proposed in wd22 and wd23] [Note 2 – Determine to delete this note with highlighted in January 13]

The OAM Mux Sub-process is responsible for multiplexing together (PDU, TTL, PHB) signals from other sub-processes, and passing them to the G-ACh Insertion process along with the appropriate Channel Type. Similarly, the OAM Demux sub-process receives (PDU, PHB, LStack, Channel Type) signals from the G-ACh Extraction process, and passes on the (PDU, PHB, LStack) signals to the other sub-processes as appropriate depending on the Channel Type.

The following subclauses describes the other sub-processes listed above. They are organised by function (eg CCCV, On-demand CV, etc), with all of the sub-processes relevant to a particular function described together.

[Note: CCCV should be updated]

8.8.1. CC/CV Processes

An overview of the CC/CV processes is shown in the Ffigure 8-5 below:

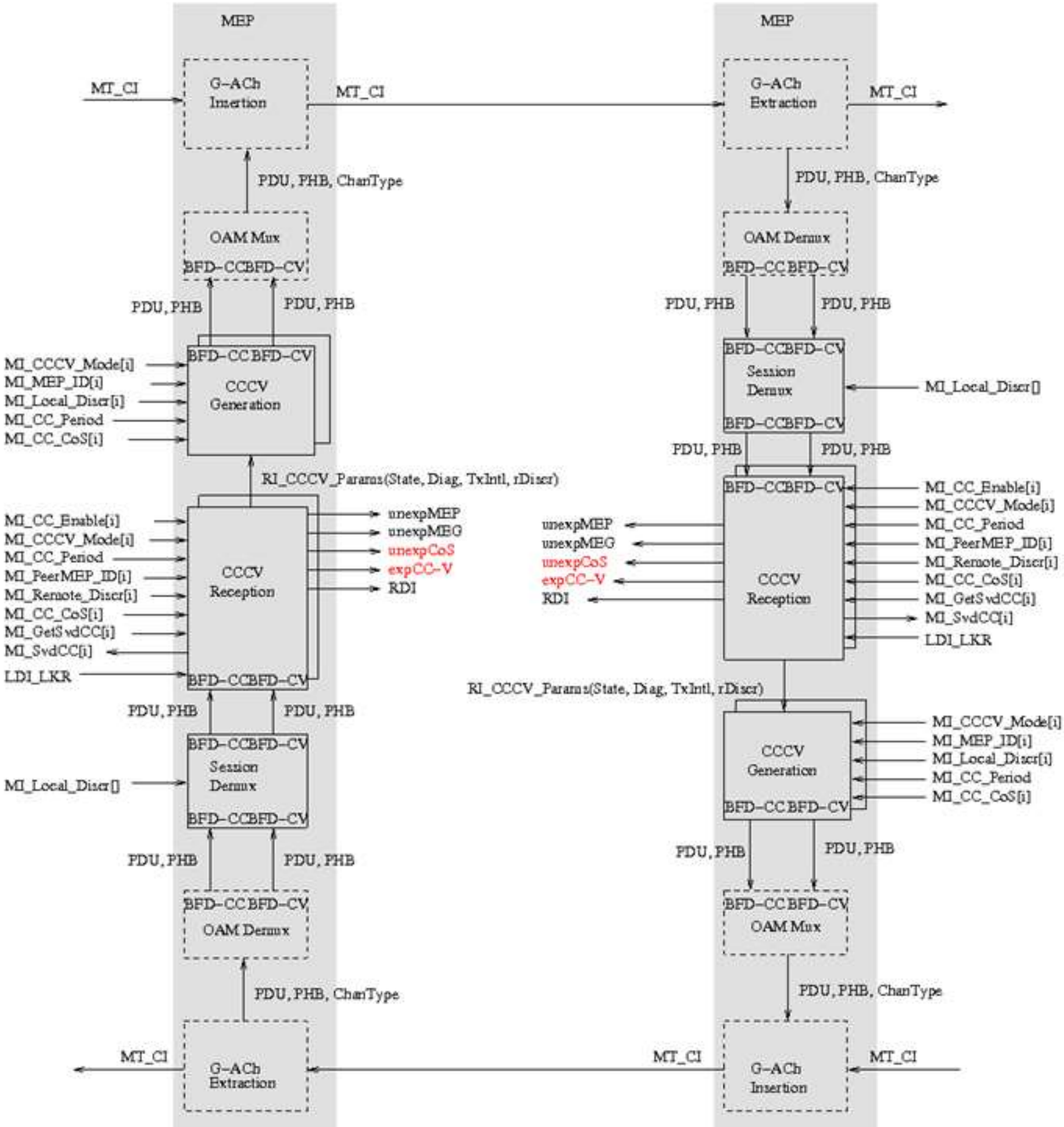


Figure 8-x5/G.8121.2/Y.1381.2 – Overview of CC/CV processes
[updated per Annex IV, C.2025]

The CCCV reception process controls the operation of the CCCV protocol. It operates when MI_CC_Enable is TRUE, according to the value of MI_CCCV_Mode. MI_CCCV_Mode takes one of the following values:

- COORD – Co-ordinated mode; operate a single co-ordinated BFD session
- SRC – Independent Source; operate as the source MEP in an independent BFD session
- SINK – Independent Sink; operate as the sink MEP in an independent BFD session

Note- [RFC 6428] defines two modes for bidirectional LSPs operation, i.e. Coordinated mode and Independent mode. In independent mode, separate sessions are used for each direction and a given MEP operates as the source for one session and the sink for the other session. Thus, there are three possible values for MI_CCCV_Mode as shown above.

Multiple instances of the CCCV reception process may be created for multiple BFD sessions; when operating in independent mode, it is expected that a pair of instances are created, one acting as the source and one as the sink.

MI_CC_Period specifies the desired period between successive BFD-CC messages, and MI_PeerMEP_ID specifies the MEP ID value to expect in received messages, in one of the formats described in [RFC 6428].

The CCCV generation process sends periodic BFD-CC and BFD-CV messages, when MI_CC_Enable is TRUE. There is a separate instance of the process for each corresponding instance of the CCCV reception process. MI_MEP_ID and MI_Local_Discr specify the local MEP ID and session discriminator values to send in the packets.

The Session Demux process demultiplexes received BFD-CC and BFD-CV messages to the correct instance of the CCCV reception process, based on the “Your discriminator” field in the received BFD-CC or BFD-CV packet. Demultiplexing of received packets where the “Your discriminator” field is 0 is FFS.

8.8.1.1. CCCV Reception Process

The CCCV Reception Process controls the operation of the BFD protocol, according to MI_CC_Enable and MI_CCCV_Mode. Multiple instances of the CCCV Reception Process can be instantiated. Each one has a corresponding instance of the CCCV Generation Process; the contents and period for sending CCCV packets are controlled via the RI_CCCV_Params() signal.

The CCCV Reception Process is described in Figure 8-~~xx~~6, Figure 8-~~xx~~7, and Figure 8-~~xx~~8x. In Disabled state, all received BFD-CC and BFD-CV packets are discarded and no packets are sent. In Enabled state, received BFD-CC packets are processed, and received BFD-CV packets are processed when the BFD state machine is UP. BFD-CC and BFD-CV packets are sent, except if the process is operating in SINK mode. When MI_CC_Enabled is set to FALSE, the process moves to Disabling state so that the ADMIN_DOWN diagnostic code can be signalled to the peer MEP. The process stays in Disabling state for three times the transmit interval, before moving to Disabled state. In Disabling state, BFD-CC packets are sent, but received BFD-CC and BFD-CV packets are used only for updating the timer.

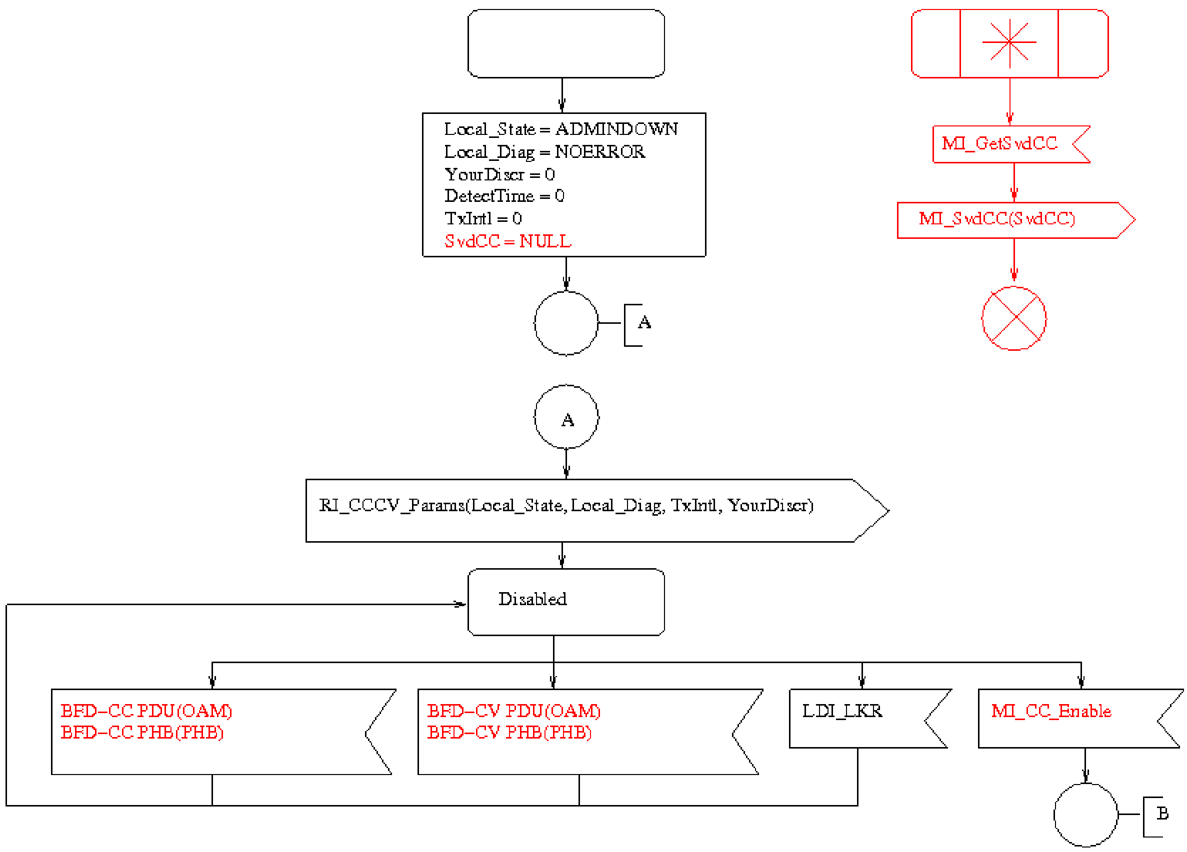


Figure 8-~~6x~~/G.8121.2/Y.1381.2 - CCCV Reception Process (A)

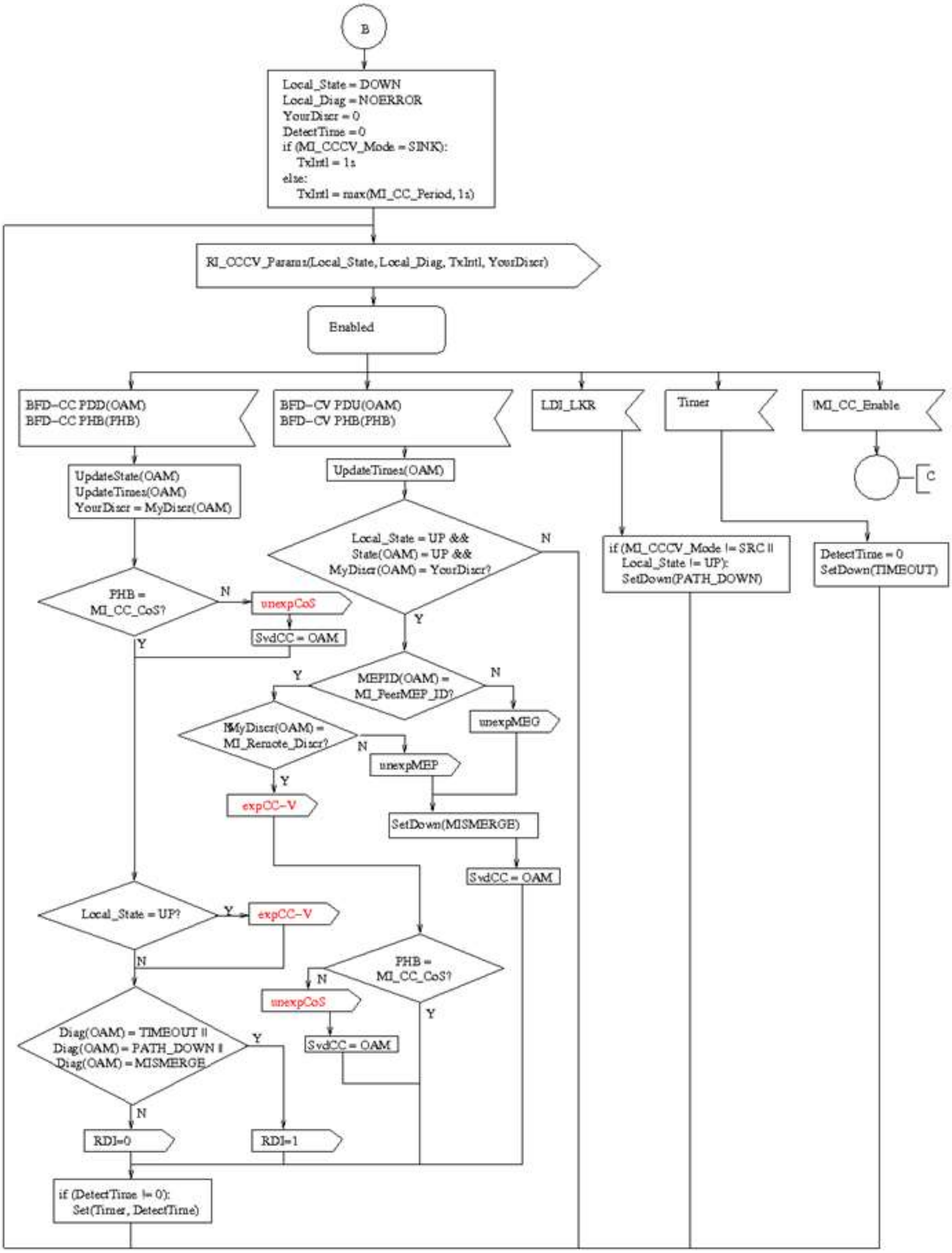


Figure 8-~~x~~7/G.8121.2/Y.1381.2 - CCCV Reception Process (B)
~~*[updated per Annex V, C.2025]*~~

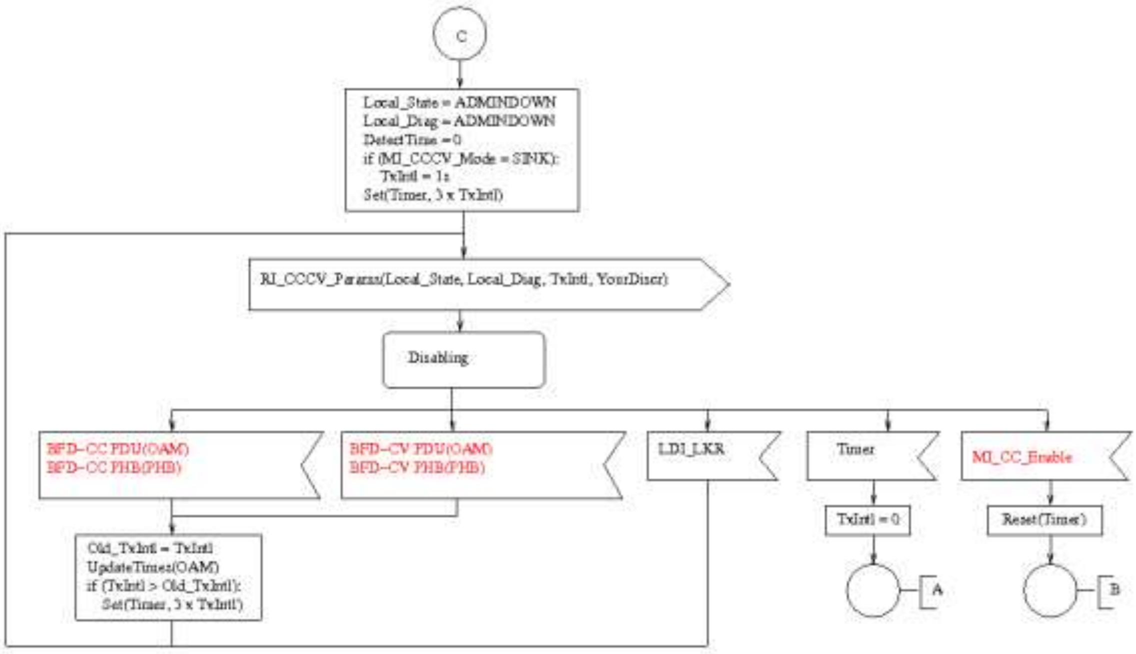


Figure 8-~~x~~8/G.8121.2/Y.1381.2 - CCCV Reception Process (C)

The values of State and Diag correspond with those in [RFC5880] and [RFC6428].

The functions ‘SetDown’, ‘UpdateState’ and ‘UpdateTimes’ are described by the following pseudocode:

```
SetDown(new_diag) {
    if (Local_State != DOWN) {
        Local_State = DOWN
        if (Local_Diag != PATH_DOWN || new_diag != TIMEOUT) {
            Local_Diag = new_diag
        }
        if (MI_CCCV_Mode = SINK) {
            TxIntl = 1s
        }
    }
}

UpdateState(OAM) {
    if (State(OAM) = ADMINDOWN) {
        SetDown(NBR_DOWN)
    } else {
        if (Local_State = DOWN) {
            if (State(OAM) = DOWN) {
                Local_State = INIT
                Local_Diag = NOERROR
```

```
            } else if (State(OAM) = INIT ||
                       (MI_CCCV_Mode = SINK && State(OAM) = UP)) {
                Local_State = UP
                Local_Diag = NOERROR
            }
        } else if (Local_State = INIT) {
            if (State(OAM) = INIT || State(OAM) = UP) {
                Local_State = UP
                Local_Diag = NOERROR
            }
        } else {
            // Local_State must be UP
            if (state(OAM) = DOWN && MI_CCCV_Mode != SRC) {
                SetDown(NBR_DOWN)
            }
        }
    }
}

UpdateTimes(OAM) {
    if (MI_CCCV_Mode = SRC) {
        DetectTime = 0
    } else {
        DetectTime = 3 x max(MI_CC_Period, DesiredMinTxInterval(OAM))
    }
    if (MI_CCCV_Mode = SINK) {
        if (State(OAM) != LocalState) {
            TxIntl = 1s
        } else {
            TxIntl = 0
        }
    } else {
        TxIntl = max(MI_CC_Period, RequiredMinRxInterval(OAM))
    }
}
```

Use of authentication for CC/CV is FFS.

Use of the BFD Poll/Final mechanism for changing the value of TxIntl is FFS.

8.8.1.2. CCCV Generation Process

The CCCV Generation Process is responsible for generating BFD-CC and BFD-CV packets, according to the parameters set by the corresponding CCCV Reception Process in the RI_CCCV_Params(state, diag, TX-interval,your-discriminator) signal. When the TX-interval is set to 0, no BFD-CC or BFD-CV packets are generated. Otherwise, BFD-CC packets are generated at the specified interval, and BFD-CV packets are generated if the state is up, at an interval of 1s.

The CCCV Generation Process is described in Figure 8-~~xx~~9.

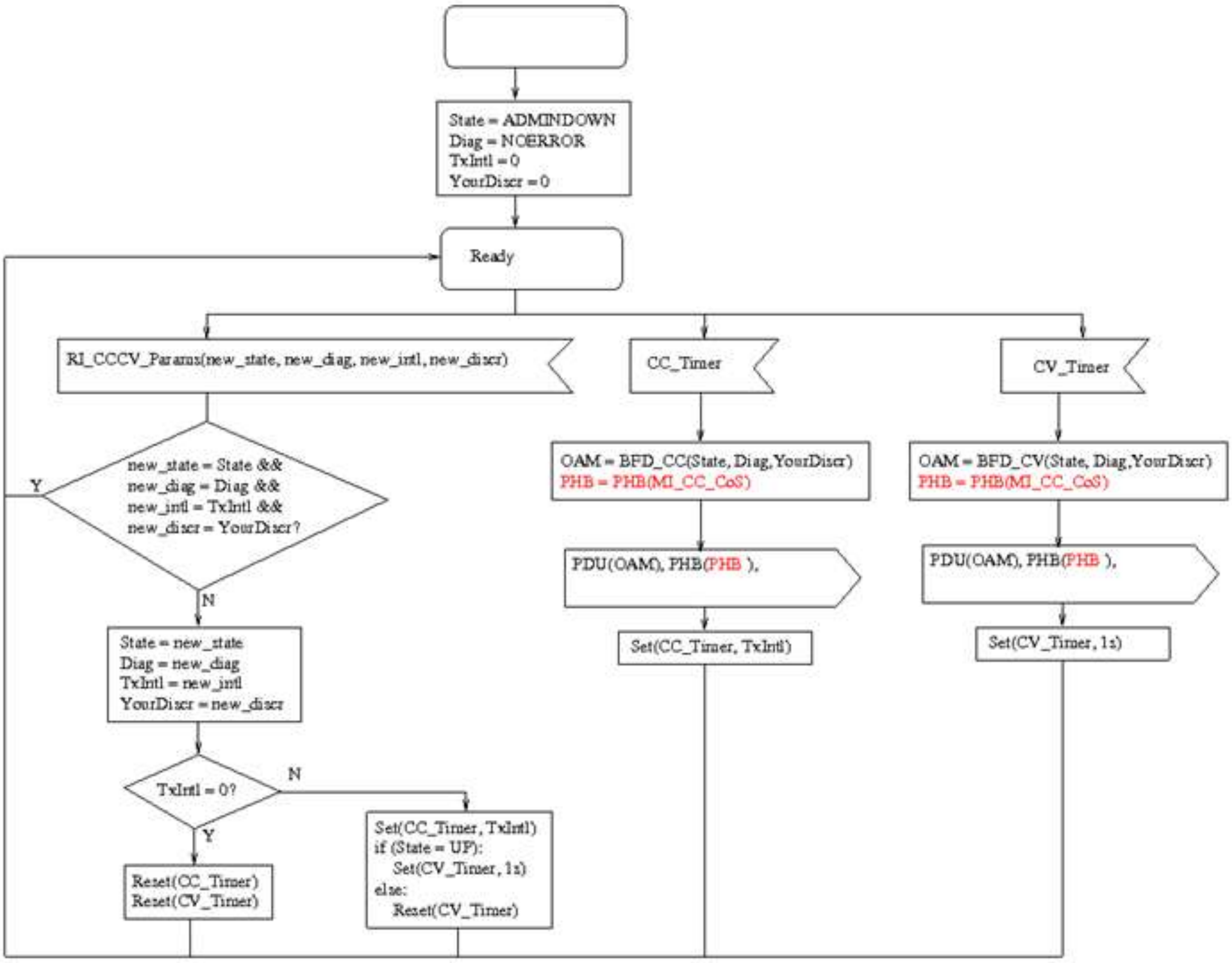


Figure 8-~~xx~~9/G.8121.2/Y.1381.2 - CCCV Generation Process
~~*[updated per Annex VII, C.2025]*~~

The BFD_CC function creates a BFD control packet according to the format described in [RFC5880]. The fields are filled in as follows:

- Vers: set to 1
- Diag: set to the value of Diag
- Sta: set to the value of State
- P, F, A, D, M flags: set to 0
- C flag: set appropriately dependent on the implementation
- Detect Mult: set to 3

- Length: set to 24
- My Discriminator: set to MI_Local_Discr
- Your Discriminator: set to YourDiscr
- Desired Min Tx Interval: set to 0 if MI_CCCV_Mode is SINK, otherwise set to MI_CC_Period
- Required Min Rx Interval: set to 0 if MI_CCCV_Mode is SRC, otherwise set to MI_CC_Period
- Required Min Echo Rx Interval: set to 0

No Authentication Section is added. Use of Authentication is FFS.

The BFD_CV function creates a BFD control packet in the same way as the BFD_CC function, and then appends a MEP Source ID TLV as described in [RFC6428], containing the value of MI_MEP_ID.

The PHB(CoS) function returns the PHB with the lowest drop precedence within the Class of Service defined by the CoS input parameter.

8.8.1.3. Session Demux Process

The session demux process receives BFD-CC and BFD-CV packets from the OAM Demux process. It performs the following checks on the packet:

- If the version number is not 1, the packet is discarded
- If the length is less than 24, the packet is discarded
- If the Detect Mult field is 0, the packet is discarded
- If any of the P, F, A, D, or M flags are set, the packet is discarded
- If the My Discriminator field is 0, the packet is discarded
- If the Required Min Echo Rx Interval is not 0, the packet is discarded
- If the Your Discriminator field is 0 and the State is not DOWN or ADMINDOWN, the packet is discarded.
- If the Your Discriminator field is not 0 and no corresponding session can be found based on MI_Local_Discr[], the packet is discarded.

If the checks pass, the packet is passed to the instance of the CCCV Reception process whose MI_Local_Discr is equal to the Your Discriminator field. Packets received on the BFD-CC port from the OAM Demux process are passed on to the BFD-CC port in the CCCV Reception process, and packets received on the BFD-CV port from the OAM Demux process are passed on to the BFD-CV port in the CCCV Reception process.

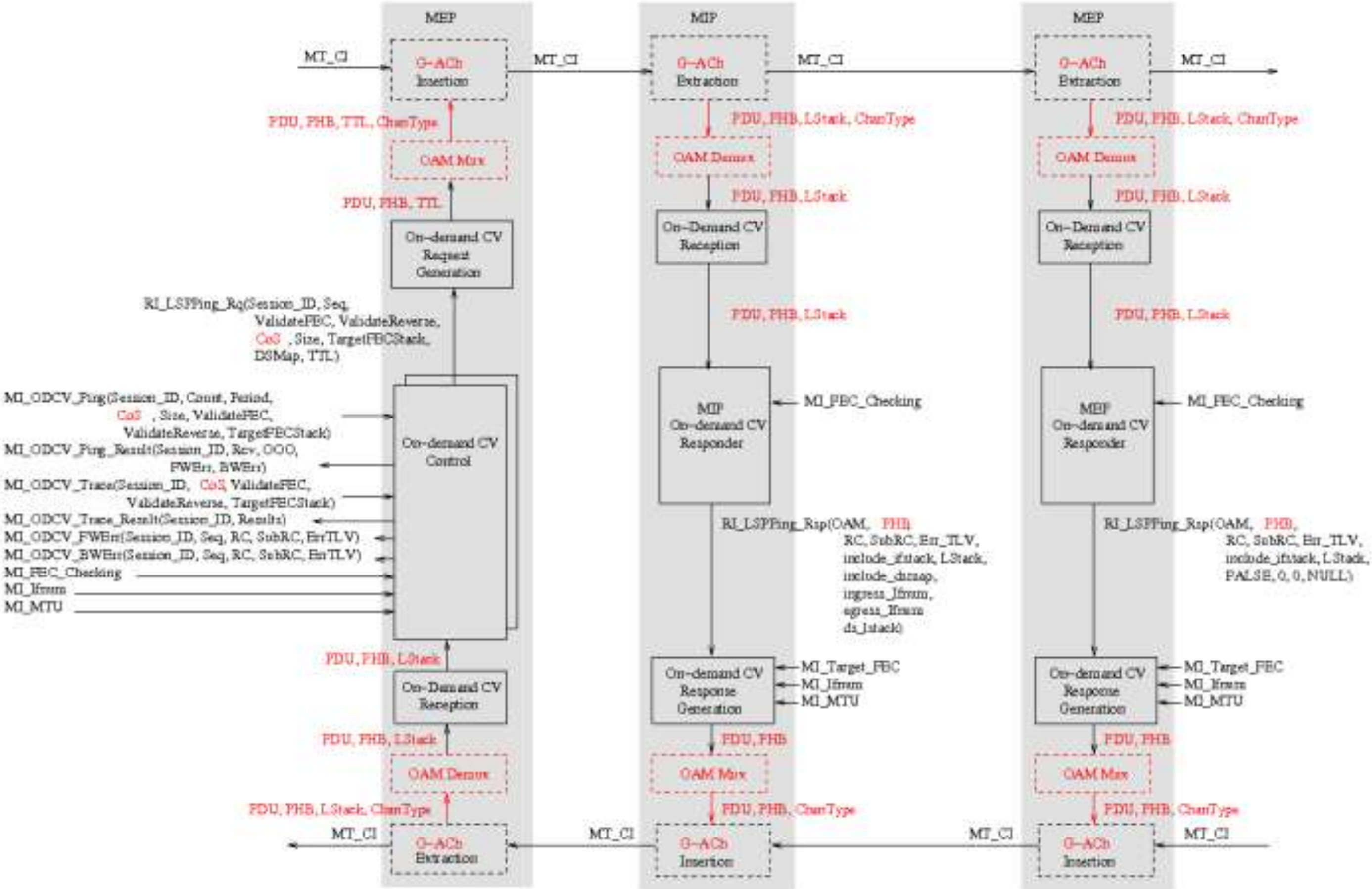
Selection of the correct CCCV Reception process when the Your Discriminator field is 0 is FFS.

8.8.2. Remote Defect Indication (RDI)

As described in rfc 6428, RDI is communicated by the BFD diagnostic field in CC messages, See clause 8.8.1~~For further study~~

~~8.8.2.~~8.8.3. On-demand CV Processes

An overview of the On-demand CV Processes is shown on Figure 8-10.



[DB1]

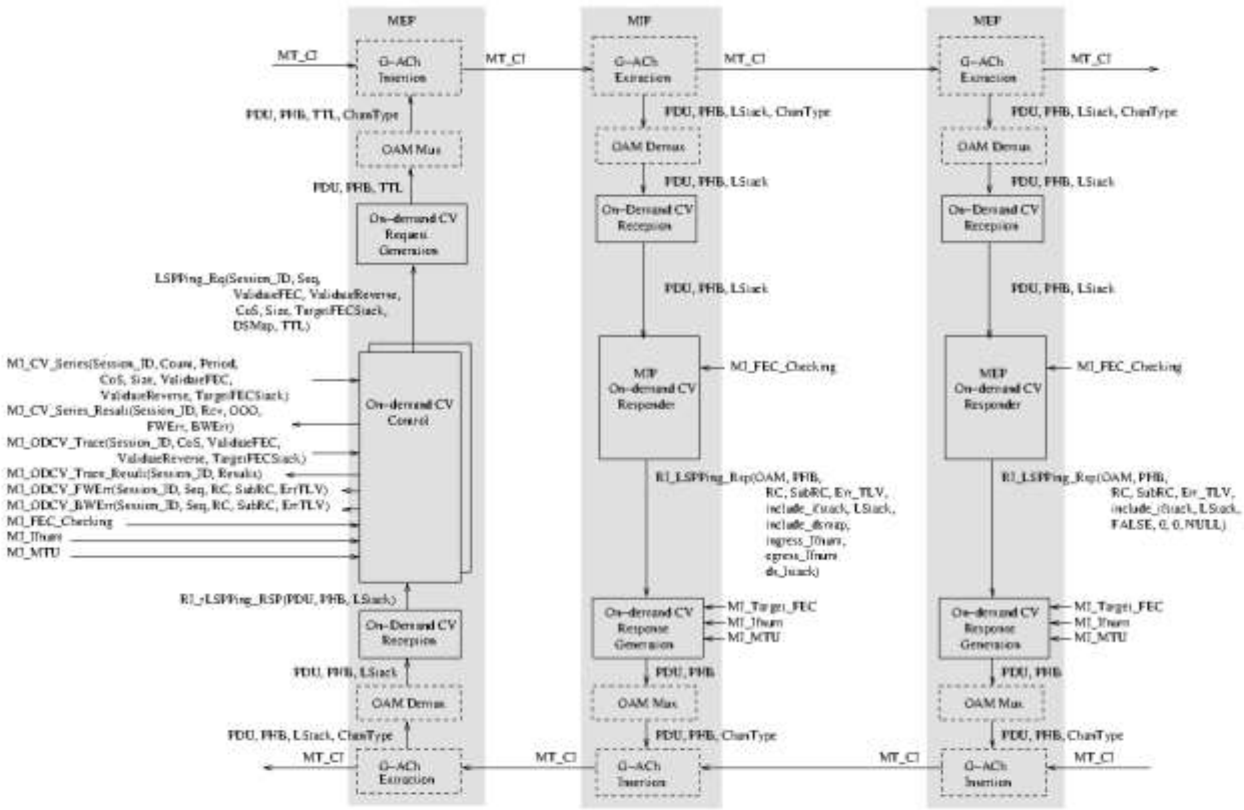


Figure 8-10/G.8121.2/Y.1381.2 - overview of the On-demand CV Processes

The On-demand CV protocol is controlled by the On-demand CV Control process. An on-demand session starts when the MI_ODCV_PingSeries() or MI_ODCV_Trace() signal is called. Multiple instances of the On-demand CV Control process can be used to run multiple on-demand CV sessions concurrently, provided each instance has a different session ID.

The On-demand CV Control process sends LSPPing Request packets via the On-Demand CV Request Generation Process, and receives LSPPing Responses via the On-Demand CV Reception process. Received responses may be checked for errors, if requested in the MI_ODCV_PingSeries() or MI_ODCV_Trace() signal.

The On-demand CV Control process reports errors in the forward direction via the MI_ODCV_FWErr() signal, and in the backward direction via the MI_ODCV_BWErr() signal. Results are reported via the MI_ODCV_PingSeries_Result() and MI_ODCV_Trace_Result() signals.

The MEP On-demand CV Responder and MIP On-demand CV Responder processes are responsible for checking received LSPPing Requests for errors, and sending responses via the On-demand CV Response Generation process.

The On-demand CV Request Generation and On-demand CV Response Generation processes generate LSPPing request and response packets in conformance with [RFC4379] and [RFC6426].

The MEP On-demand CV Responder, MIP On-demand CV Responder, and On-demand CV Control processes all perform similar steps to check received packets for errors. This checking uses the copy of the original label stack that is carried as part of the MT_CI. This common validation is described further below, followed by descriptions of each of the On-demand CV processes.

~~8.8.2.1.~~8.8.3.1. Common Validation

In the description below, label stacks and FEC stacks are denoted as arrays (Stack[]), where:

- Stack[1] is the bottom (innermost) label/FEC
- Stack[Count(Stack)] is the top (outermost) label/FEC
- Stack[0] is invalid

Count(Stack) returns the number of labels or FECs in the stack.

The validation is described by the following pseudocode. The values assigned to 'rc' are as described in [RFC4379].

```
ODCV_Validate (OAM, LStack_in[], FECStack[], MP_Type) {
    rc = 0
    sub_rc = 0
    err_TLV = NULL
    done = FALSE
    include_ifstack = FALSE
    include_dsmap = FALSE
    ldepth = 0
    LStack = LStack_in
    if (malformed(OAM)) {
        rc = 1
        done = TRUE
    } else if (OAM contains TLVs with types 4, 6, 8 or 10-32767) {
        rc = 2
        err_TLV = make_err_TLV(bad TLVs)
        done = TRUE
    } else {
        if (LStack[1] = GAL) {
            remove_GAL_from_LStack()
        }
        ldepth = count(LStack)
        while (!done && ldepth> 0) {
            if (!label_known(LStack[ldepth])) {
                rc = 11
                sub_rc = ldepth
                done = TRUE
            }
            ldepth--
        }
    }
    if (MP_Type = MEP) {
```

```
        if (!done) {
            FECdepth = 1
            L = IMPLICIT_NULL
            rc = 3
            sub_rc = 1
            if (DSMAP(OAM) != NULL && Ingress_Ifnum(DSMAP(OAM)) != 0) {
                if (DownstreamLabels(DSMAP(OAM)) != LStack) {
                    rc = 5
                    include_ifstack = TRUE
                    done = TRUE
                }
            }
        }
        while (!done) {
            (FECstatus, FECrc) = checkFEC(FECStack[FECdepth], L)
            rc = FECrc
            sub_rc = FECdepth
            if (FECstatus = 1) {
                done = TRUE
            } else {
                FECdepth++
                if (FECdepth > count(FECStack)) {
                    done = TRUE
                }
            }
            if (!done) {
                if (FECstatus = 0) {
                    ldepth++
                    if (ldepth > count(LStack)) {
                        done = TRUE
                    } else {
                        L = LStack[ldepth]
                    }
                }
            }
        }
    } else {
        // MP_Type = MIP
        if (!done) {
            rc = 8
```

```
        sub_rc = 1
        if (DSMAP(OAM) != NULL) {
            if (Ingress_Ifnum(DSMAP(OAM)) = 0) {
                rc = 6
                include_ifstack = TRUE
            } else {
                if (DownstreamLabels(DSMAP(OAM)) != LStack) {
                    rc = 5
                    include_ifstack = TRUE
                    done = TRUE
                }
            }
        }
    }
    if (!done) {
        Egress_Ifnum = get_egress_interface()
        if (Egress_Ifnum = 0) {
            rc = 9
            done = TRUE
        }
    }
    if (!done) {
        if (DSMAP(OAM) != NULL) {
            include_dsmap = TRUE
        } else {
            done = TRUE
        }
    }
    if (!done) {
        if (V(OAM) == 0 && MI_FEC_Checking = 0) {
            done = TRUE
        }
    }
    if (!done) {
        FECdepth = 0
        i = 1
        while (i > 0) {
            FECdepth++
            if (DownstreamLabels(DSMAP(OAM))[FECdepth] != IMPLICIT_NULL) {
               i--
```

```
                }
            }
            if (count(FECStack) >= FECdepth) {
                (FECstatus, FECrc) = checkFEC(FECStack[FECdepth], LStack[1])
                if (FECstatus = 2) {
                    rc = 10
                } else if (FECstatus = 1) {
                    rc = FECrc
                    sub_rc = FECdepth
                }
            }
        }
    }
    return(rc, sub_rc, err_TLV, include_ifstack, include_dsmap)
}
```

The utility functions used in the pseudocode above are described below:

- malformed(OAM) checks that the packet is in accordance with the format described in [RFC4379] and [RFC6426]. It also checks that:
    - If the packet is a request, it contains a Target FEC Stack TLV
    - If the packet is a reply and the R flag is set, it contains a Reverse Target FEC Stack TLV
    - The Target FEC Stack or Reverse Target FEC Stack TLVs contain only sub-types 'Static LSP', 'Static Pseudowire' and 'Nil FEC'. Use of other subtypes are FFS.
    - If the packet contains a Downstream Mapping TLV, the address type is 'Non-IP'. Use of other address types is FFS.
- make_err_TLV(TLVs) creates an 'Errored TLVs' TLV according to [RFC4379] and copies the bad TLVs into it.
- remove_GAL_from_LStack removes the GAL from the bottom of the label stack, so that LStack[1] now refers to the label that immediately preceded the GAL.
- label_known(Label) checks whether the Label value is known and can be processed.
- checkFEC(FEC, Label) implements the FEC checking procedure described in [RFC4379] section 4.4.1.
- get_egress_interface() returns MI_Ifnum if this is the egress interface, otherwise it uses forwarding information to find the egress interface and returns its interface number, or 0 if no egress interface was found or it is not MPLS-enabled.

~~8.8.2.2.~~8.8.3.2. On-demand CV Control process

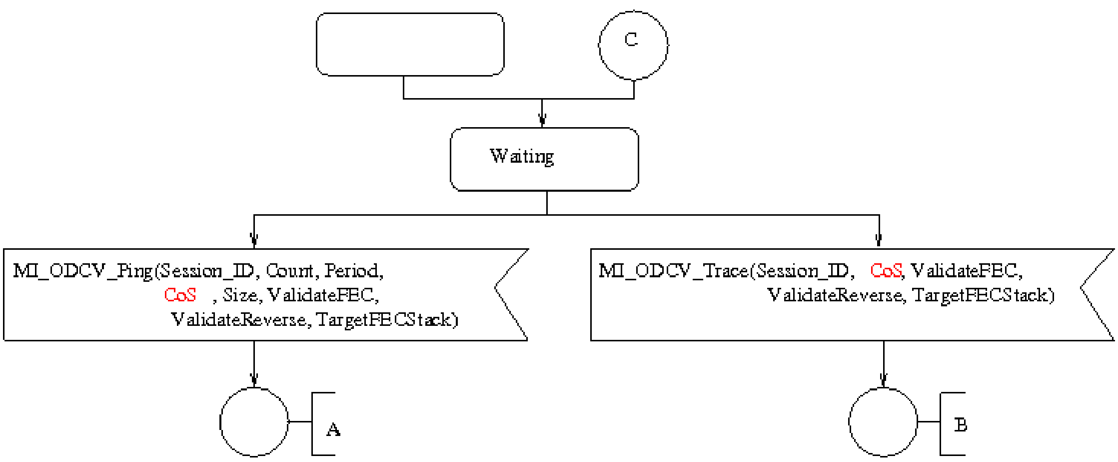
The On-demand CV Control process operates the LSPPing on-demand CV protocol. An LSPPing session is started by the MI_~~OD~~CV_~~Ping~~Series() or MI_ODCV_Trace() signals. In either case, a

Session ID is supplied; multiple instances of the On-demand CV Control process can be created, provided each has a unique Session ID.

The Target FEC Stack to be checked by the peer device is specified in the MI_ODCV_PingSeries() or MI_ODCV_Trace() signal. Other mechanisms for deriving the Target FEC Stack, for example if dynamic signalling protocols are in use, are FFS. The Target FEC Stack passed in the MI_ODCV_PingSeries() or MI_ODCV_Trace() signals must only contain FECs with subtypes 'Static LSP', 'Static Pseudowire' or 'Nil FEC'.

Results are reported by the On-demand CV Control process using the MI_ODCV_PingSeries_Result() or MI_ODCV_Trace_Result() signals when the session ends. In addition, any errors detected while the session is running are reported by using the MI_ODCV_FWErr() signal (for errors in the Control-to-Responder direction) or MI_ODCV_BWErr() signal (for errors on the Responder-to-Control direction). Note that errors in the Responder-to-Control direction are only detected if ValidateReverse is set to TRUE in the MI_ODCV_PingSeries() or MI_ODCV_Trace() signal.

The behaviour of the On-demand CV Control process is shown in the figures below (Figure 8-11, 12, 13, and 14). In PingRunning state, the process sends LSPPing Requests periodically, and handles any received replies by counting them and checking for any errors. In TraceRunning state, an initial LSPPing Request is sent with TTL 1, so that it is intercepted by the first MIP (or MEP) reached. When a response is received, it is first checked for any errors. Then, if the response was from a MIP (ie it contains a DSMap TLV), the TTL is incremented and a new LSPPing request is sent. Incrementing the TTL ensures the request is intercepted by the next MIP (or MEP). If no response is received, 3 attempts are made to resend the request, before giving up and reporting any results collected so far.



[DB2]

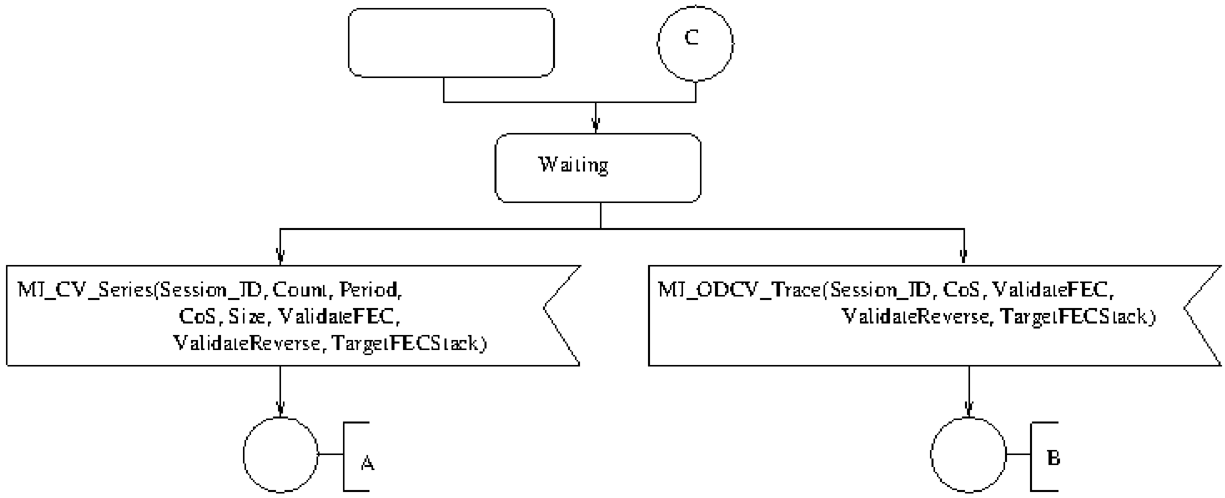
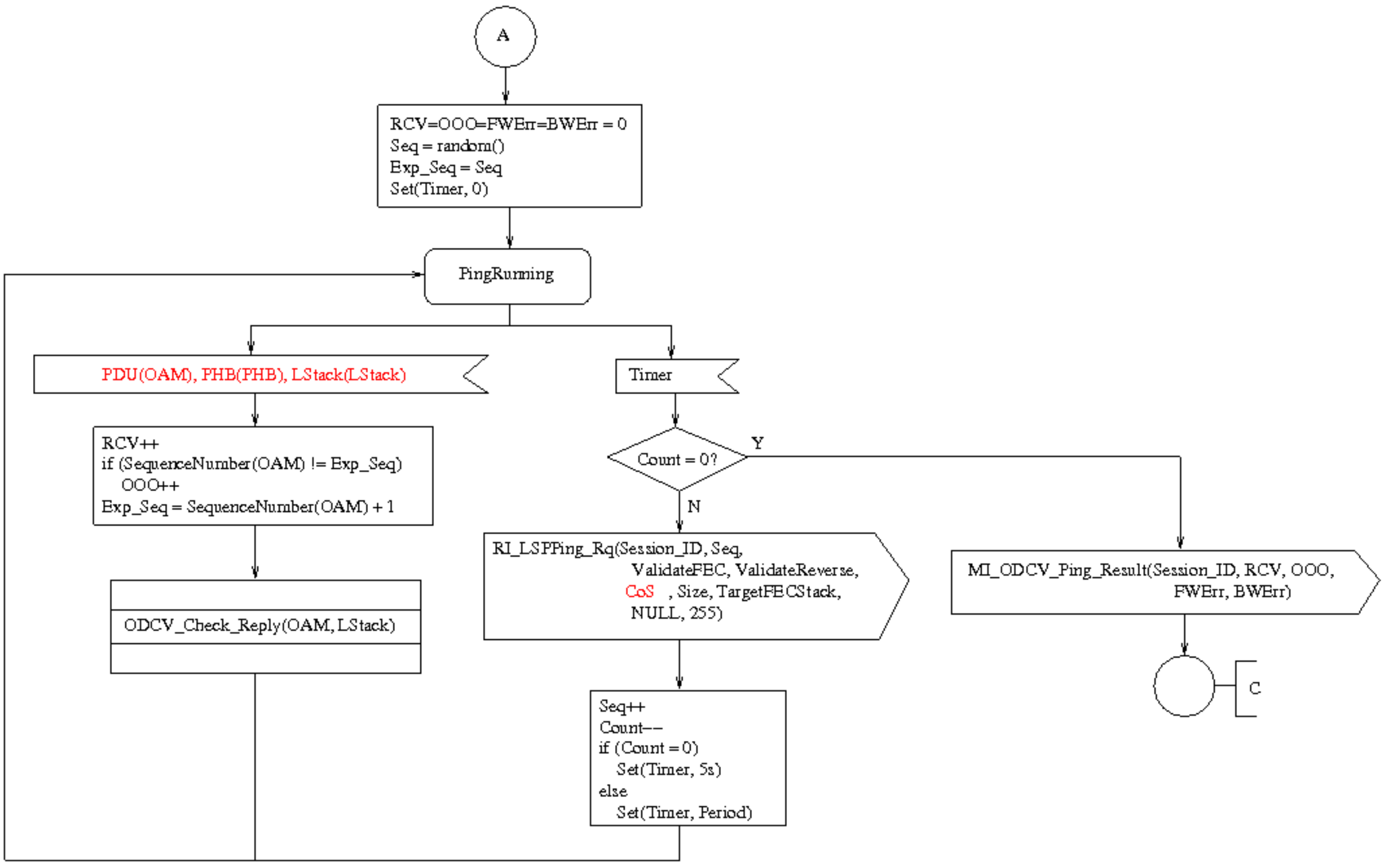


Figure 8-11/G.8121.2/Y.1381.2 - On-demand CV Control Process



[DB3]

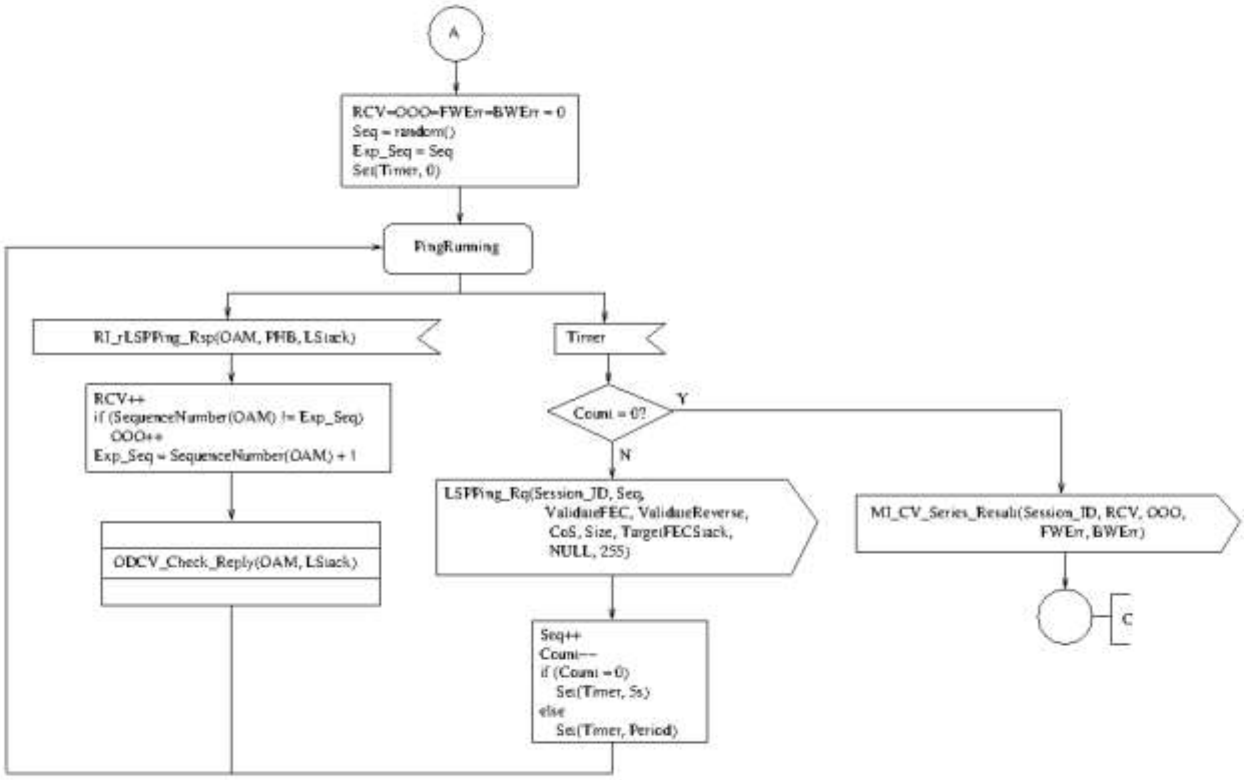
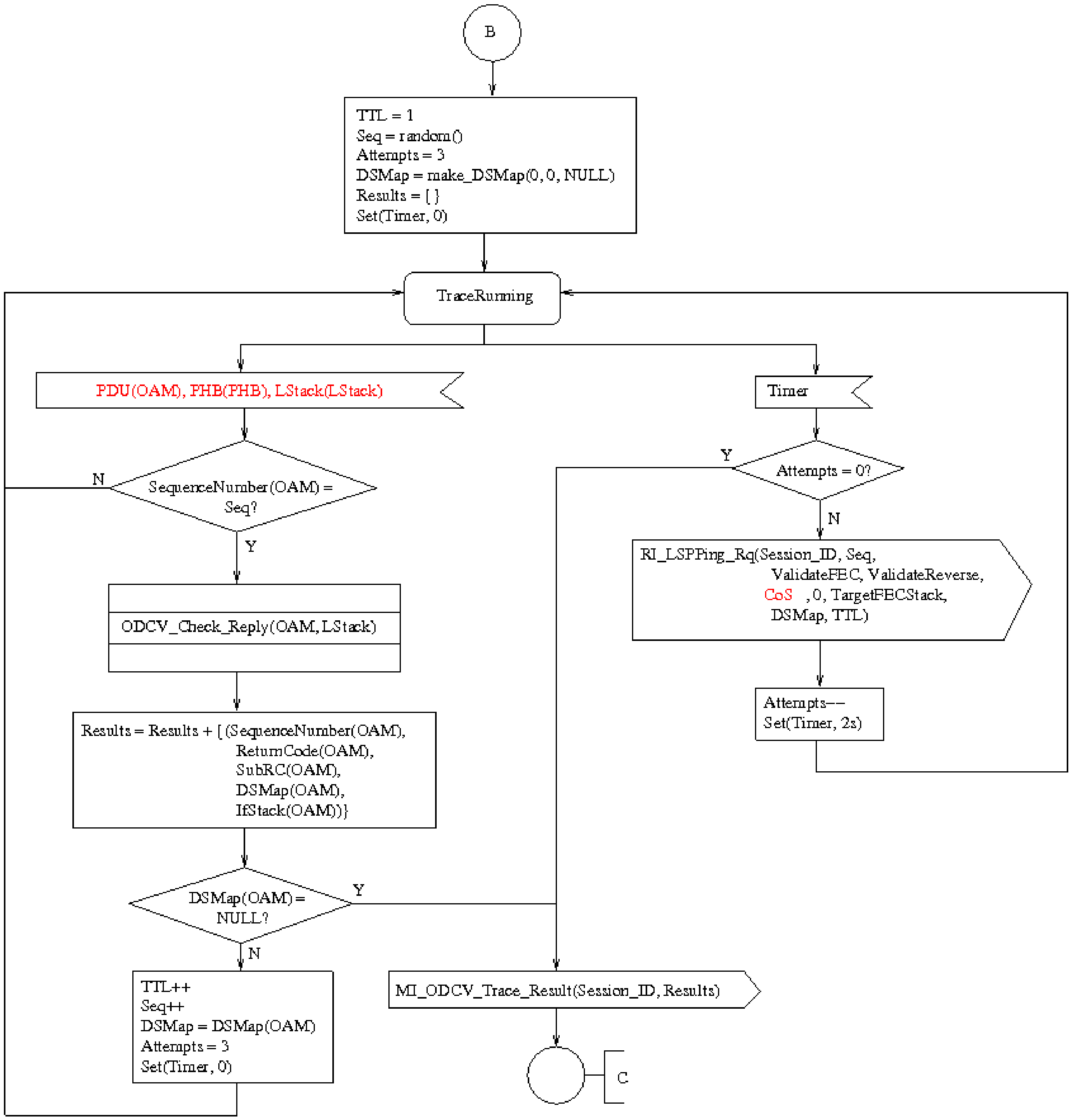


Figure 8-12/G.8121.2/Y.1381.2 - On-demand CV Control Process


B
TTL = 1
Seq = random()
Attempts = 3
DSMap = make_DSMap(0, 0, NULL)
Results = [ ]
Set(Timer, 0)
TraceRunning
PDU(OAM), PHB(PHB), LStack(LStack)
Timer
N
SequenceNumber(OAM) = Seq?
Y
Attempts = 0?
N
ODCV_Check_Reply(OAM, LStack)
RI_LSPPing_Rq(Session_ID, Seq, ValidateFEC, ValidateReverse, CoS , 0, TargetFECStack, DSMap, TTL)
Attempts--
Set(Timer, 2s)
Results = Results + [(SequenceNumber(OAM), ReturnCode(OAM), SubRC(OAM), DSMap(OAM), IfStack(OAM))}
DSMap(OAM) = NULL?
TTL++
Seq++
DSMap = DSMap(OAM)
Attempts = 3
Set(Timer, 0)
MI_ODCV_Trace_Result(Session_ID, Results)
C

[DB4]

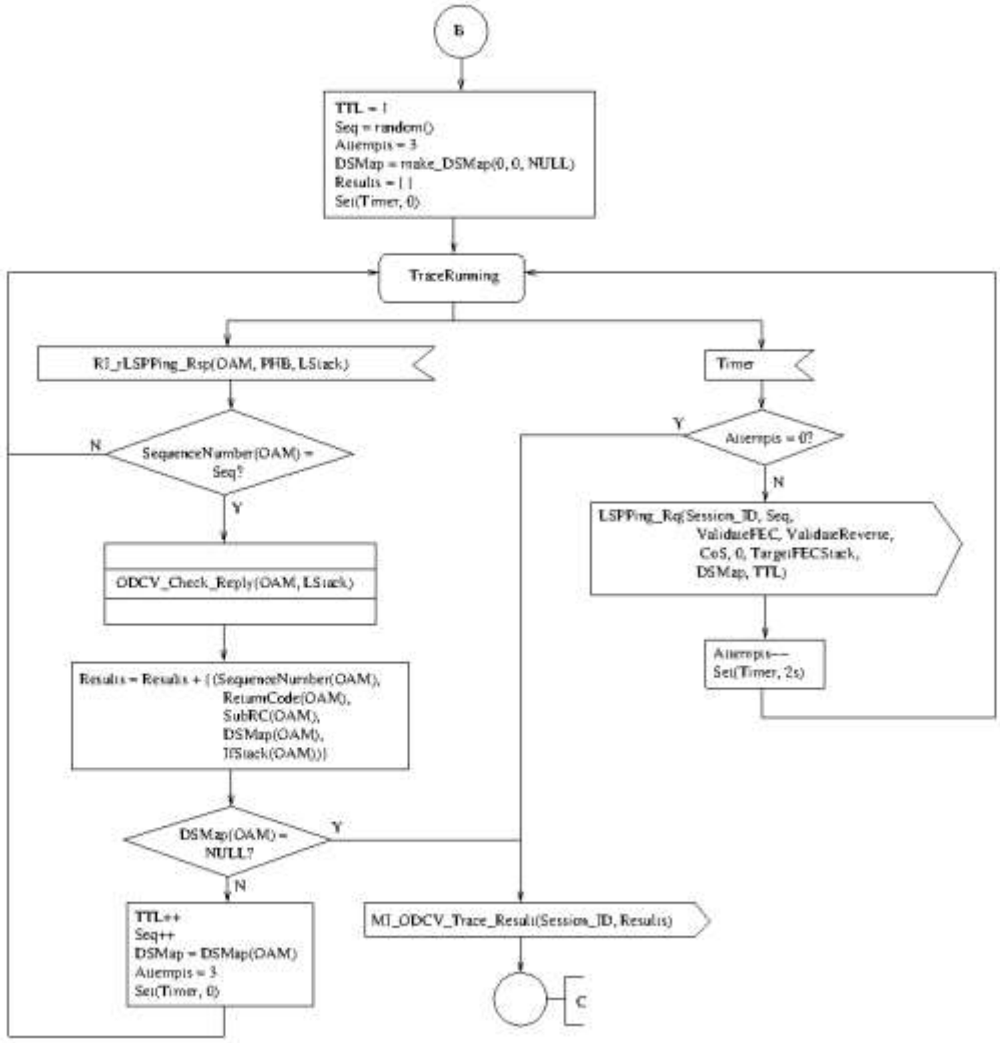


Figure 8-13/G.8121.2/Y.1381.2 - On-demand CV Control Process

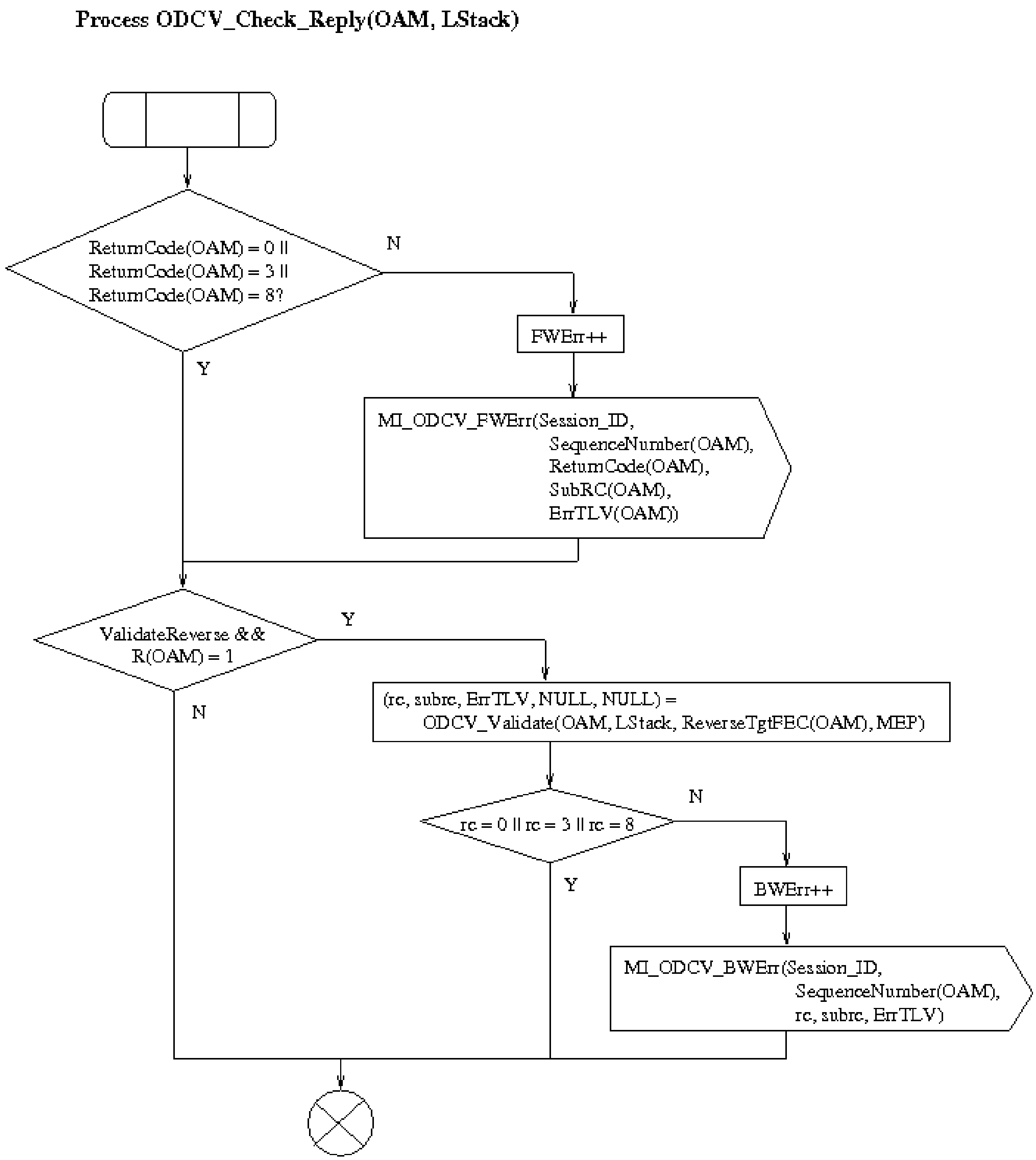


Figure 8-~~x~~14/G.8121.2/Y.1381.2 - On-demand CV Control Process

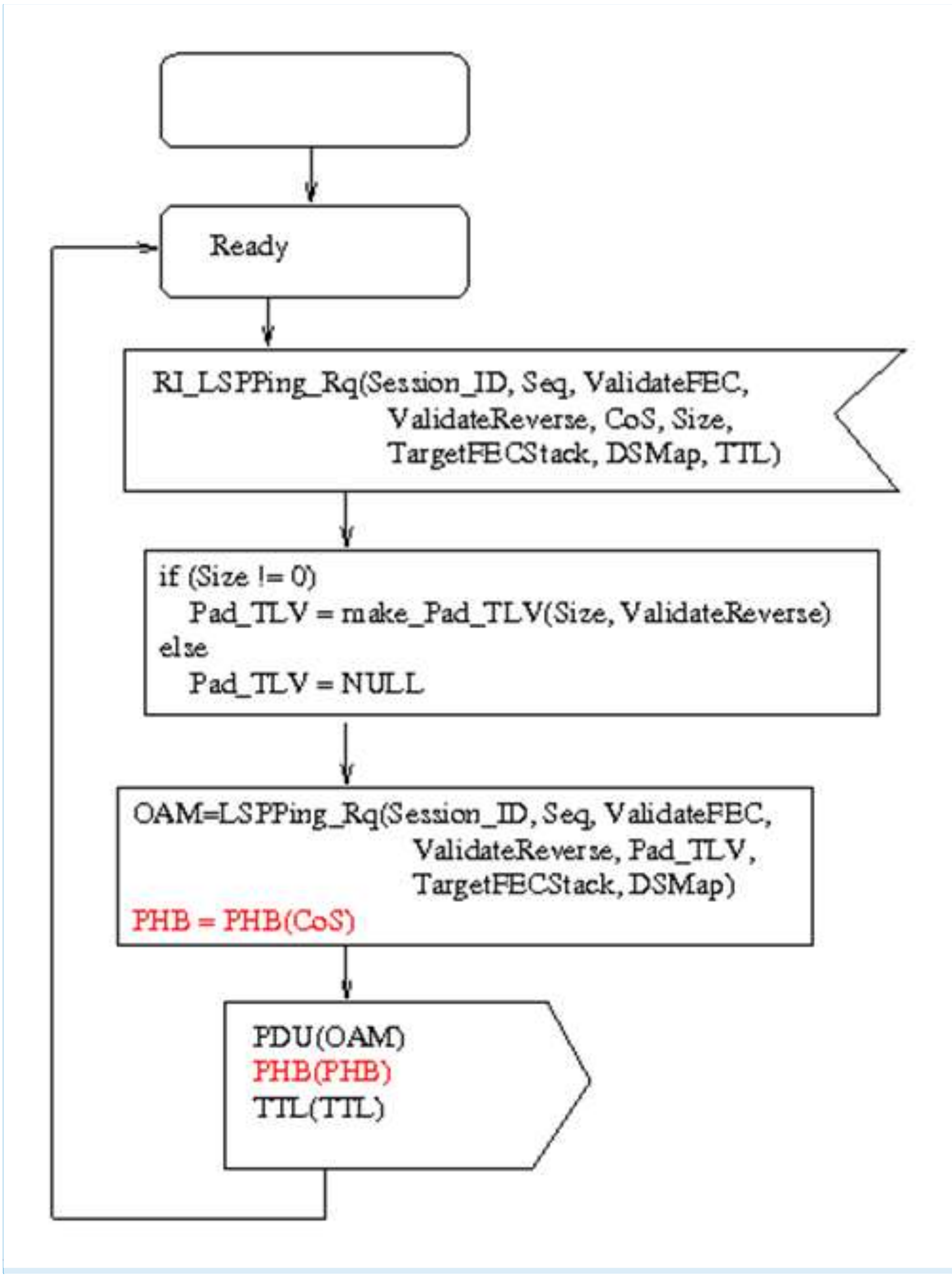
The make_DSMap(ingress_ifnum, egress_ifnum, ds_lstack) function creates a Downstream Mapping TLV according to [RFC4379] and [RFC6426]. The fields are filled in as follows:

- MTU: Set to MI_MTU

- Address Type: set to 5 (Non IP). Use of other address types is FFS.
- DS Flags: The I flag is set to 1, all other flags are set to 0.
- Ingress Ifnum: set to ingress_ifnum
- Egress Ifnum: set to egress_ifnum
- Multipath Type: set to 0 (no Multipath)
- Depth Limit: set to 0
- Multipath Length: set to 0
- Downstream Labels: derived from ds_lstack as described in [RFC4379]. The protocol is set to 1 (Static). Use of other values isare FFS.

8.8.2.3.8.8.3.3. On-demand CV Request Generation process

The On-demand CV Request Generation process is shown in Figure 8-15:


Ready
RI_LSPPing_Rq(Session_ID, Seq, ValidateFEC,
ValidateReverse, CoS, Size,
TargetFECStack, DSMap, TTL)
if (Size != 0)
Pad_TLV = make_Pad_TLV(Size, ValidateReverse)
else
Pad_TLV = NULL
OAM=LSPPing_Rq(Session_ID, Seq, ValidateFEC,
ValidateReverse, Pad_TLV,
TargetFECStack, DSMap)
PHB = PHB(CoS)
PDU(OAM)
PHB(PHB)
TTL(TTL)

[DB5]

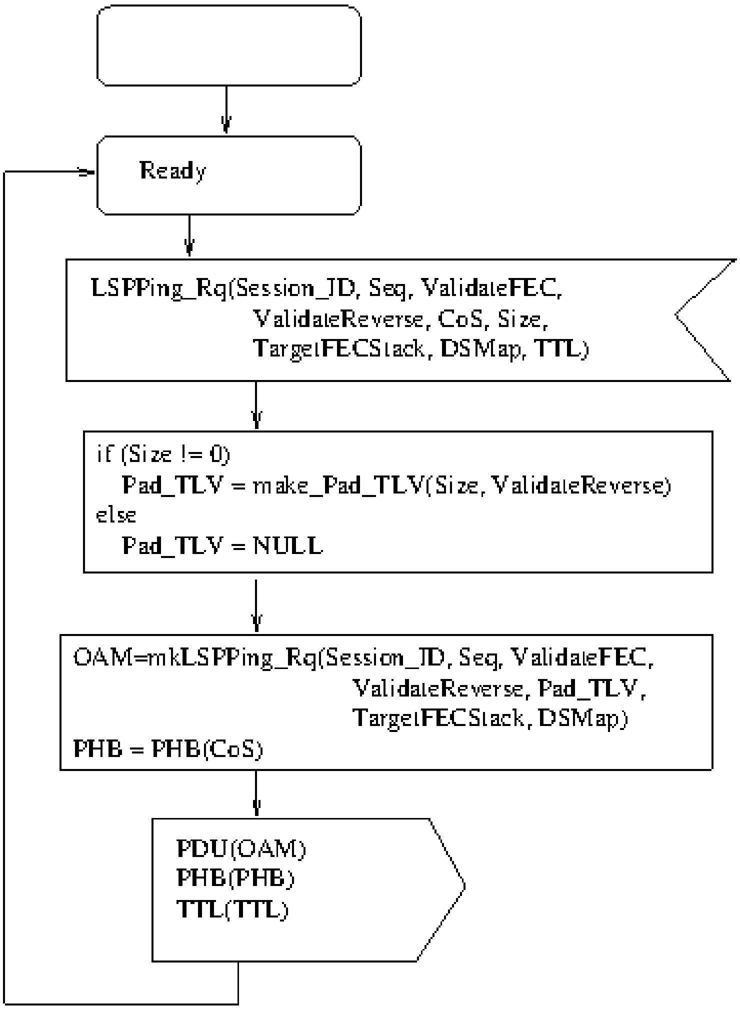


Figure 8-15/G.8121.2/Y.1381.2 On-demand CV Request Generation Process

The make_Pad_TLV(Size) function creates a Pad TLV in accordance with [RFC4379]. The Length field is set to Size. The first octet of the value field is set to 2 (Copy Pad TLV) if ValiadateReverse is FALSE, and 1 (Drop Pad TLV) if ValidateReverse is TRUE.

The PHB(CoS) function returns the PHB with the lowest drop precedence within the Class of Service defined by the CoS input parameter.

Note: Size is only non-zero in Ping mode, when no DSMap TLV is included. In this case, the responder will not add any additional TLVs (eg an interface and label stack TLV) to the reply unless the 'R' (ValidateReverse) flag is set, and so the Pad TLV can be safely copied into the reply.

The mkLSPPing_Rq function creates an LSPPing Echo Request packet in accordance with [RFC4379] and [RFC6426]. The fields are filled in as follows:

- Version Number: set to 1
- Global Flags: if ValidateFEC is TRUE, the V flag is set to 1; if ValidateReverse is TRUE, the R flag is set to 1; all other flags are set to 0.
- Message Type: set to MPLS Echo Request

- Reply Mode: set to 4 (reply via application control channel)
- Return Code: set to 0
- Return Subcode: set to 0
- Sender's Handle: set to the value of Session_ID
- Sequence Number: set to the value of Seq
- Timestamp Sent: set to LocalTime.
- Timestamp Received: set to 0.

The following TLVs are added:

- A Target FEC Stack TLV is added containing the contents of TargetFECStack.
- If Pad_TLV is not NULL, a Pad TLV is added containing the contents of Pad_TLV.
- If DSMap is not NULL, a Downstream Mapping TLV is added containing the contents of DSMap.

~~8.8.2.4.~~8.8.3.4. On-demand CV Reception process

The On-demand CV Reception process demultiplexes received LSPPing packets (formed of OAM, PHB, LStack signals) as follows:

- If the Message Type is MPLS Echo Request, the ~~packet~~ received OAM, PHB, and LStack signals ~~is~~ are passed to the MIP On-demand CV Responder or MEP On-demand CV Responder process as appropriate
- Otherwise, if this is a MIP the packet is discarded.
- If this is a MEP and the Message Type is MPLS Echo Reply, the On-demand CV Reception process passes the received OAM, PHB, and LStack signals ~~packet~~ to the instance of the On-demand CV Control process whose Session ID is equal to the "Sender's handle" in the received packet, via RI_rLSPPing_Rsp(OAM, PHB, LStack). If there is no such instance of the On-demand CV Control process, the packet is discarded.

~~8.8.2.5.~~8.8.3.5. MIP On-demand CV Responder process

The MIP On-demand CV Responder process is described in Figure 8-16~~the figure below~~.

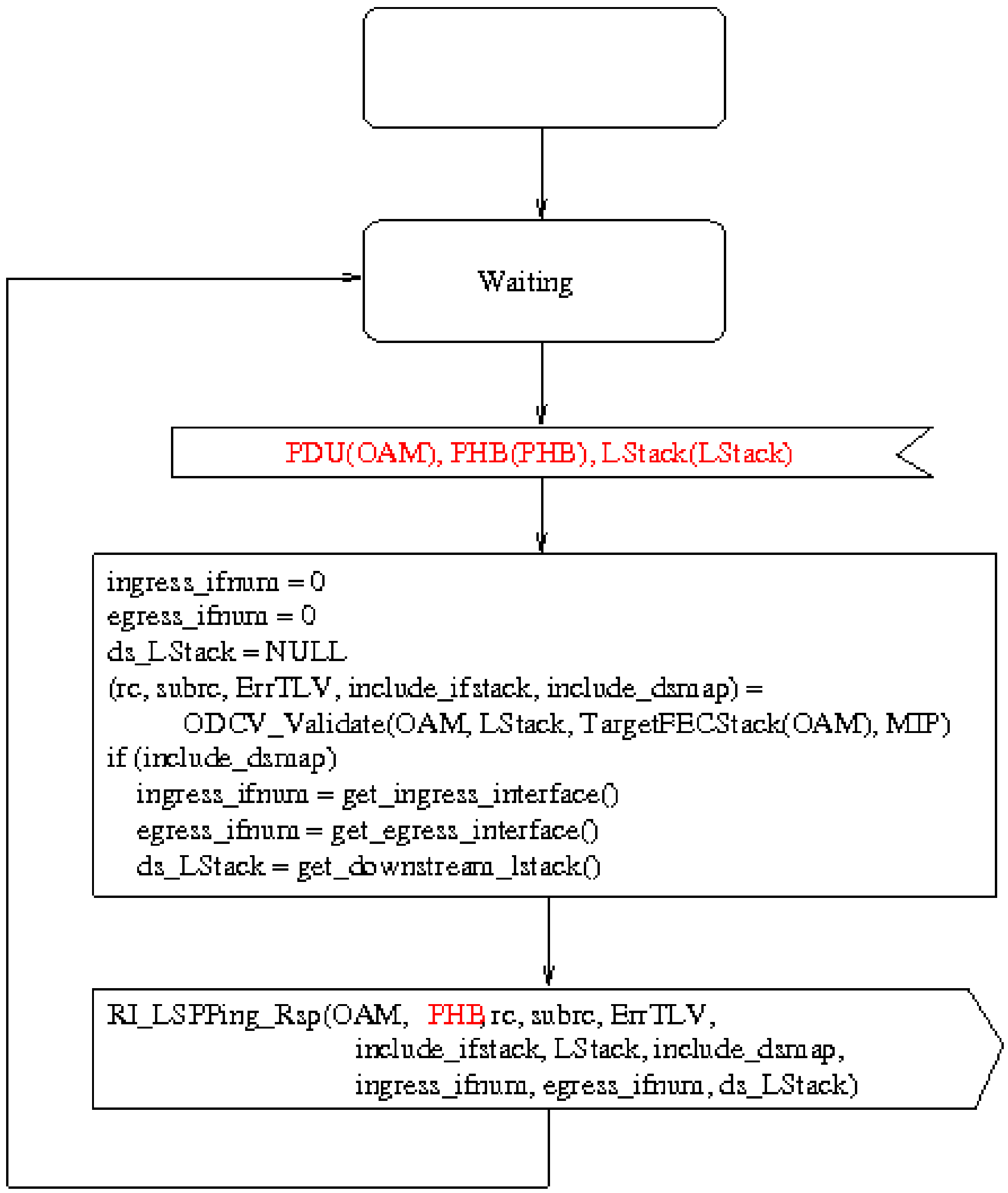


Figure 8-~~x~~16/G.8121.2/Y.1381.2 MIP On-demand CV Responder Process

The get_egress_interface() function is described in section 8.8.2.1 above.

The get_ingress_interface() function returns MI_Ifnum if this is the ingress interface, otherwise it returns the interface number of the interface where the packet arrived.

The get_downsteam_lstack() returns the label stack that would be attached to the packet if it were to be forwarded out of the egress interface, derived as described in [RFC4379].

~~8.8.2.6.~~8.8.3.6. MEP On-demand CV Responder process

The MEP On-demand CV Responder process is described in Figure 8-17~~the figure below~~.

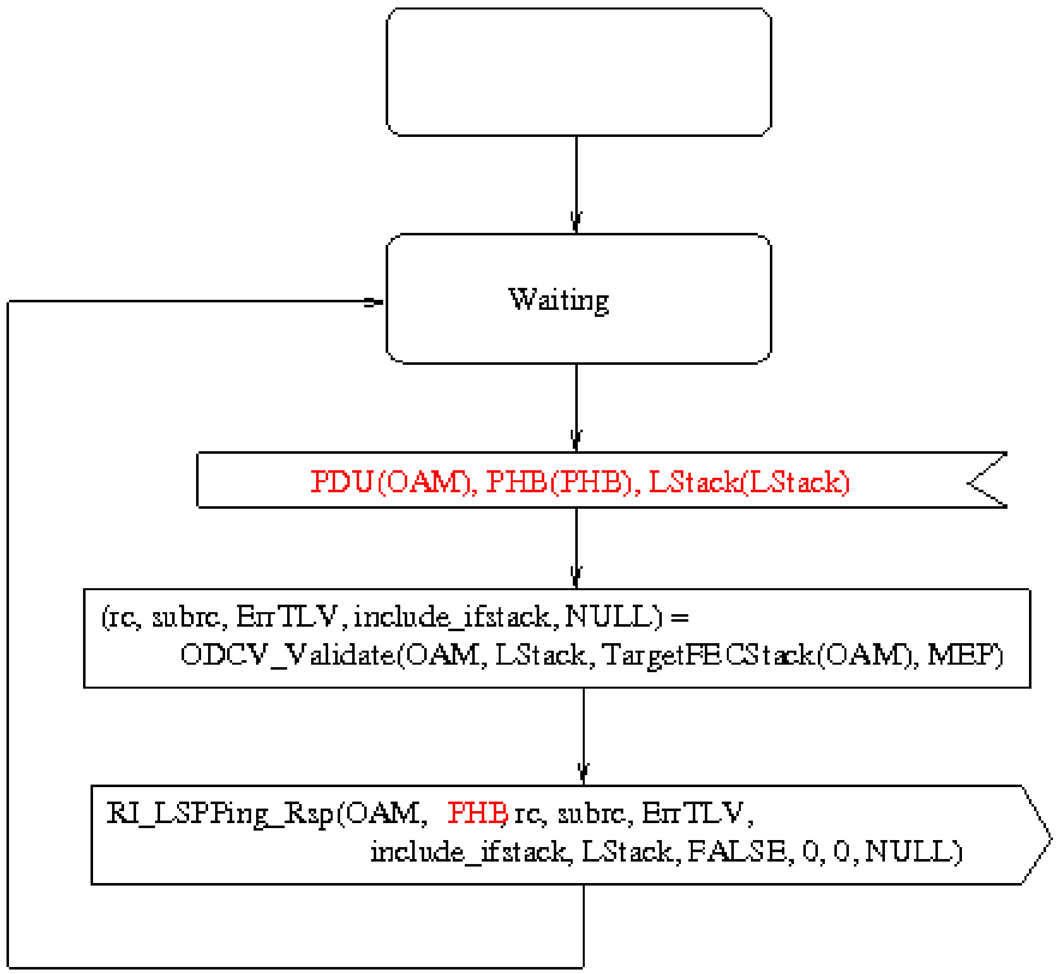


Figure 8-~~x~~17/G.8121.2/Y.1381.2 MEP On-demand CV Responder Process

~~8.8.2.7.~~8.8.3.7. On-demand CV Response Generation process

The On-demand CV Response Generation process is shown in Figure 8-18~~the following figure~~:

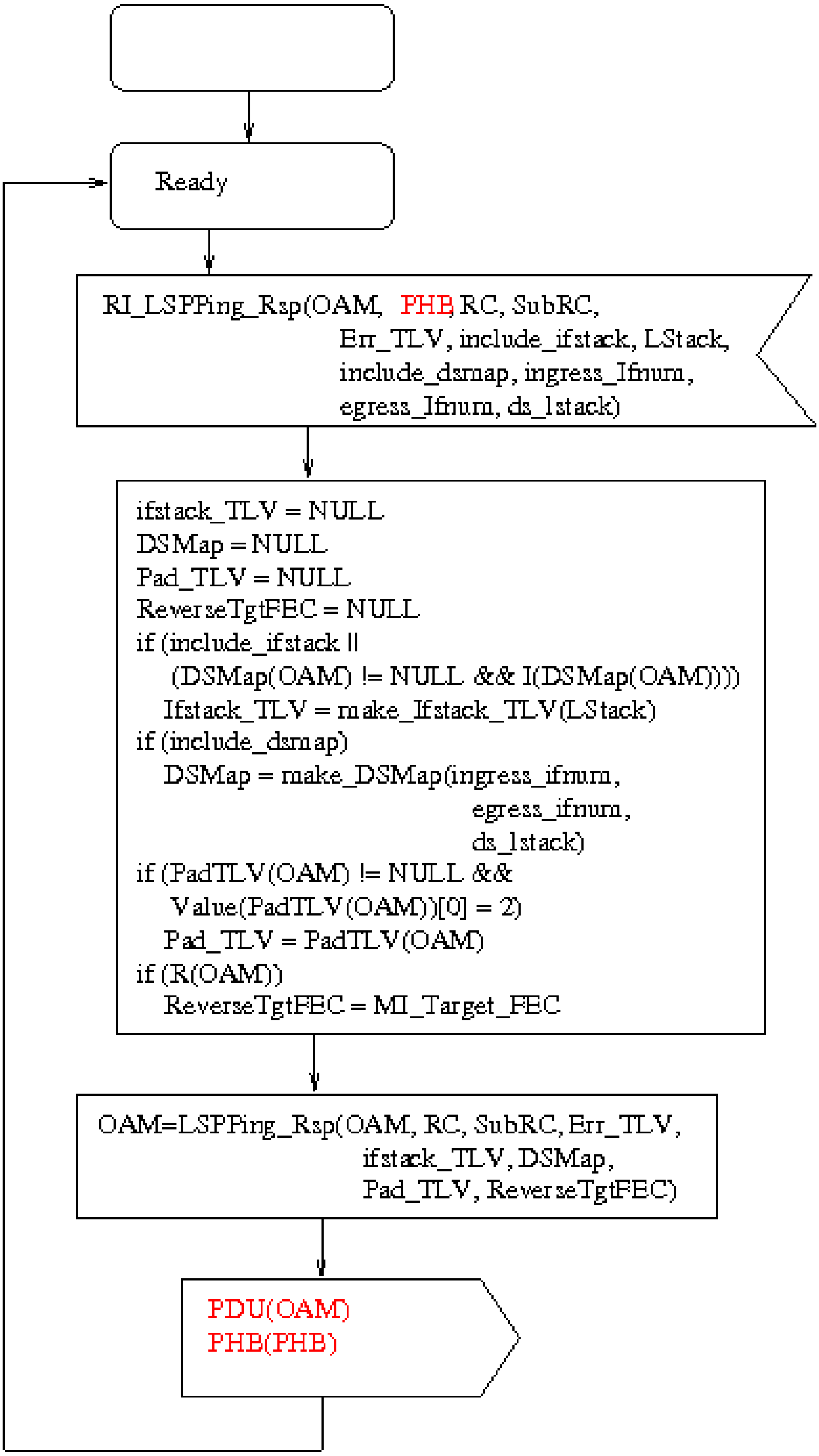


Figure 8-x18/G.8121.2/Y.1381.2 On-demand CV Response Generation Process

The make_Ifstack_TLV(LStack) function creates an Interface and Label Stack TLV according to [RFC4379] and [RFC6426]. The fields are filled in as follows:

- Address Type: set to IPv4 Unnumbered
- IP Address: set to 0
- Interface: set to MI_Ifnum
- Label Stack: Copied from LStack.

Use of other values in the Interface and Label Stack TLV is FFS.

[Editor's note: this use of MI_Ifnum should align with [draft-ietf-mpls-tp-mib-management-overview]].

The make_DSMap(ingress_ifnum, egress_ifnum, ds_lstack) function is described in section 8.8.2.2 above.

The LSPPing_Rsp function creates an LSPPing Echo Reply packet in accordance with [RFC4379] and [RFC6426]. The fields are filled in as follows:

- Version Number: set to 1.
- Global Flags: copied from the received Echo Request.
- Message Type: set to MPLS Echo Reply.
- Reply Mode: set to 0 (do not reply).
- Return Code: set to RC.
- Return Subcode: set to SubRC.
- Sender's Handle: copied from the received Echo Request.
- Sequence Number: copied from the received Echo Request.
- Timestamp Sent: copied from the received Echo Request.
- Timestamp Received: set to LocalTime.

If reverse FEC checking was requested in the LSPPing request (ie, the R flag was set), a Reverse Target FEC Stack is created based on MI_Target_FEC. Other mechanisms for deriving the FEC stack, for example if dynamic signalling protocols are in use, are FFS.

The following TLVs are added:

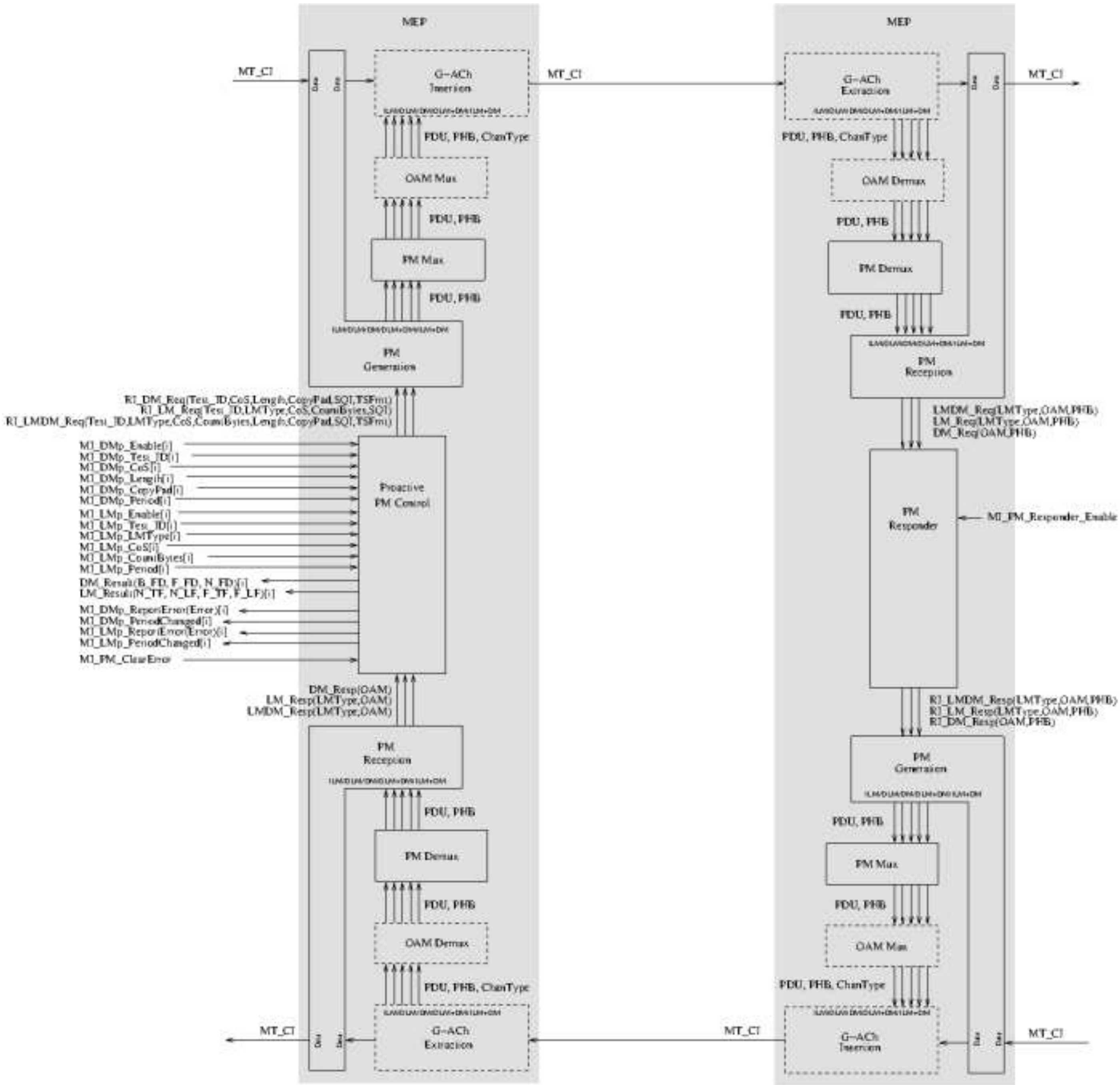
- The TargetFECStack TLV is copied from the received packet.
- If Err_TLV is not NULL, an Errored TLVs TLV is added containing the contents of Err_TLV
- If Ifstack_TLV is not NULL, an Interface and Label Stack TLV is added containing the contents of Ifstack_TLV
- If DSMap is not NULL, a Downstream Mapping TLV is added containing the contents of DSMap.
- If Pad_TLV is not NULL, a Pad TLV is added containing the contents of Pad_TLV.

- If ReverseTgtFEC is not NULL, a Reverse-path Target FEC Stack TLV is added containing the contents of ReverseTgtFEC.

~~8.8.3.~~8.8.4. Proactive Packet Loss Measurement (LMp)

As described in clauses 7.2.2.1 and 8.6 to 8.8 of [ITU-T G.8113.2], loss and delay measurements may be combined. The format for the combined measurement, refered to here as LMDM, is described in section 3.3 of [RFC 6374]. In addition, the same LM and DM protocols can be used for both proactive and on-demand measurement.

An overview of the performance monitoring processes for a single proactive PM session is shown in figure 8-18a. The same processes are used for LM, DM or LMDM.



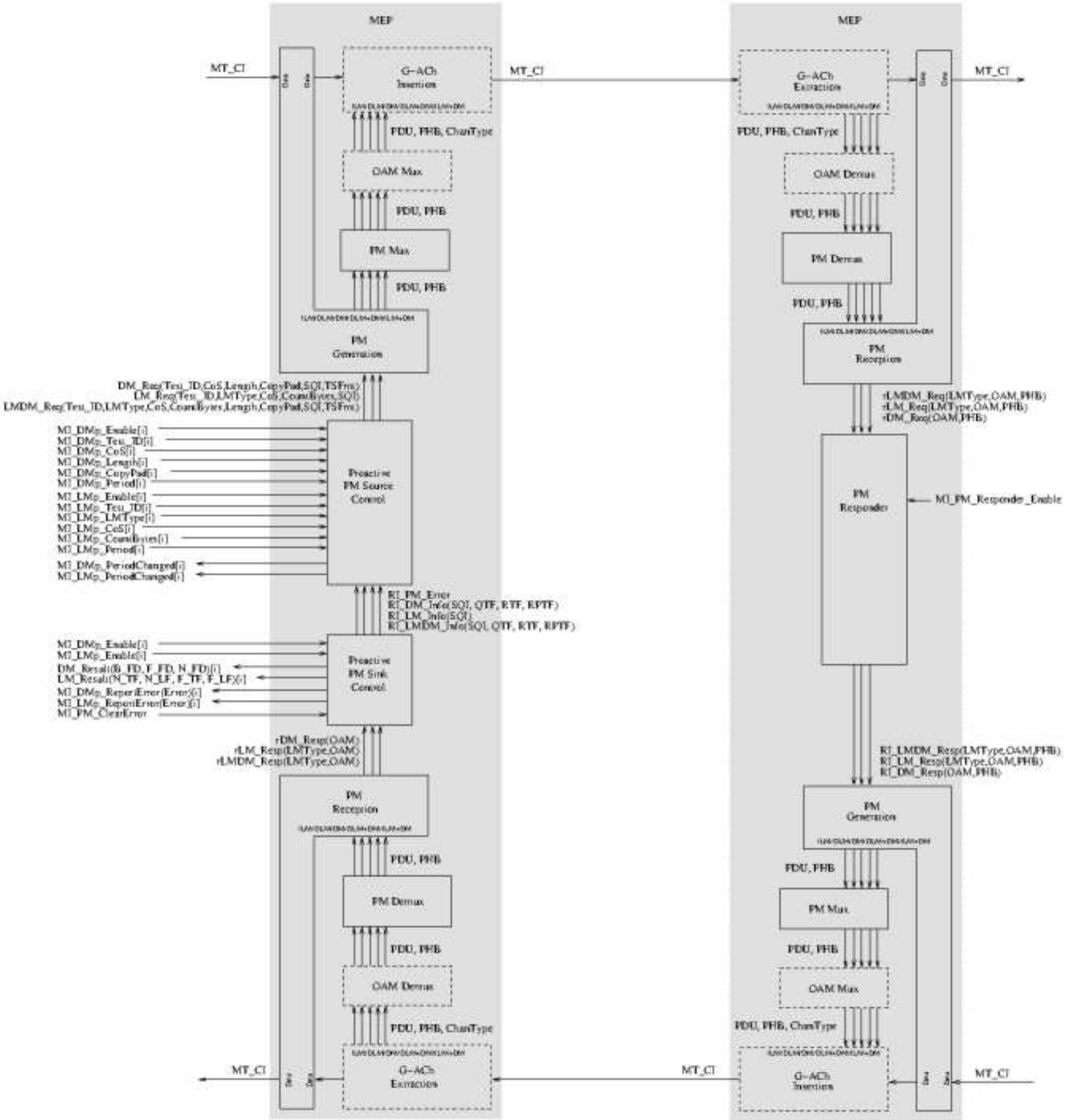


Figure 8-18a/G.8121.2/Y.1381.2 Proactive PM processes

The Proactive PM Source control process controls the session, including scheduling request packets;~~, and~~the Proactive PM Sink control process handles processing responses to calculate performance metrics.

The PM generation process generates requests and responses for the five different types of PM PDUs: ILM, DLM, DM, ILM+DM and DLM+DM. It also counts data traffic (including test packets) and is responsible for writing the counters and/or timestamps into the outgoing PM PDUs.

The location of the counter part is shown on the figure above for illustration only; the exact set of packets to be counted is implementation-specific, as described in [RFC6374].

The PM reception process handles received requests and responses. Like the PM generation process, it counts the appropriate packets and writes the counters and/or timestamps into the received PM PDUs. Again, the location of the counter part is shown on the figure above for illustration only; the exact set of packets to be counted is implementation-specific, as described in [RFC6374].

The PM responder is responsible for replying to received PM request packets.

Multiple PM sessions can be used simultaneously, by instantiating multiple instances of the PM Source control, PM Sink Control, PM reception, PM generation and PM responder processes. Each instance of these processes supports a single PM session. Each PM session (proactive or on-demand) must have a unique test ID. For each test ID, a pair of~~the~~ control processes (ie, source and sink) ~~is~~are associated with a corresponding instance of the PM reception and PM generation processes. Similarly, the responder process for a given session is associated with a corresponding instance of the PM reception and PM generation processes. The PM Mux process multiplexes PM packets for different sessions, while the PM Demux process demultiplexes them based on the Test ID (Session ID) and R (response) flag.

Note therefore that a given instance of the PM reception process is associated with exactly one other process to which it passes received packets. Depending on how it is instantiated, this could be the proactive PM sink control process, the on-demand PM control process (see 8.8.5), or the PM responder process.

8.8.4.1 Proactive PM Source control process for LM

The proactive PM Source control process includes LM, DM and LMDM. Each instance of the process operates a single proactive PM session. Multiple sessions can be supported by instantiating multiple instances of the process, along with corresponding instances of the PM Sink Control, PM generation and PM reception processes.

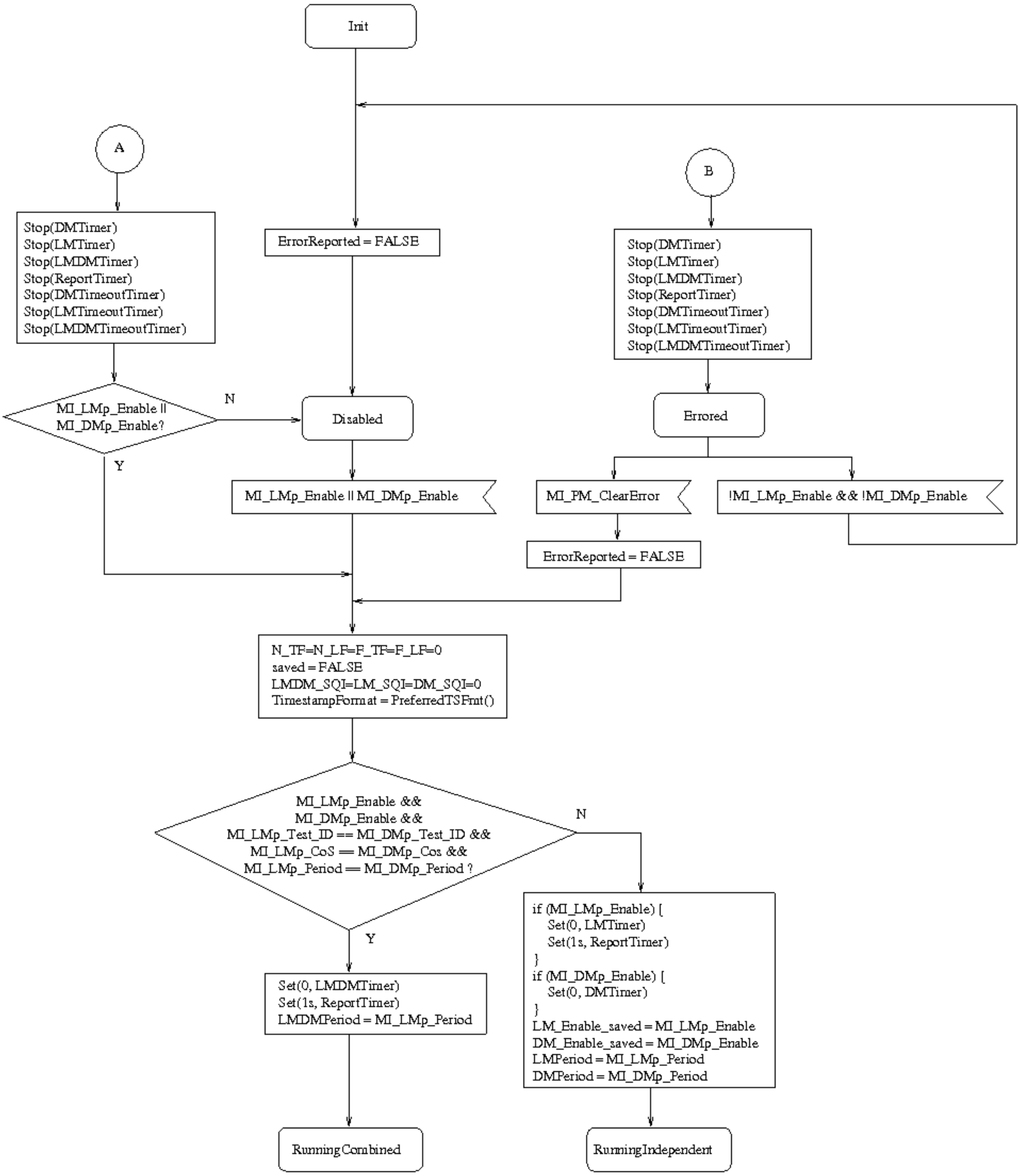
The Proactive PM Source control process performs delay measurements when MI_DMp_Enable is true and performs loss measurements when MI_LMp_Enable is true. If both are enabled, then where possible, the same PDUs are used to make both measurements (ie ILM+DM or DLM+DM PDUs). Otherwise, separate PDUs are used for loss (ILM or DLM) and delay (DM). The type of PDU used for loss is determined by MI_LMp_LMType, and can be “ILM” for inferred (synthetic) loss or “DLM” for direct (data traffic) loss measurement.

If an error is detected while the session is running, this is ~~reported~~ signalled via ~~MI_DMp_Report~~RI_PM_Error being set to True~~ or MI_LMp_ReportError,~~ and the session is ~~disabled~~ stopped until ~~MI_PM_ClearError is set~~RI_PM_Error is set to False or the session is disabled.

The PM protocol includes a mechanism to negotiate the packet sending period with the responder. If the period is changed from that specified by the management information (MI_DMp_Period or MI_LMp_Period), this is signalled via MI_DMp_PeriodChanged or MI_LMp_PeriodChanged.

MI_LMp_CoS and MI_DMp_CoS specify the CoS (traffic class) to use for the measurement. In the case of MI_LMp_CoS, this can either be a specific value, or the special value “ALL” indicating that loss across all traffic classes should be measured.

The proactive PM control process is described in Figure 8-19, 8-20, and 8-21. These figures include LM, DM and LMDM.


Init
A
B
Stop(DMTimer)
Stop(LMTimer)
Stop(LMDMTimer)
Stop(ReportTimer)
Stop(DMTimeoutTimer)
Stop(LMTimeoutTimer)
Stop(LMDMTimeoutTimer)
ErrorReported = FALSE
Stop(DMTimer)
Stop(LMTimer)
Stop(LMDMTimer)
Stop(ReportTimer)
Stop(DMTimeoutTimer)
Stop(LMTimeoutTimer)
Stop(LMDMTimeoutTimer)
MI_LMp_Enable ||
MI_DMp_Enable?
N
Y
Disabled
Errored
MI_LMp_Enable || MI_DMp_Enable
MI_PM_ClearError
!MI_LMp_Enable && !MI_DMp_Enable
ErrorReported = FALSE
N_TF=N_LF=F_TF=F_LF=0
saved = FALSE
LMDM_SQI=LM_SQI=DM_SQI=0
TimestampFormat = PreferredTSFmt()
MI_LMp_Enable &&
MI_DMp_Enable &&
MI_LMp_Test_ID == MI_DMp_Test_ID &&
MI_LMp_CoS == MI_DMp_Cos &&
MI_LMp_Period == MI_DMp_Period ?
N
Y
Set(0, LMDMTimer)
Set(1s, ReportTimer)
LMDMPeriod = MI_LMp_Period
if (MI_LMp_Enable) {
Set(0, LMTimer)
Set(1s, ReportTimer)
}
if (MI_DMp_Enable) {
Set(0, DMTimer)
}
LM_Enable_saved = MI_LMp_Enable
DM_Enable_saved = MI_DMp_Enable
LMPeriod = MI_LMp_Period
DMPeriod = MI_DMp_Period
RunningCombined
RunningIndependent

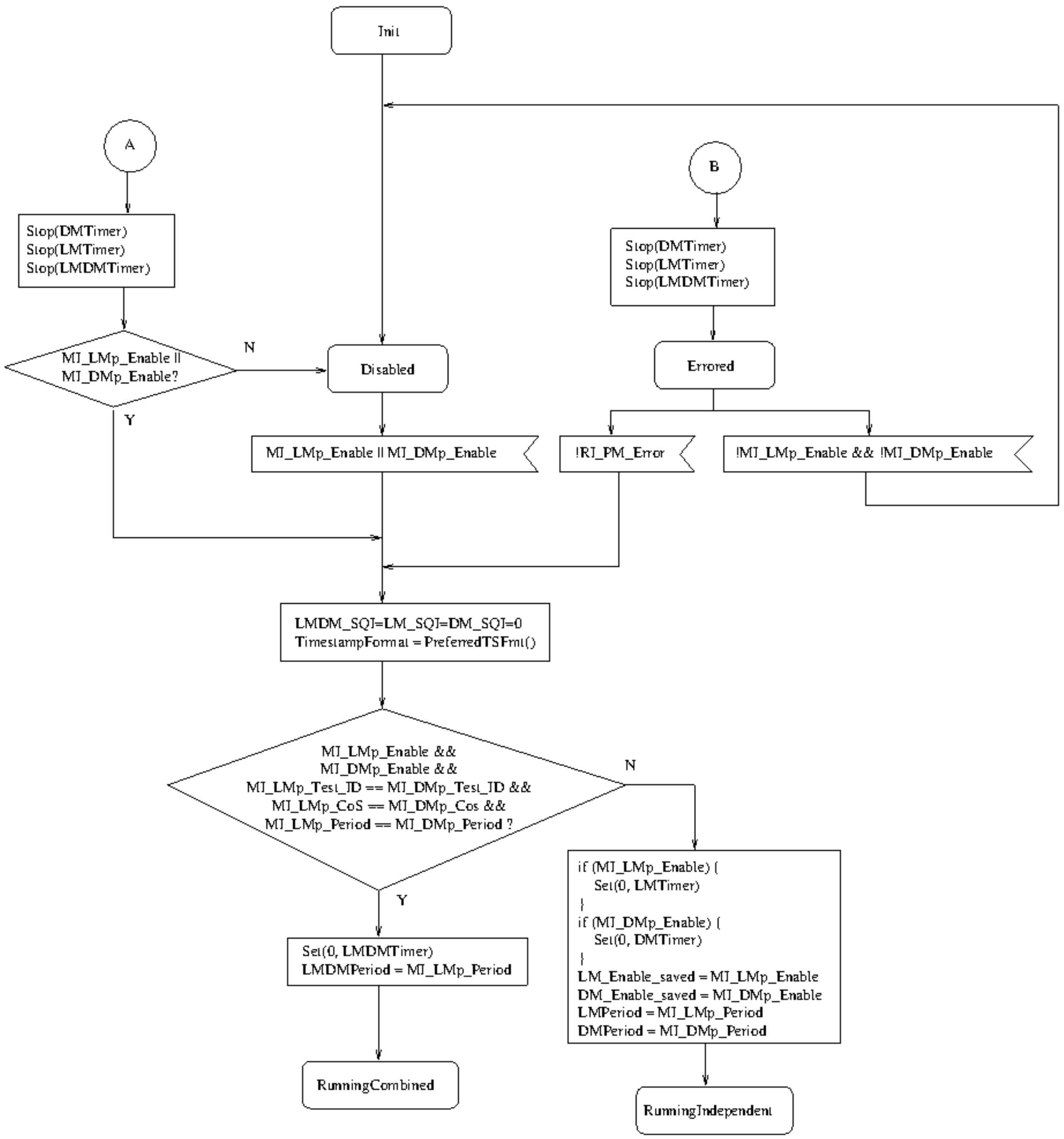
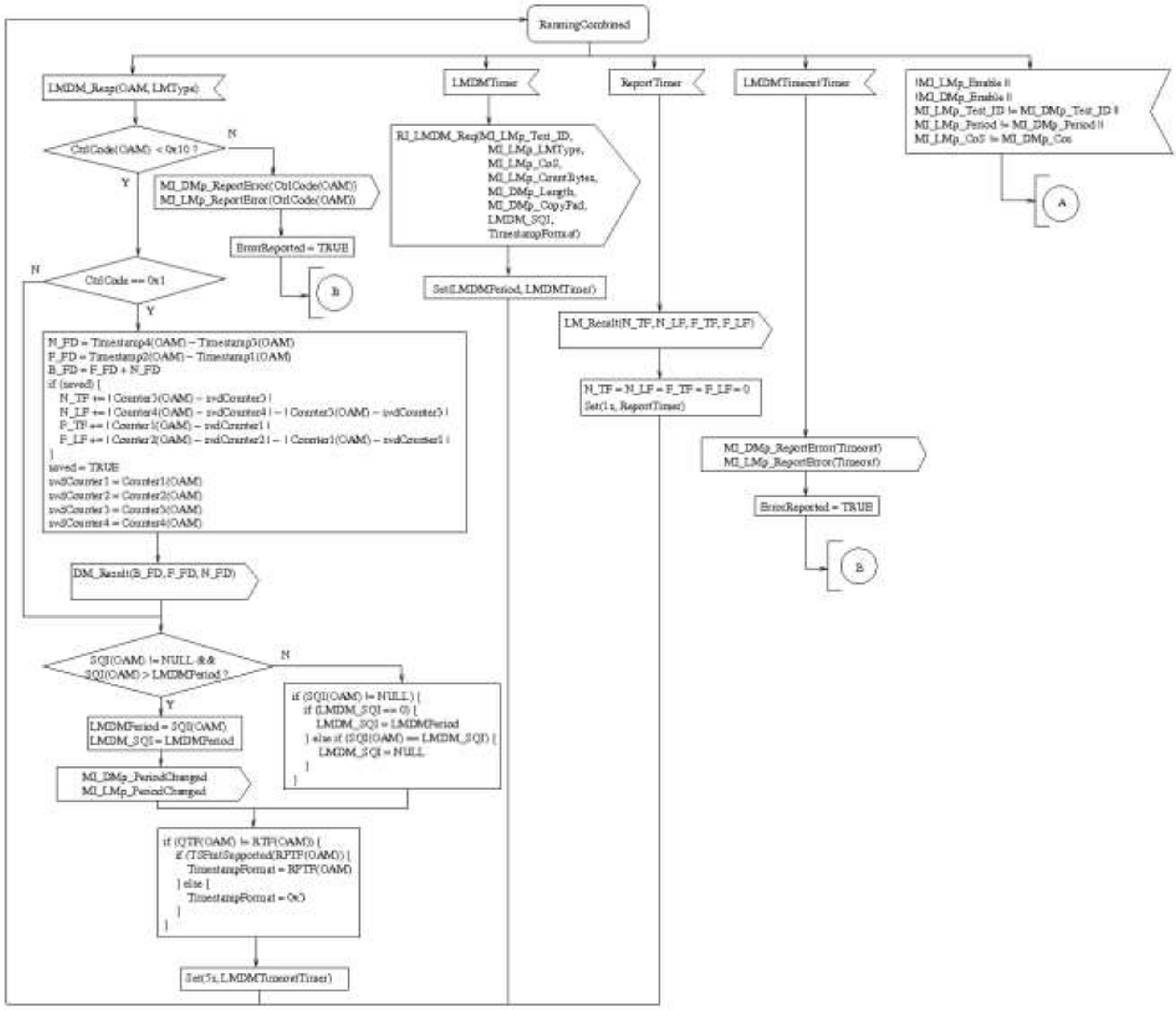
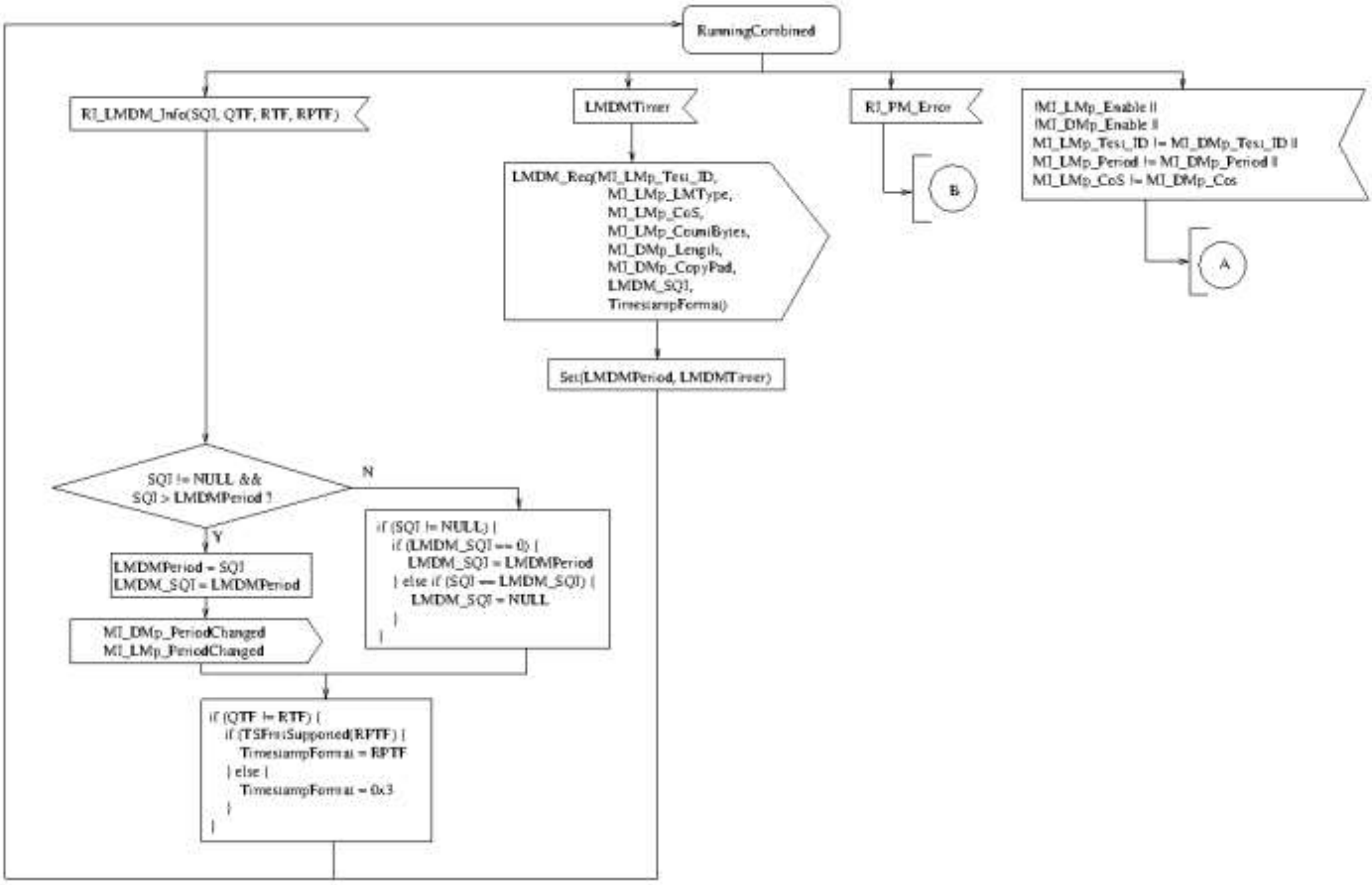
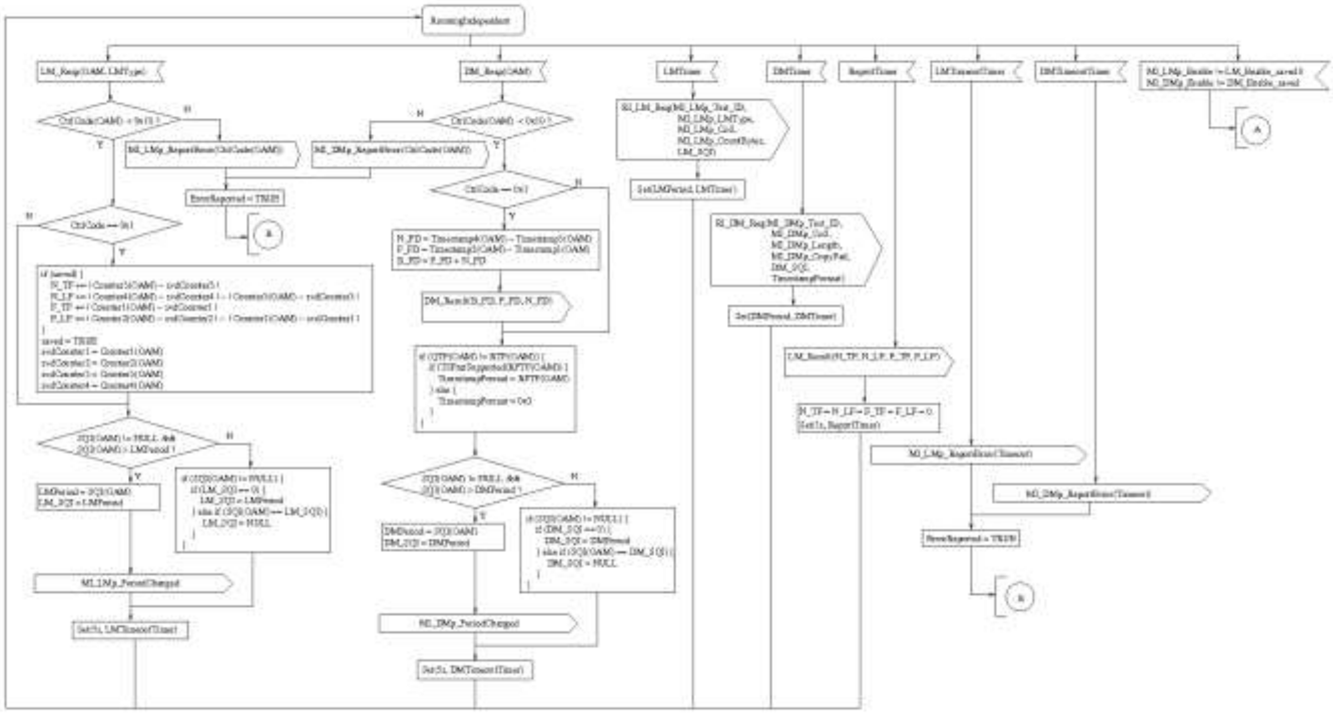


Figure 8-19/G.8121.2/Y.1381.2 proactive PM Source control process


RunningCombined
LMDM_Resp(OAM, LMType)
LMDMTimer
ReportTimer
LMDMTimeoutTimer
!MI_LMp_Enable ||
!MI_DMp_Enable ||
MI_LMp_Test_ID != MI_DMp_Test_ID ||
MI_LMp_Period != MI_DMp_Period ||
MI_LMp_CoS != MI_DMp_Cos
A
CtrlCode(OAM) < 0x10 ?
N
Y
MI_DMp_ReportError(CtrlCode(OAM))
MI_LMp_ReportError(CtrlCode(OAM))
ErrorReported = TRUE
B
CtrlCode == 0x1
N
Y
N_FD = Timestamp4(OAM) – Timestamp3(OAM)
F_FD = Timestamp2(OAM) – Timestamp1(OAM)
B_FD = F_FD + N_FD
if (saved) {
N_TF += | Counter3(OAM) – svdCounter3 |
N_LF += | Counter4(OAM) – svdCounter4 | – | Counter3(OAM) – svdCounter3 |
F_TF += | Counter1(OAM) – svdCounter1 |
F_LF += | Counter2(OAM) – svdCounter2 | – | Counter1(OAM) – svdCounter1 |
}
saved = TRUE
svdCounter1 = Counter1(OAM)
svdCounter2 = Counter2(OAM)
svdCounter3 = Counter3(OAM)
svdCounter4 = Counter4(OAM)
DM_Result(B_FD, F_FD, N_FD)
RI_LMDM_Req(MI_LMp_Test_ID,
MI_LMp_LMType,
MI_LMp_CoS,
MI_LMp_CountBytes,
MI_DMp_Length,
MI_DMp_CopyPad,
LMDM_SQI,
TimestampFormat)
Set(LMDMPeriod, LMDMTimer)
LM_Result(N_TF, N_LF, F_TF, F_LF)
N_TF = N_LF = F_TF = F_LF = 0
Set(1s, ReportTimer)
MI_DMp_ReportError(Timeout)
MI_LMp_ReportError(Timeout)
ErrorReported = TRUE
B
SQI(OAM) != NULL &&
SQI(OAM) > LMDMPeriod ?
N
Y
LMDMPeriod = SQI(OAM)
LMDM_SQI = LMDMPeriod
MI_DMp_PeriodChanged
MI_LMp_PeriodChanged
if (SQI(OAM) != NULL) {
if (LMDM_SQI == 0) {
LMDM_SQI = LMDMPeriod
} else if (SQI(OAM) == LMDM_SQI) {
LMDM_SQI = NULL
}
}
if (QTF(OAM) != RTF(OAM)) {
if (TSFmtSupported(RPTF(OAM)) {
TimestampFormat = RPTF(OAM)
} else {
TimestampFormat = 0x3
}
}
Set(5s, LMDMTimeoutTimer)



<u>**Figure 8-20/G.8121.2/Y.1381.2 proactive PM Source control process**</u>



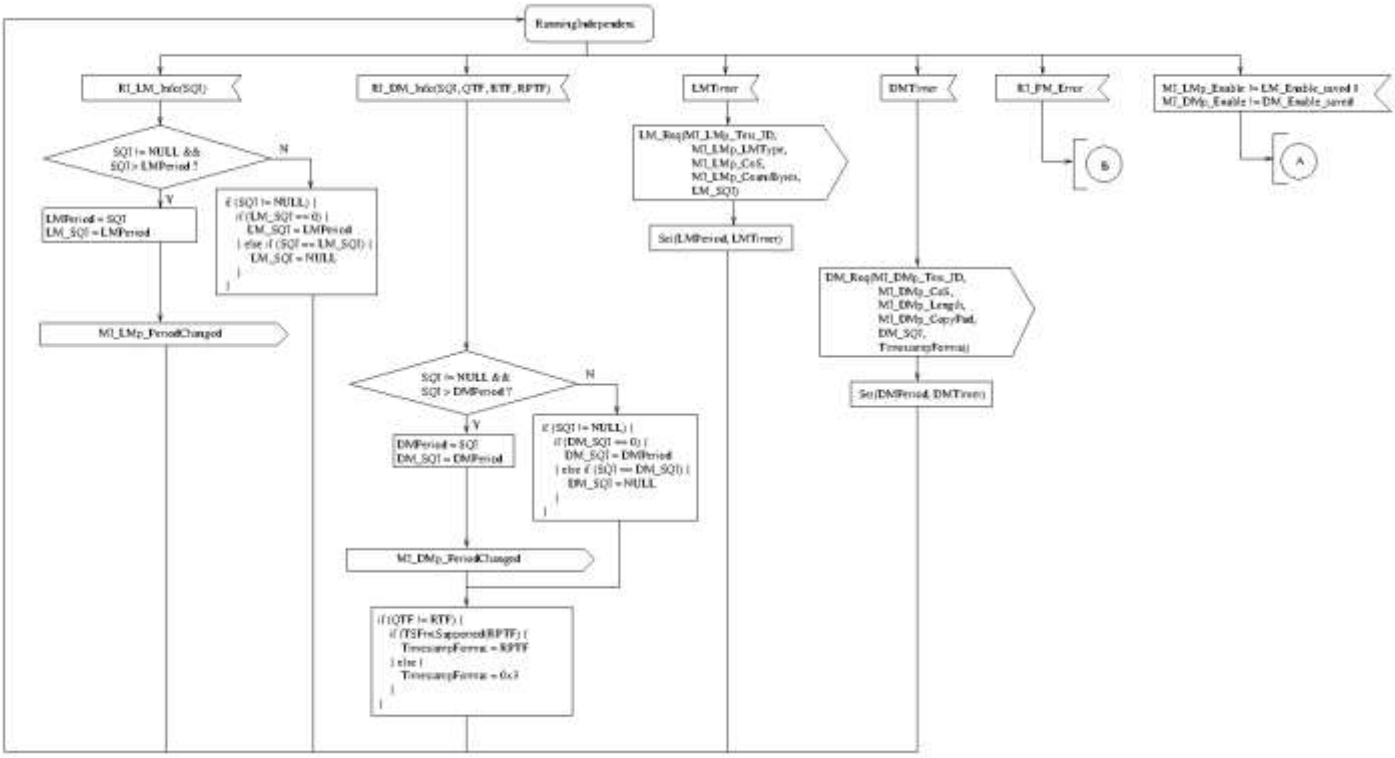


Figure 8-21/G.8121.2/Y.1381.2 proactive PM Source control process

The TSFmtSupported() function determines whether the specified timestamp format, from [RFC6374], is supported by the implementation, while the PreferredTSFmt() function returns the timestamp format that is preferred by the implementation, as described in [RFC6374].

Note that both the period and the timestamp format are negotiated with the responder. The period is negotiated by setting the SQI appropriately, while the timestamp format is negotiated via the QTF, RTF and RPTF fields. Initially, the implementation's preferred timestamp is used. If the responder does not respond to the first request using the same timestamp format, then the responder's preferred timestamp format is used if it is supported, otherwise the IEEE 1588v1 format is used as described in [RFC6374]. Note that support for this format is mandatory.

8.8.4.2 Proactive PM Sink control process for LM

The proactive PM Source control process includes LM, DM and LMDM. Each instance of the process operates a single proactive PM session. Multiple sessions can be supported by instantiating multiple instances of the process, along with corresponding instances of the PM Source Control, PM generation and PM reception processes.

As for the source control process, the Proactive PM Sink control process performs delay measurements when MI_DMp_Enable is true and performs loss measurements when MI_LMp_Enable is true. If both are enabled, then where possible, the same PDUs are used to make both measurements (ie ILM+DM or DLM+DM PDUs). Otherwise, separate PDUs are used for loss (ILM or DLM) and delay (DM).

If an error is detected while the session is running, this is reported via MI_DMp_ReportError or MI_LMp_ReportError, and the session is stopped until MI_PM_ClearError is set or the session is disabled.

The proactive PM sink control process is described in Figure 8-21a, 8-21b, and 8-21c. These figures include LM, DM and LMDM.

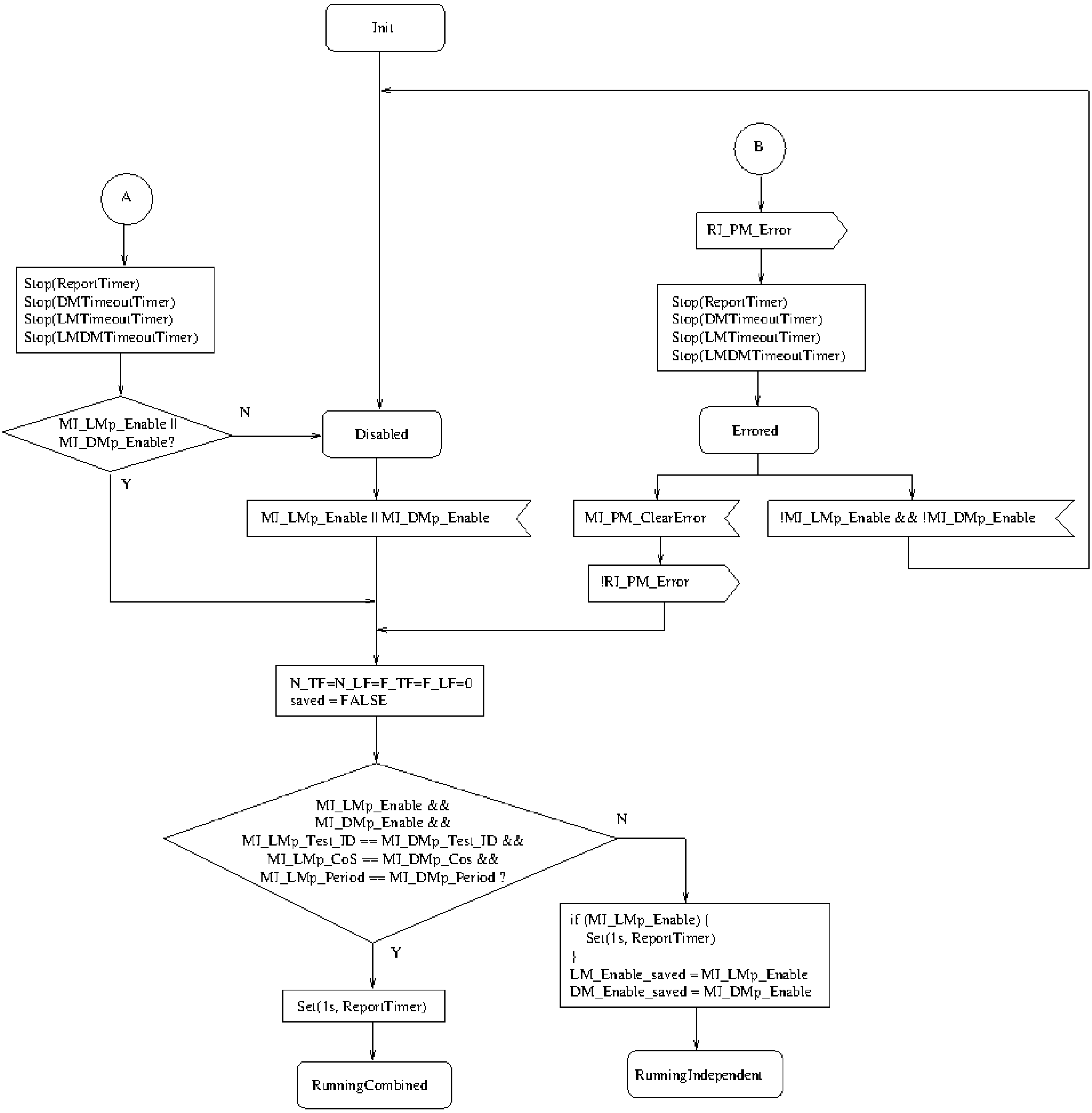


Figure 8-21a/G.8121.2/Y.1381.2 proactive PM Sink control process

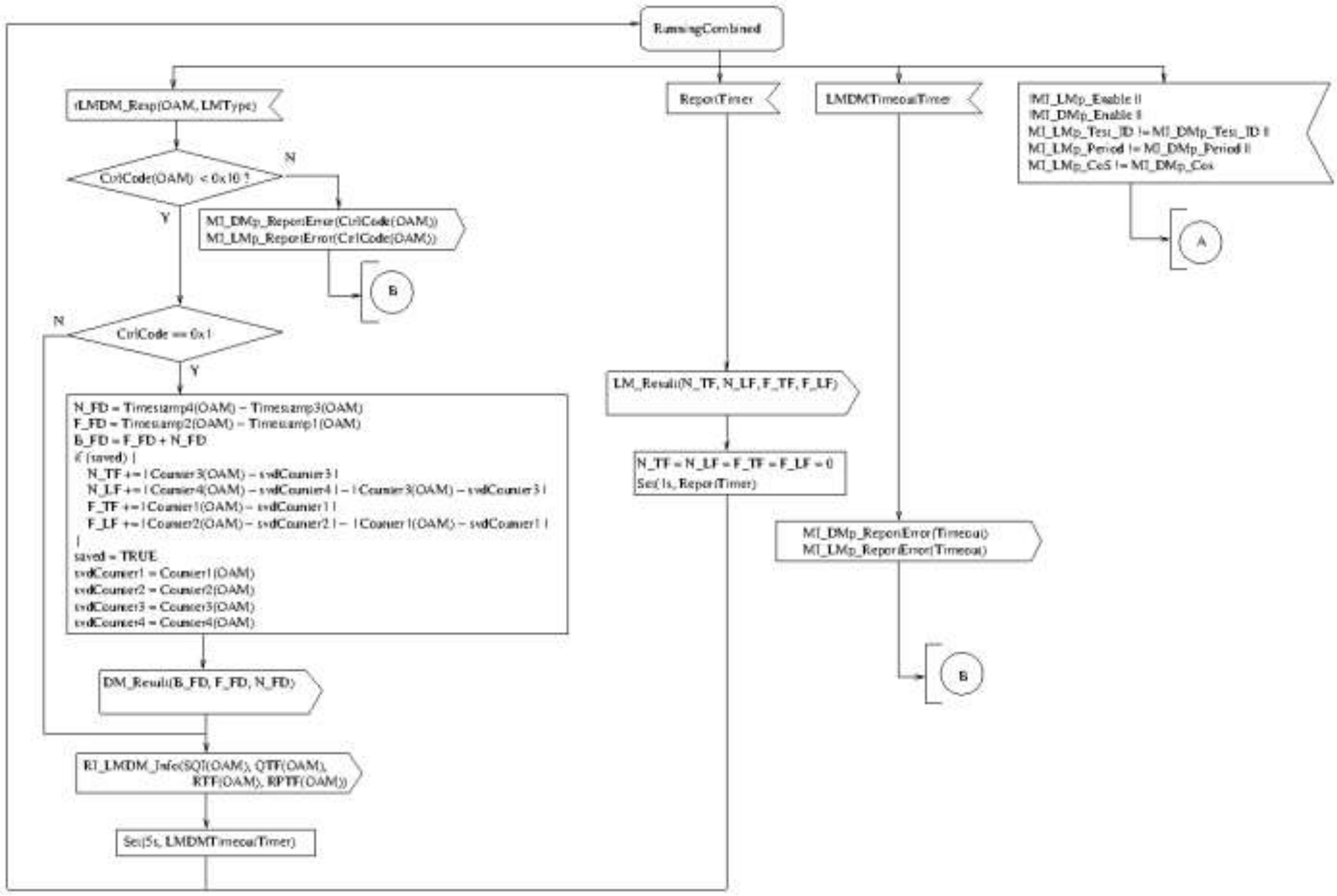


Figure 8-21b/G.8121.2/Y.1381.2 proactive PM Sink control process

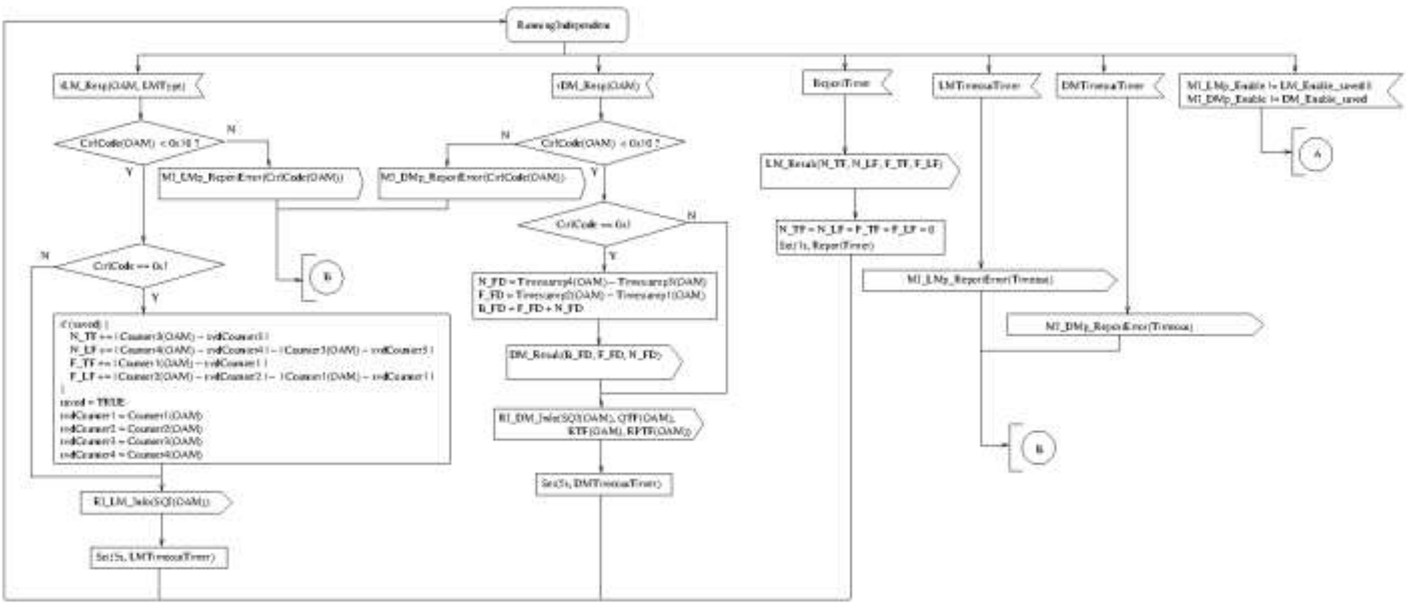


Figure 8-21c/G.8121.2/Y.1381.2 proactive PM Sink control process

8.8.4.3 PM generation process for Proactive LM

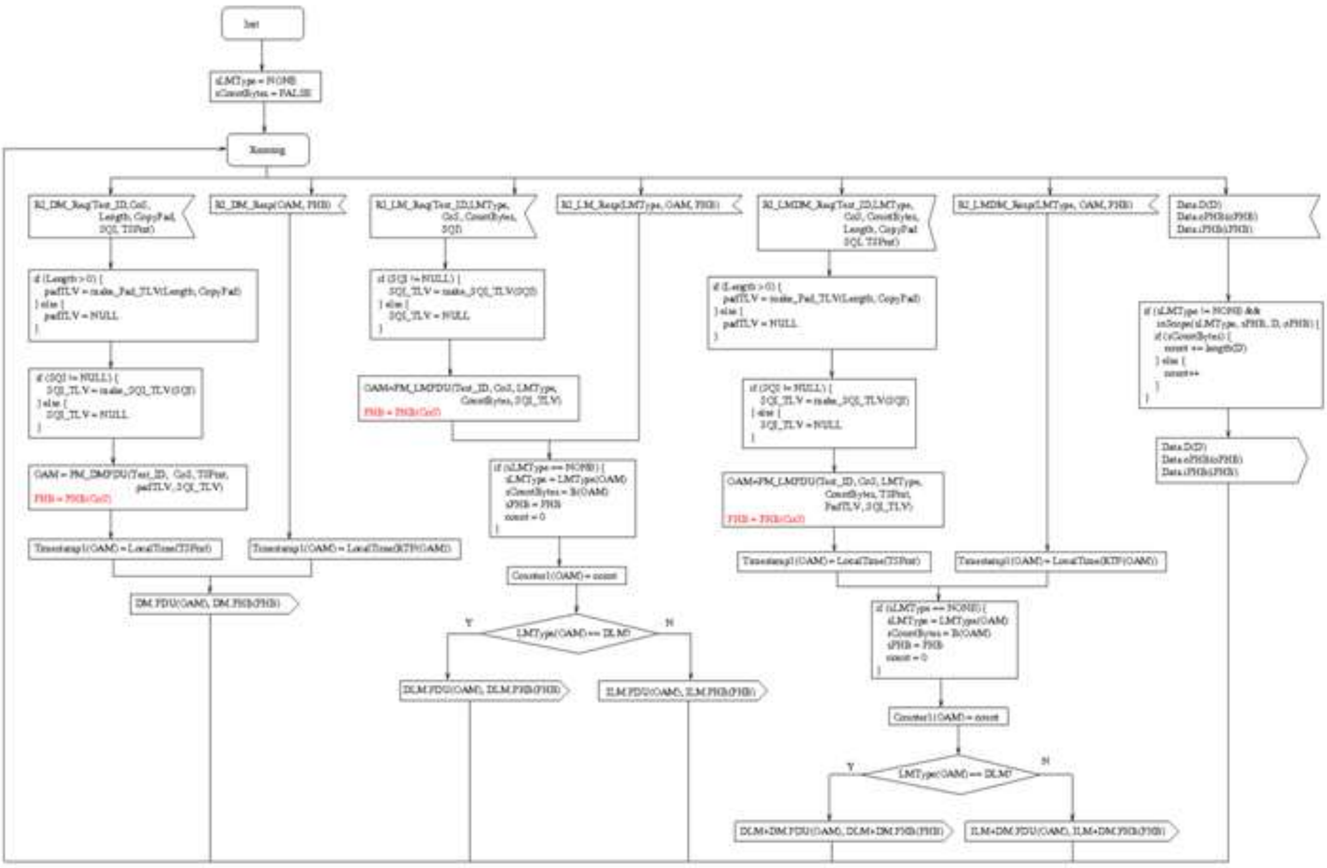
The PM generation process includes LM, DM and LMDM. It generates PM requests when it receives the ~~RI_~~DM_Req, ~~RI_~~LM_Req or ~~RI_~~LMDM_Req signals from the corresponding Proactive Source or On-demand control process, and generates PM responses when it receives the RI_DM_Resp, RI_LM_Resp or RI_LMDM_Resp signals from the corresponding PM responder process.

For delay measurement, it writes the packet send time into the PDU, using the requested timestamp format.

For loss measurement, it counts the appropriate traffic depending on the type of loss measurement, and writes the counters into the transmitted PM PDUs. The packets to count are dependent on the LMType (ILM or DLM) and the CoS (which may be a particular value, or the special value “ALL”). If the CountBytes parameter is set, the number of bytes in each matching packet is counted, otherwise the count is simply incremented for each matching packet.

In the ~~RI_~~DM_Req, ~~RI_~~LM_Req and ~~RI_~~LMDM_Req signals, the SQI parameter specifies the value to place in the SQI TLV. If it is set to NULL, no SQI TLV is included. The TSFmt parameter specifies the timestamp format to use when writing timestamps. The Length parameter specifies the length of padding to include in the PDU. If set to 0, no Padding TLV is included.

The PM generation process is described in Figure 8-22:



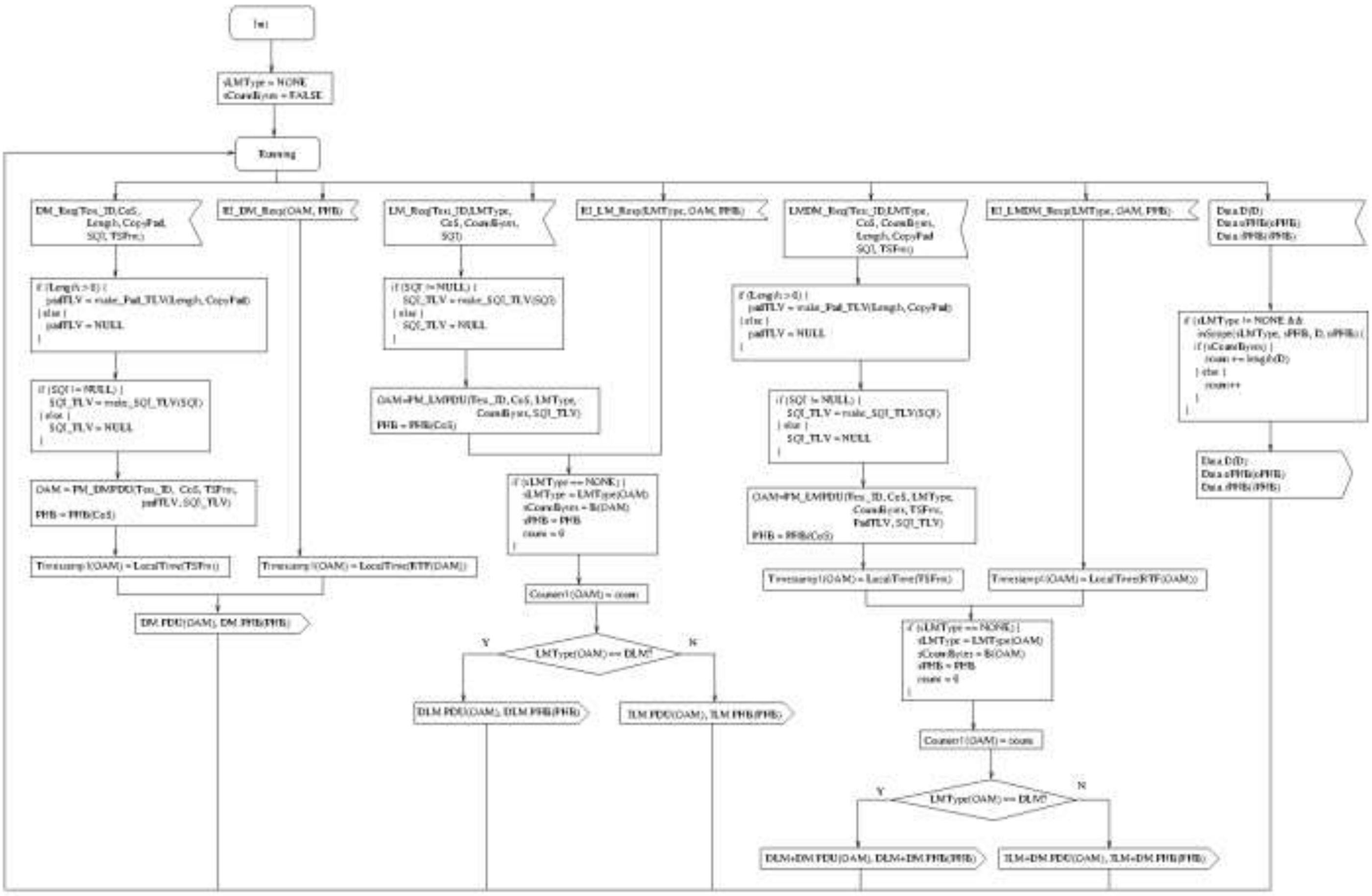


Figure 8-22/G.8121.2/Y.1381.2 - PM Generation Process

The make_Pad_TLV(Length, CopyPad) function creates a Padding TLV as specified in [RFC6374], as follows:

- If CopyPad is set, the Type is set to 0, otherwise it is set to 128.
- The Length field is set to Length
- The Value field is set to all 0s.

The make_SQI_TLV(SQI) function creates an SQI TLV as specified in [RFC6374], as follows:

- The Type is set to 2
- The Length field is set to 4.
- The Value field is set to SQI.

The PM_DMPDU(Test_ID, TSFmt, CoS, padTLV, SQI_TLV) function creates a DM PDU as specified in [RFC6374], as follows:

- The version is set to 0
- The R flag is unset; the T flag is set; and the rest of the flags field is set to 0.
- The control code is set to 0. Other values for the control code are FFS.
- The message length is set to the total length of the PDU.

- The QTF field is set to TSFmt, the RTF and RPTF fields are set to 0
- The reserved field is set to 0.
- The session ID and DS fields are set to Test_ID and CoS respectively.
- The timestamp fields are all set to 0.
- The pad TLV and SQI TLV, if not NULL, are appended to the message. The use of other TLVs is FFS.

The PM_LMPDU() function creates an ILM or DLM PDU as specified in [RFC6374], as follows:

- The version is set to 0
- The R flag is unset; the T flag is set if a specific CoS value has been specified and is unset if the CoS is set to "ALL"; and the rest of the flags field is set to 0.
- The control code is set to 0. Other values for the control code are FFS.
- The message length is set to the total length of the PDU.
- In the Dflag field, the X flag is set appropriate depending whether the implementation writes 32 or 64 bit counters; the B flag is set if CountBytes is set, and is unset otherwise; and the rest of the field is set to 0.
- The OTF field is set to the implementations preferred timestamp format.
- The reserved field is set to 0.
- The session ID and DS fields are set to Test_ID and CoS respectively. If the CoS is "ALL", the DS field is set to 0.
- The origin timestamp field is set to the local time-of-day, using the format specified in the OTF field.
- The counter fields are all set to 0.
- The SQI TLV, if not NULL, is appended to the message. The use of other TLVs is FFS.

The PHB(CoS) function returns the PHB with the lowest drop precedence within the Class of Service defined by the CoS input parameter.

The PM_LMDMPDU() function creates an ILM+DM or DLM_DM PDU as specified in [RFC6374], in a similar way to the DM and LM cases described above.

The InScope() function determines whether a given data packet should be counted, depending on the LM Type (ILM or DLM) and the CoS/PHB (a specific TC value or "ALL").

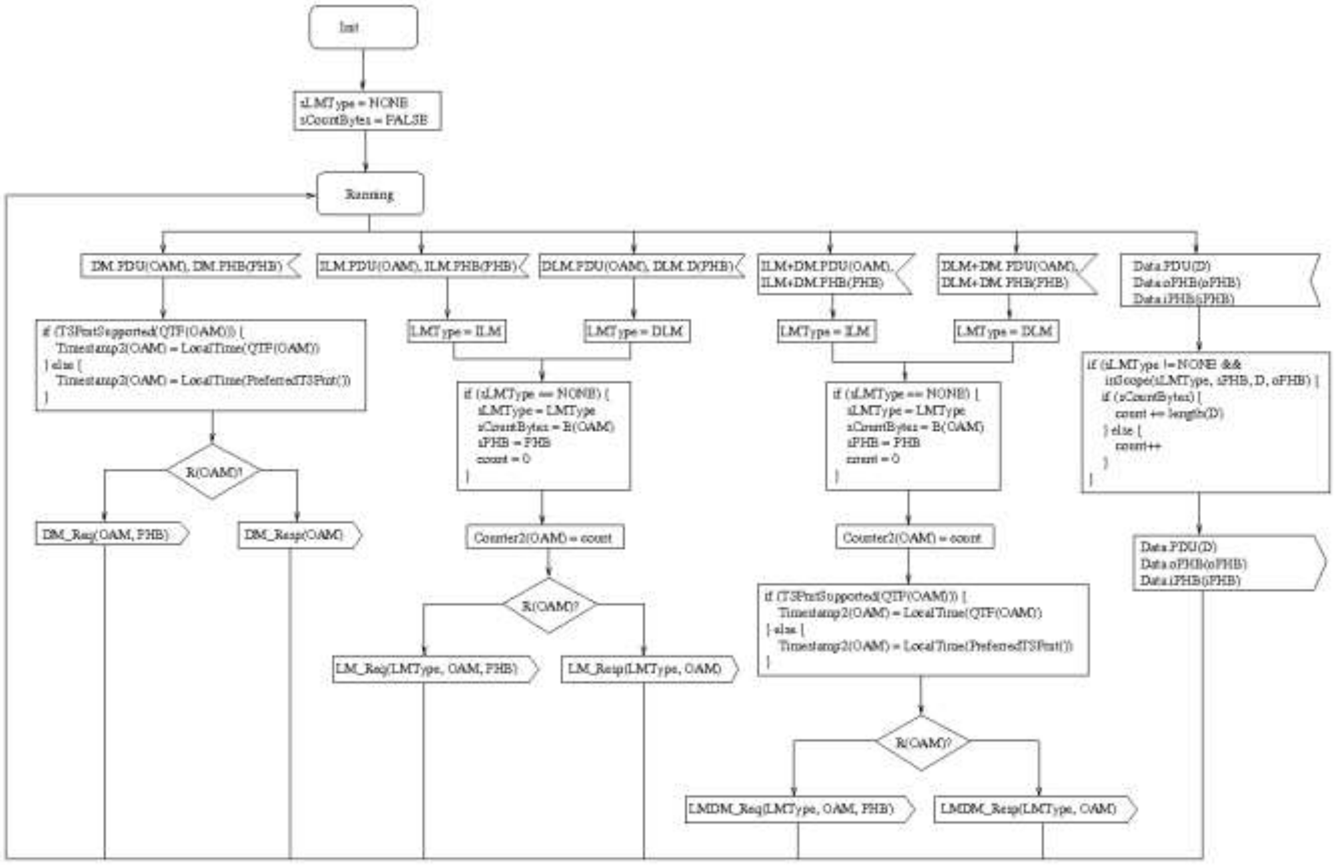
The LocalTime(TSFmt) function returns the local time-of-day, in the format specified.

8.8.4.4 PM reception process for Proactive LM

The PM Reception process receives PM message for a given Test ID, and passes them to the corresponding Proactive or On-demand control process or PM Responder process.

For delay measurement, it writes the packet receive time into the PDU. For loss measurement, it counts the appropriate traffic depending on the type of loss measurement, and writes the counters into the received PM PDUs. The packets to count are dependent on the LMType (ILM or DLM) and the CoS (which may be a particular value, or the special value "ALL"). If the CountBytes bit is set, the number of bytes in each matching packet is counted, otherwise the count is simply incremented for each matching packet.

The PM reception process is described in Figure 8-23:



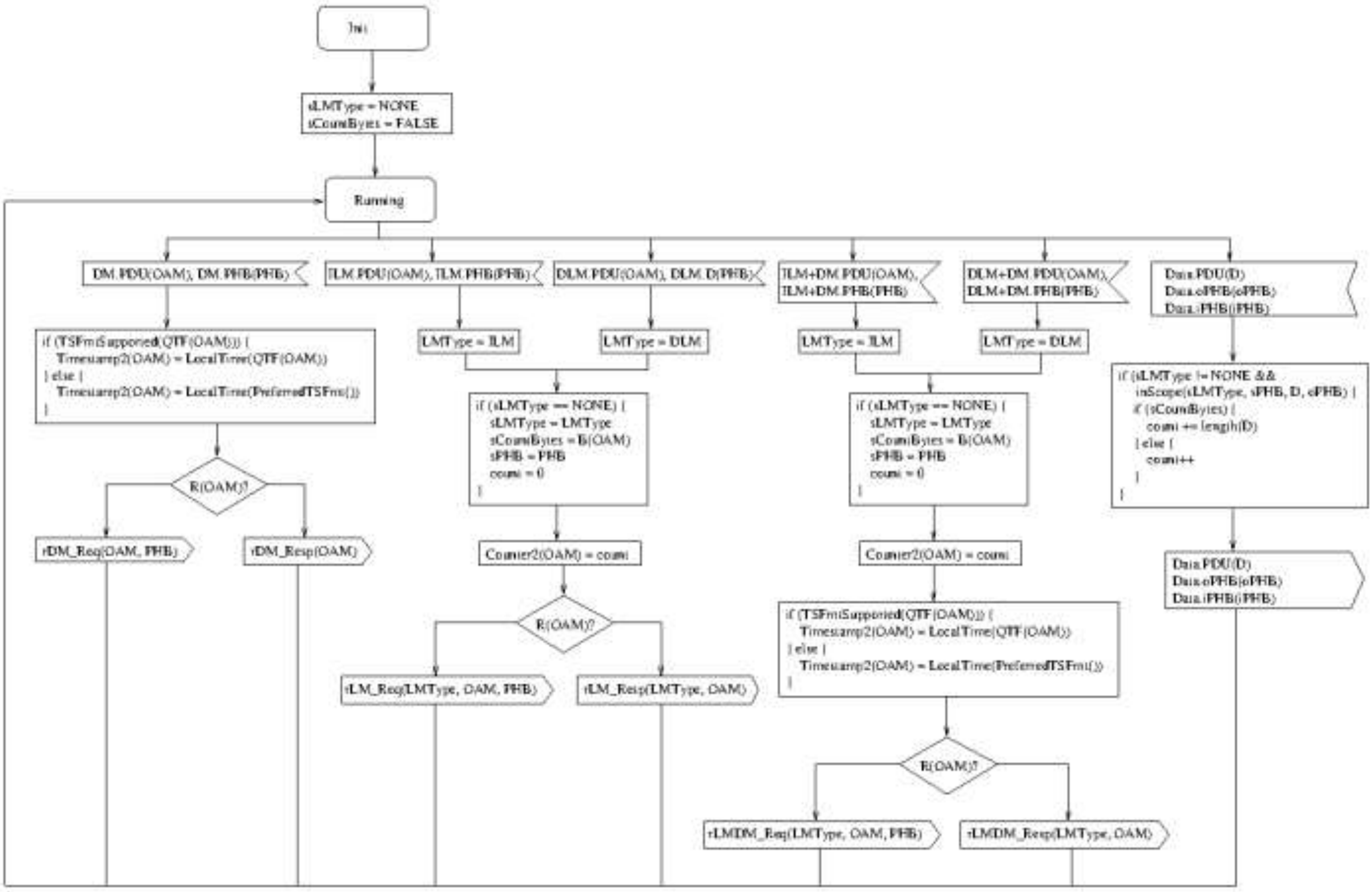
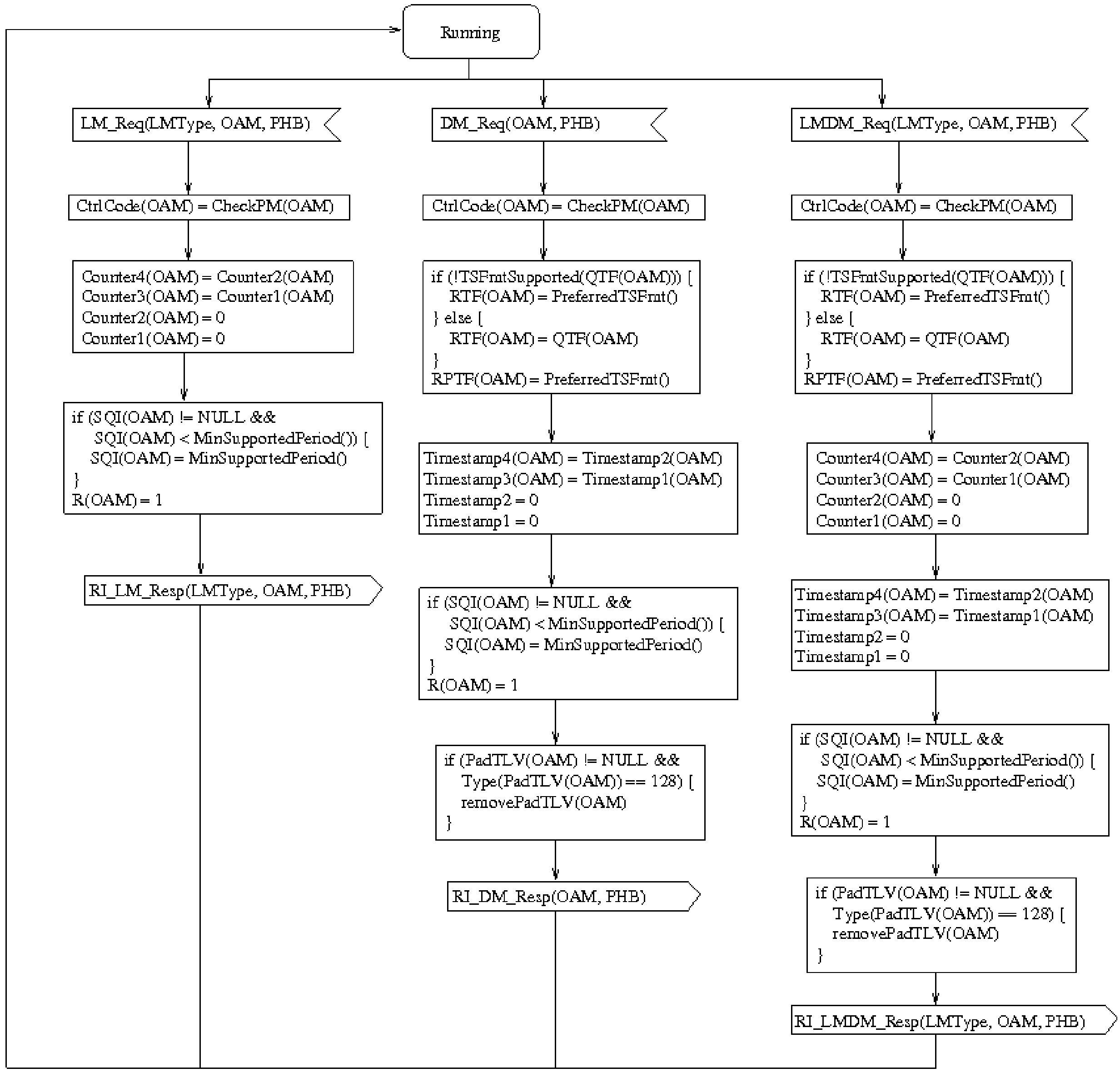


Figure 8-23/G.8121.2/Y.1381.2 - PM reception process

8.8.4.5 PM Responder process for Proactive LM

The PM responder process responds to PM messages for a single PM session. Multiple sessions can be supported by instantiating multiple instances of the process, along with corresponding instances of the PM generation and PM reception processes.

The PM responder process is described in figure 8-24:


Running
LM_Req(LMType, OAM, PHB)
DM_Req(OAM, PHB)
LMDM_Req(LMType, OAM, PHB)
CtrlCode(OAM) = CheckPM(OAM)
CtrlCode(OAM) = CheckPM(OAM)
CtrlCode(OAM) = CheckPM(OAM)
Counter4(OAM) = Counter2(OAM)
Counter3(OAM) = Counter1(OAM)
Counter2(OAM) = 0
Counter1(OAM) = 0
if (!TSFmtSupported(QTF(OAM))) {
RTF(OAM) = PreferredTSFmt()
} else {
RTF(OAM) = QTF(OAM)
}
RPTF(OAM) = PreferredTSFmt()
if (!TSFmtSupported(QTF(OAM))) {
RTF(OAM) = PreferredTSFmt()
} else {
RTF(OAM) = QTF(OAM)
}
RPTF(OAM) = PreferredTSFmt()
if (SQI(OAM) != NULL &&
SQI(OAM) < MinSupportedPeriod()) {
SQI(OAM) = MinSupportedPeriod()
}
R(OAM) = 1
Timestamp4(OAM) = Timestamp2(OAM)
Timestamp3(OAM) = Timestamp1(OAM)
Timestamp2 = 0
Timestamp1 = 0
Counter4(OAM) = Counter2(OAM)
Counter3(OAM) = Counter1(OAM)
Counter2(OAM) = 0
Counter1(OAM) = 0
RI_LM_Resp(LMType, OAM, PHB)
if (SQI(OAM) != NULL &&
SQI(OAM) < MinSupportedPeriod()) {
SQI(OAM) = MinSupportedPeriod()
}
R(OAM) = 1
Timestamp4(OAM) = Timestamp2(OAM)
Timestamp3(OAM) = Timestamp1(OAM)
Timestamp2 = 0
Timestamp1 = 0
if (PadTLV(OAM) != NULL &&
Type(PadTLV(OAM)) == 128) {
removePadTLV(OAM)
}
if (SQI(OAM) != NULL &&
SQI(OAM) < MinSupportedPeriod()) {
SQI(OAM) = MinSupportedPeriod()
}
R(OAM) = 1
RI_DM_Resp(OAM, PHB)
if (PadTLV(OAM) != NULL &&
Type(PadTLV(OAM)) == 128) {
removePadTLV(OAM)
}
RI_LMDM_Resp(LMType, OAM, PHB)

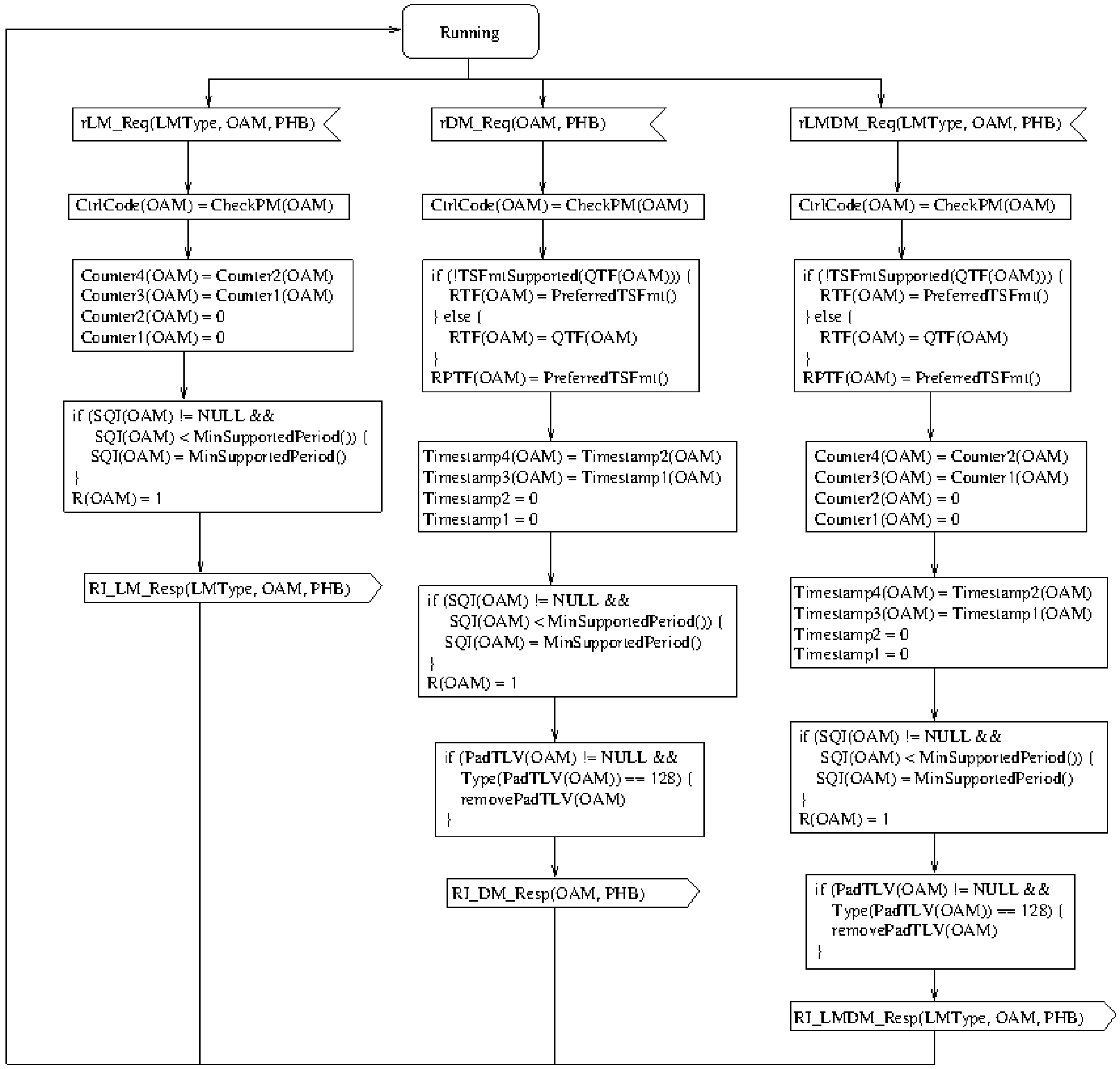


Figure 8-24/G.8121.2/Y.1381.2 PM responder process

The CheckPM() function checks the received PDU and returns an appropriate control code, as described in [RFC6374]. In particular, it returns 0x19 (Administrative Block) if MI_PM_Responder_Enable is not set, and 0x2 (Data Format Invalid) if the QTF in a DM, ILM+DM or DLM_DM message is not supported.

Note that when MI_PM_Responder_Enable is not set, responses are still sent, with the above error.

The PM responder process also unsets the X flag in LM messages if the implementation does not support 64 bit counters.

8.8.5. On-demand Packet Loss Measurement (LMo)

As described in clauses 7.2.2.1 and 8.6 to 8.8 of [ITU-T G.8113.2], loss and delay measurements may be combined. The format for the combined measurement, refered to here as LMDM, is described in section 3.3 of [RFC 6374]. In addition, the same LM and DM protocols can be used for both proactive and on-demand measurement.

An overview of the performance monitoring processes for a single on-demand PM session is shown in figure 8-24a. The same processes are used for LM, DM or LMDM.

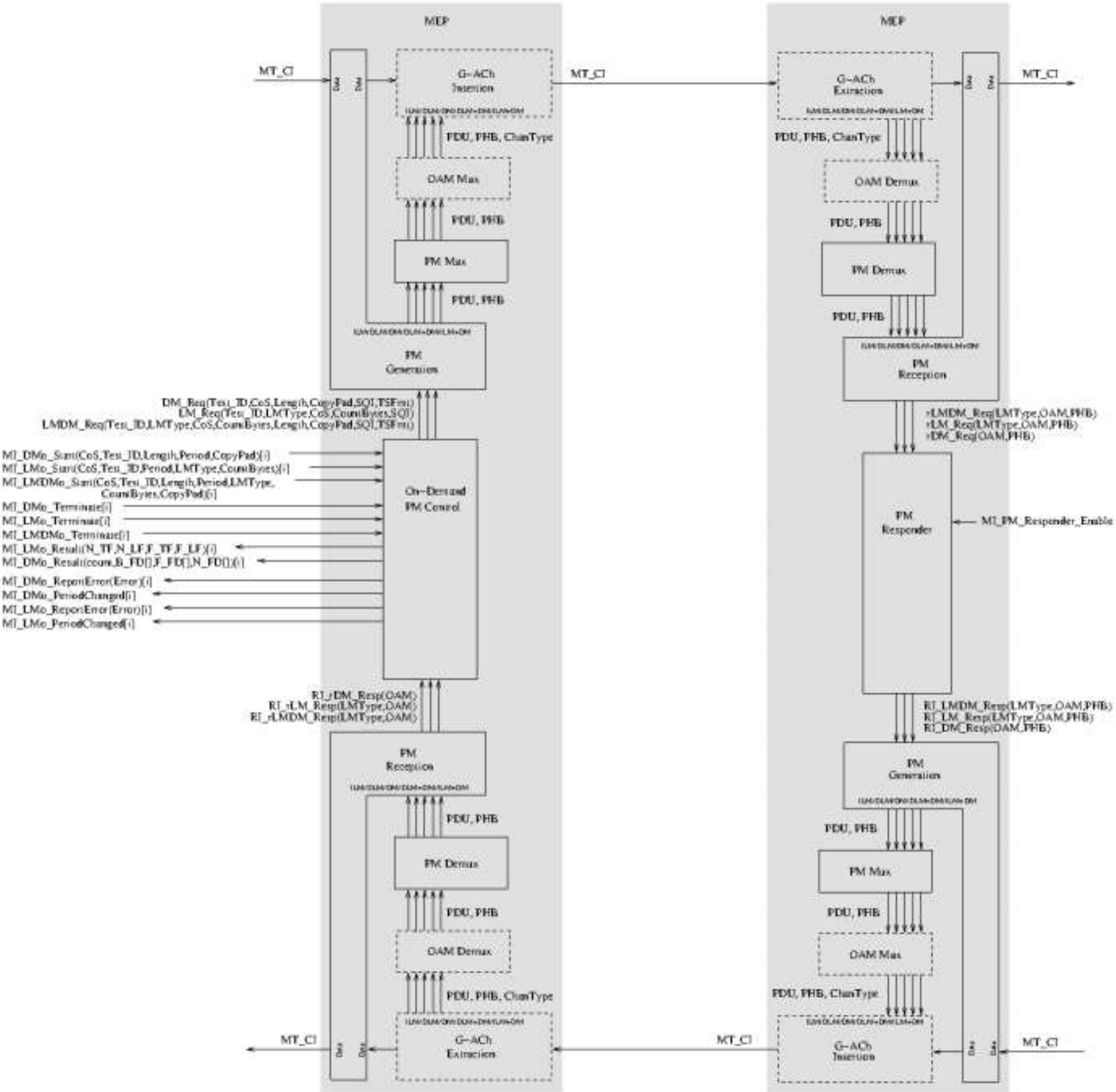


Figure 8-24a/G.8121.2/Y.1381.2 On-demand PM processes

The On-Demand PM control process controls the session, including scheduling request packets, and processing responses to calculate performance metrics. The other processes shown are the same as those used for Proactive LM, as described in clause 8.8.4.

8.8.5.1 On-Demand control process for LM

The on-demand PM control process includes LM, DM and LMDM. Each instance of the process operates a single on-demand PM session. Multiple sessions can be supported by instantiating multiple instances of the process, along with corresponding instances of the PM generation and PM reception processes.

The on-demand PM control process performs either delay measurement (via MI_DMo_Start/MI_DMo_Terminate), loss measurement (via MI_LMo_Start/MI_LMo_Terminate) or both simultaneously (via MI_LMDMo_Start/MI_LMDMo_Terminate). The type of loss measurement to perform is specified by the LMType parameter, and can be “ILM” for inferred (synthetic) loss or “DLM” for direct (data traffic) loss.

Results are reported via MI_DMo_Result and MI_LMo_Result.

If an error is detected while the session is running, this is reported via MI_DMo_ReportError or MI_LMo_ReportError, and the session is terminated automatically. The results collected up to that point are reported.

The PM protocol includes a mechanism to negotiate the packet sending period with the responder. If the period is changed from that specified when the session was started, this is signalled via MI_DMo_PeriodChanged or MI_LMo_PeriodChanged.

The CoS parameter of MI_LMo_Start, MI_DMo_Start or MI_LMDMo_Start specifies the CoS (traffic class) to use for the measurement. In the case of MI_LMo_Start, this can either be a specific value, or the special value “ALL” indicating that loss across all traffic classes should be measured.

The on-demand PM control process is described in Figure 8-25, 8-26, 8-27, and 8-28..

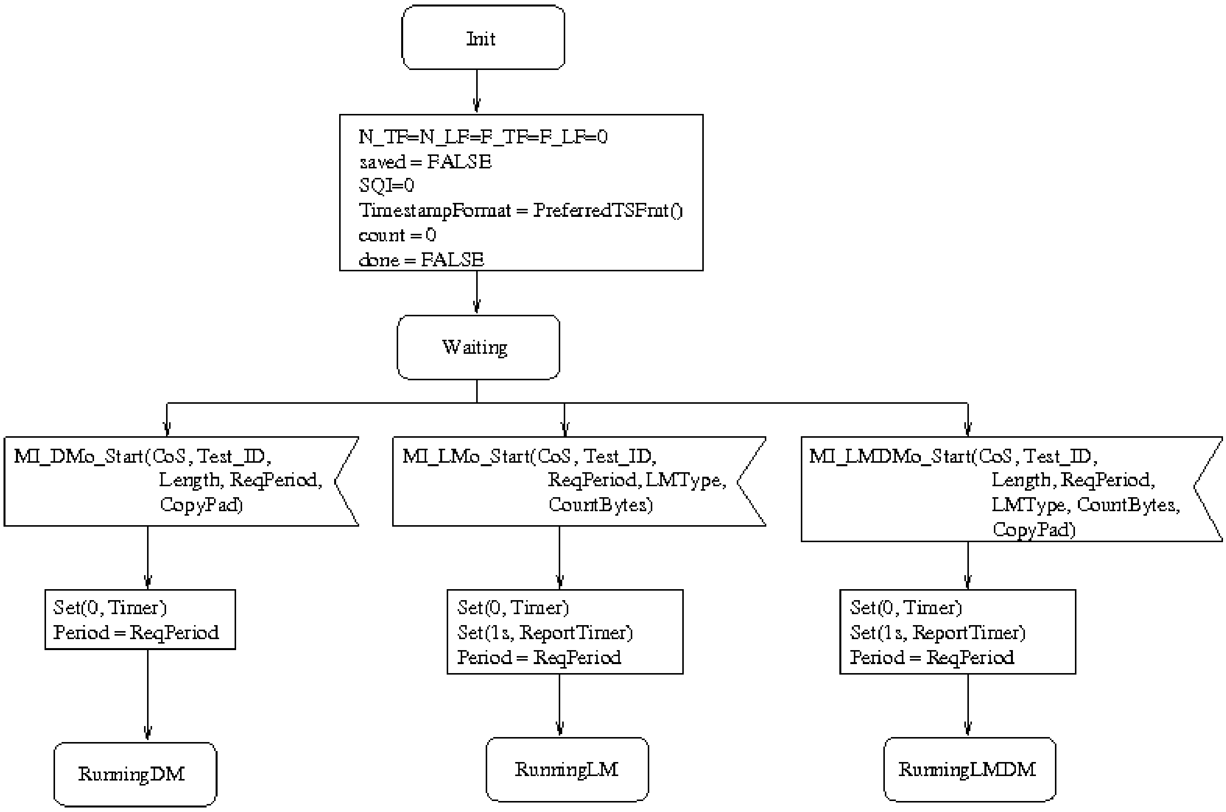
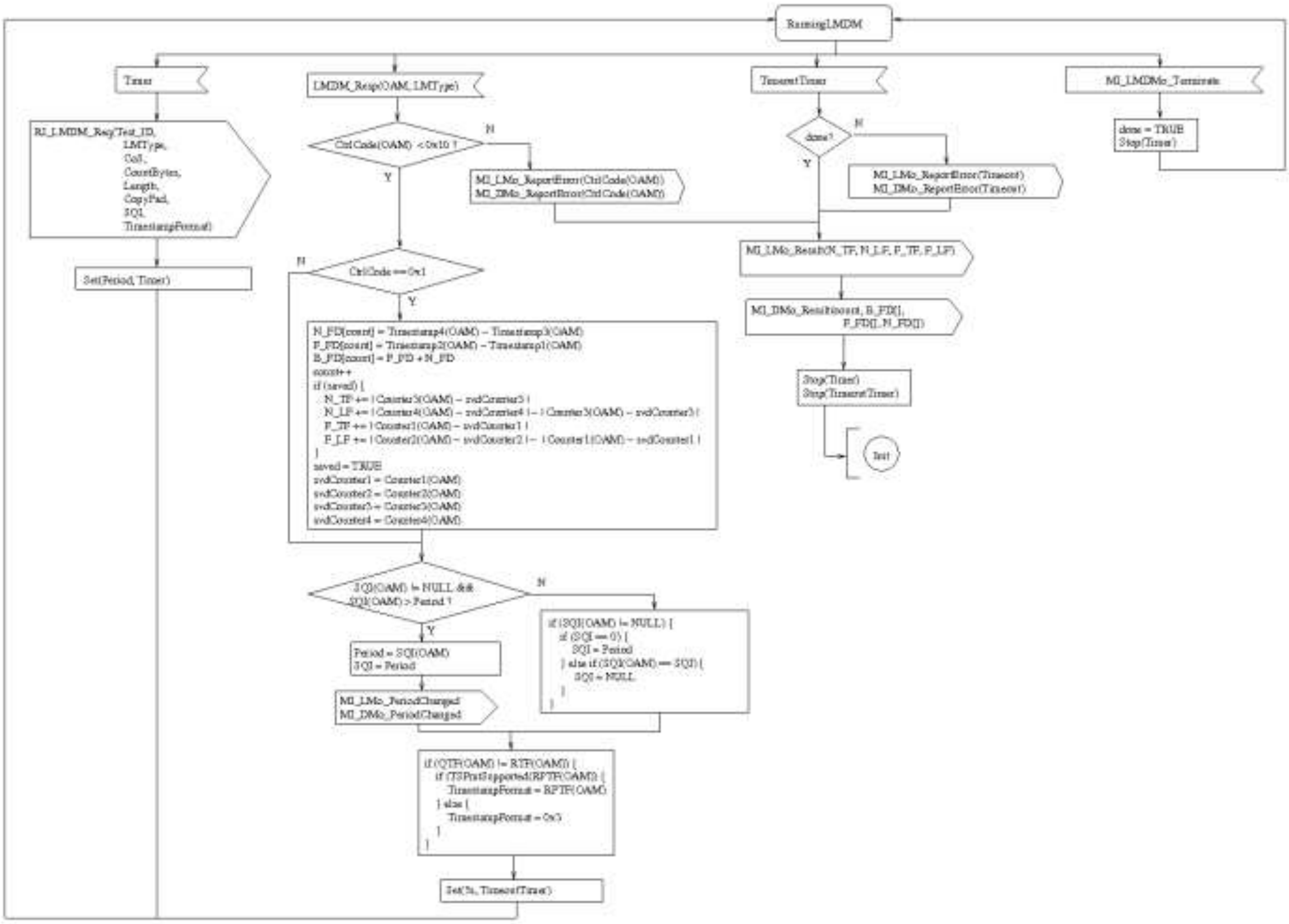


Figure 8-25/G.8121.2/Y.1381.2 On-demand PM control process


RunningLMDM
Timer
LMDM_Resp(OAM, LMType)
TimeoutTimer
MI_LMDMo_Terminate
RI_LMDM_Req(Test_ID,
LMType,
Coll,
CountBytes,
Length,
CopyPad,
SQI,
TimestampFormat)
Set(Period, Timer)
CtrlCode(OAM) < 0x10 ?
N
Y
MI_LMo_ReportError(CtrlCode(OAM))
MI_DMo_ReportError(CtrlCode(OAM))
CtrlCode == 0x1
N_FD[count] = Timestamp4(OAM) – Timestamp3(OAM)
F_FD[count] = Timestamp2(OAM) – Timestamp1(OAM)
B_FD[count] = F_FD + N_FD
count++
if (saved) {
N_TF += | Counter3(OAM) – svdCounter3 |
N_LF += | Counter4(OAM) – svdCounter4 | – | Counter3(OAM) – svdCounter3 |
F_TF += | Counter1(OAM) – svdCounter1 |
F_LF += | Counter2(OAM) – svdCounter2 | – | Counter1(OAM) – svdCounter1 |
}
saved = TRUE
svdCounter1 = Counter1(OAM)
svdCounter2 = Counter2(OAM)
svdCounter3 = Counter3(OAM)
svdCounter4 = Counter4(OAM)
done ?
MI_LMo_ReportError(Timeout)
MI_DMo_ReportError(Timeout)
MI_LMo_Result(N_TF, N_LF, F_TF, F_LF)
MI_DMo_Result(count, B_FD[],
F_FD[], N_FD[])
Stop(Timer)
Stop(TimeoutTimer)
Init
done = TRUE
Stop(Timer)
SQI(OAM) != NULL &&
SQI(OAM) > Period ?
Period = SQI(OAM)
SQI = Period
MI_LMo_PeriodChanged
MI_DMo_PeriodChanged
if (SQI(OAM) != NULL) {
if (SQI == 0) {
SQI = Period
} else if (SQI(OAM) == SQI) {
SQI = NULL
}
}
if (QTF(OAM) != RTF(OAM)) {
if (TSFmtSupported(RPTF(OAM)) {
TimestampFormat = RPTF(OAM)
} else {
TimestampFormat = 0x3
}
}
Set(3s, TimeoutTimer)

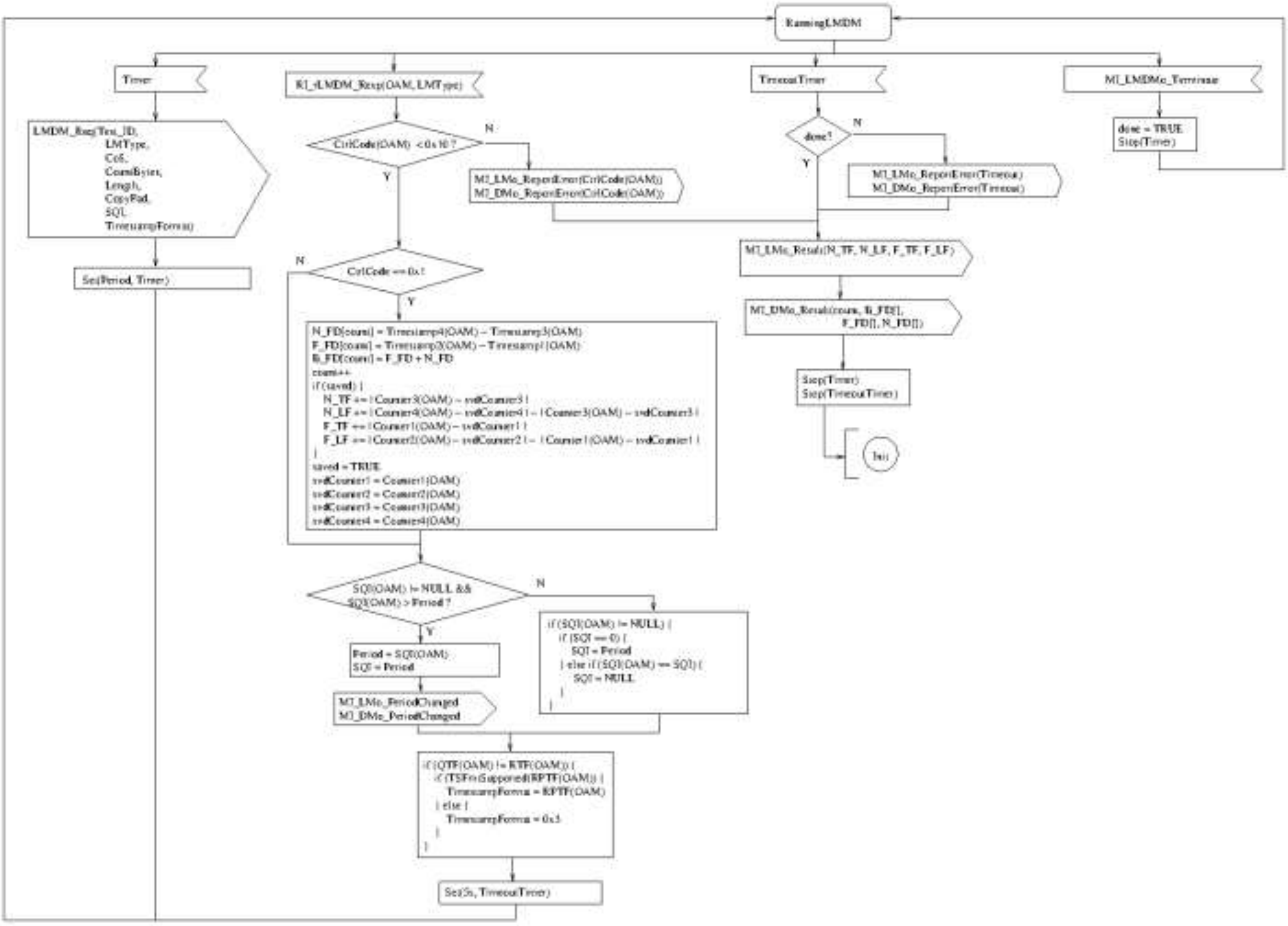
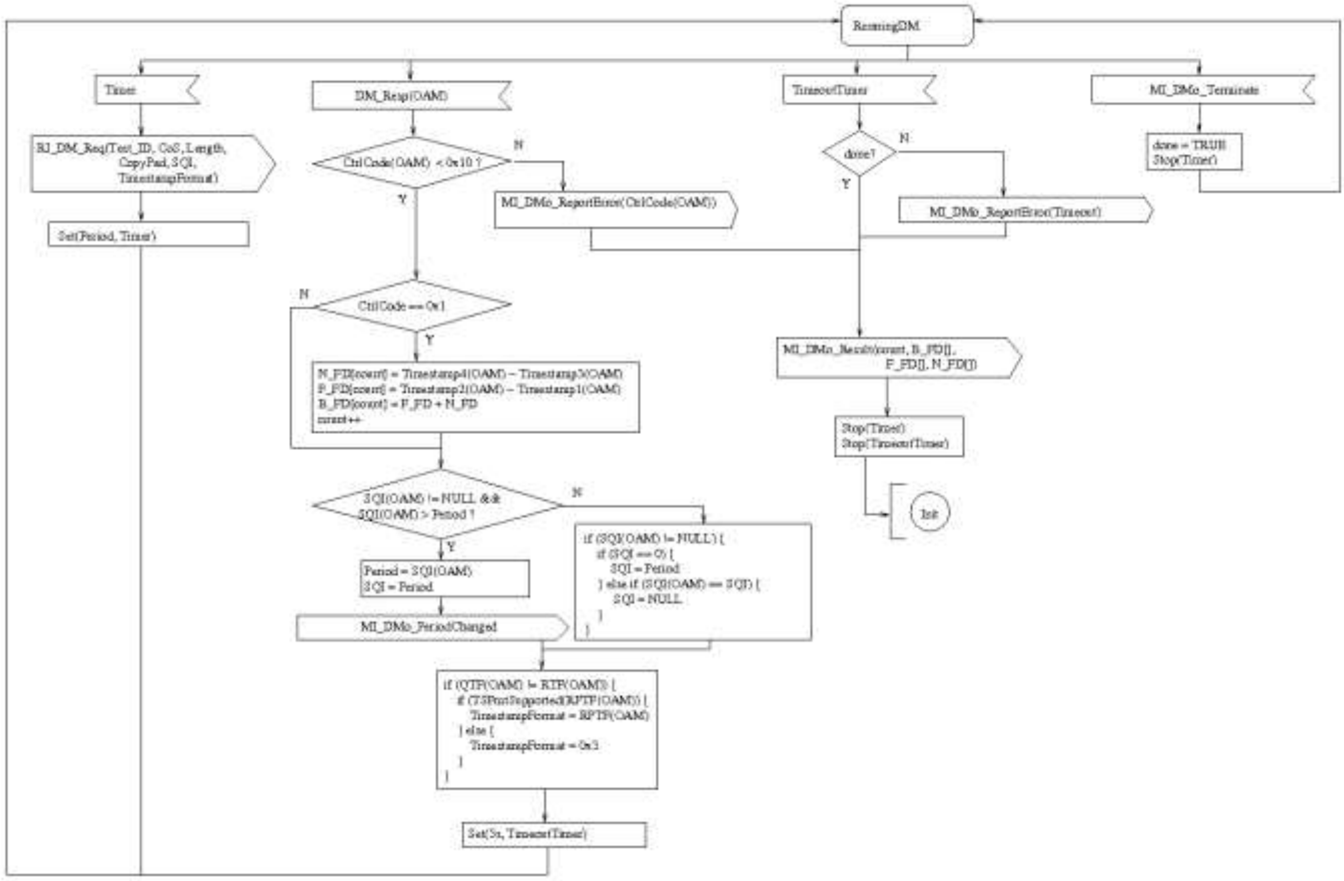


Figure 8-26/G.8121.2/Y.1381.2 On-demand PM control process


RunningDM
Timer
DM_Resp(OAM)
TimeoutTimer
MI_DMo_Terminate
RI_DM_Req(Test_ID, CoS, Length, CopyPad, SQI, TimestampFormat)
Set(Period, Timer)
CtrlCode(OAM) < 0x10 ?
N
Y
MI_DMo_ReportError(CtrlCode(OAM))
done?
MI_DMo_ReportError(Timeout)
done = TRUE
Stop(Timer)
CtrlCode == 0x1
N_FD[count] = Timestamp4(OAM) – Timestamp3(OAM)
F_FD[count] = Timestamp2(OAM) – Timestamp1(OAM)
B_FD[count] = F_FD + N_FD
count++
MI_DMo_Result(count, B_FD[], F_FD[], N_FD[])
Stop(Timer)
Stop(TimeoutTimer)
Init
SQI(OAM) != NULL &&
SQI(OAM) > Period ?
Period = SQI(OAM)
SQI = Period
if (SQI(OAM) != NULL) {
if (SQI == 0) {
SQI = Period
} else if (SQI(OAM) == SQI) {
SQI = NULL
}
}
MI_DMo_PeriodChanged
if (QTF(OAM) != RTF(OAM)) {
if (TSPortSupported(RPTF(OAM)) {
TimestampFormat = RPTF(OAM)
} else {
TimestampFormat = 0x3
}
}
Set(5s, TimeoutTimer)

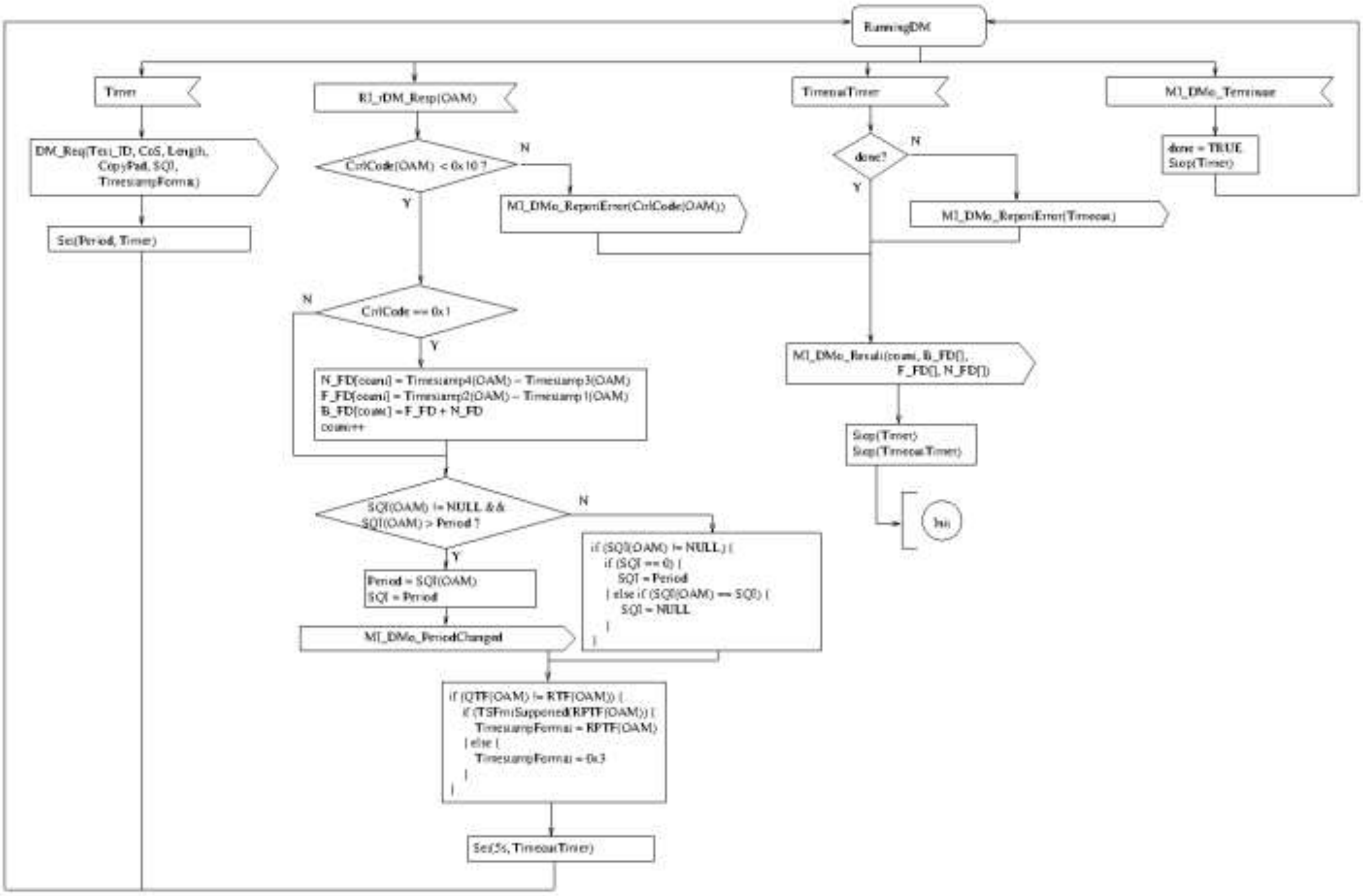
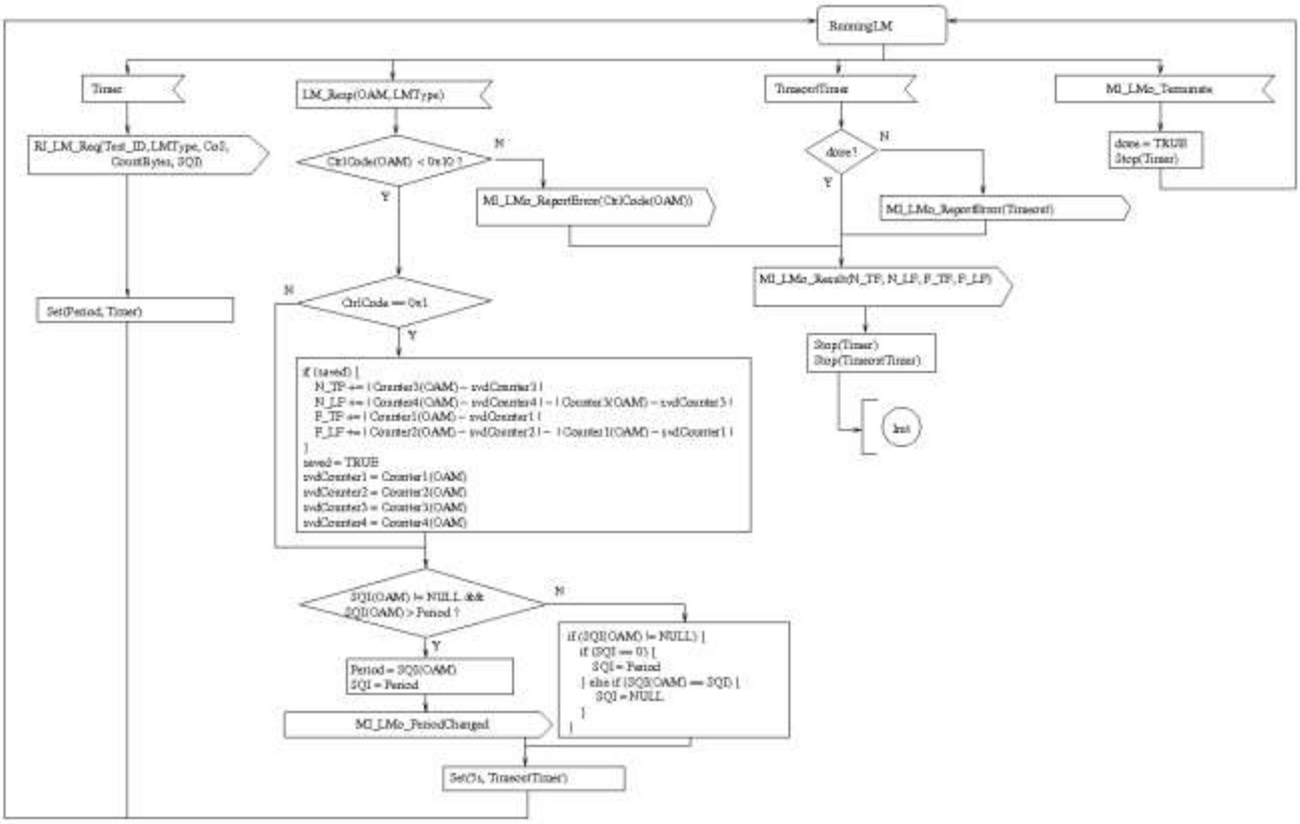


Figure 8-27/G.8121.2/Y.1381.2 On-demand PM control process


RunningLM
Timer
LM_Resp(OAM, LMType)
TimeoutTimer
MI_LMo_Terminate
RI_LM_Req(Test_ID, LMType, CoS, CountBytes, SQI)
CtrlCode(OAM) < 0x10 ?
N
Y
MI_LMo_ReportError(CtrlCode(OAM))
done ?
N
Y
MI_LMo_ReportError(Timeout)
done = TRUE
Stop(Timer)
MI_LMo_Result(N_TF, N_LF, F_TF, F_LF)
Set(Period, Timer)
N
CtrlCode == 0x1
Y
Stop(Timer)
Stop(TimeoutTimer)
Init
if (saved) {
N_TF += | Counter3(OAM) – svdCounter3 |
N_LF += | Counter4(OAM) – svdCounter4 | – | Counter3(OAM) – svdCounter3 |
F_TF += | Counter1(OAM) – svdCounter1 |
F_LF += | Counter2(OAM) – svdCounter2 | – | Counter1(OAM) – svdCounter1 |
}
saved = TRUE
svdCounter1 = Counter1(OAM)
svdCounter2 = Counter2(OAM)
svdCounter3 = Counter3(OAM)
svdCounter4 = Counter4(OAM)
SQI(OAM) != NULL && SQI(OAM) > Period ?
N
Y
Period = SQI(OAM)
SQI = Period
if (SQI(OAM) != NULL) {
if (SQI == 0) {
SQI = Period
} else if (SQI(OAM) == SQI) {
SQI = NULL
}
}
MI_LMo_PeriodChanged
Set(5s, TimeoutTimer)

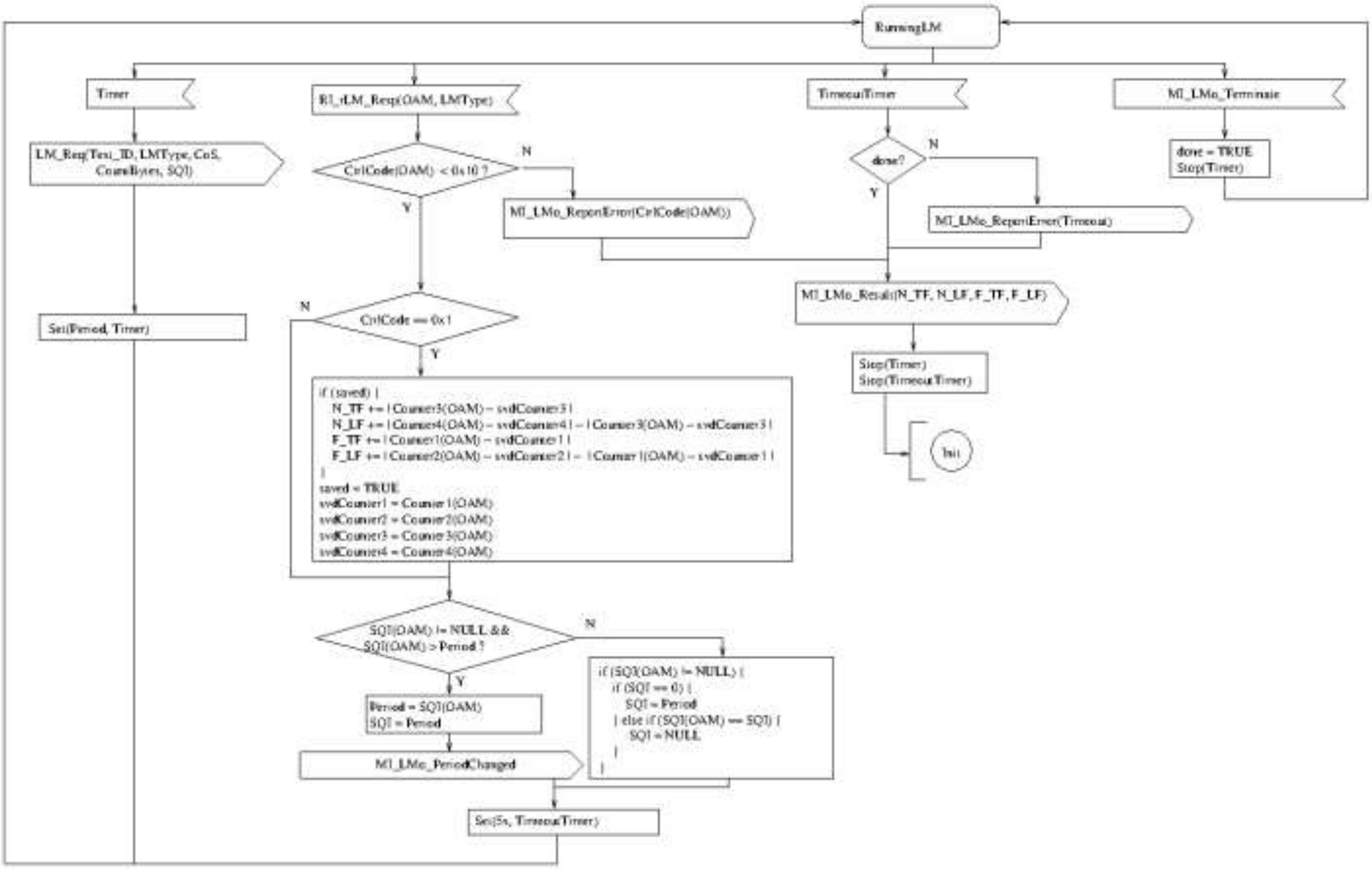


Figure 8-28/G.8121.2/Y.1381.2 On-demand PM control process

As in the proactive PM control process, the period and timestamp format are negotiated with the responder, as described in clause 8.8.4.1.

8.8.5.2 PM generation process for On-Demand LM

The PM generation process for On-demand LM is identical to that for proactive LM, and is described in clause 8.8.4.2

8.8.5.3 PM reception process for On-Demand LM

The PM reception process for On-demand LM is identical to that for proactive LM, and is described in clause 8.8.4.3

8.8.5.4 PM Responder process for On-Demand LM~~On-demand Responder process for LMp~~

The PM Responder process for On-demand LM is identical to that for proactive LM, and is described in clause 8.8.4.4

~~The On-demand Responder process is in common with that for Proactive.~~

~~See clause 8.8.4.4~~

8.8.6. Proactive Packet Delay Measurement (DMp)

As described in clauses 7.2.2.1 and 8.6 to 8.8 of [ITU-T G.8113.2], loss and delay measurements may be combined. The format for the combined measurement, refered to here as LMDM, is described in section 3.3 of [RFC 6374]. In addition, the same LM and DM protocols can be used for both proactive and on-demand measurement.

The processes for Proactive Delay Measurement are described in clause 8.8.4 in this Recommendation.

8.8.7. On-demand Packet Delay Measurement (DMo)

As described in clauses 7.2.2.1 and 8.6 to 8.8 of [ITU-T G.8113.2], loss and delay measurements may be combined. The format for the combined measurement, refered to here as LMDM, is described in section 3.3 of [RFC 6374]. In addition, the same LM and DM protocols can be used for both proactive and on-demand measurement.

The processes for On-Demand Delay Measurement are described in clause 8.8.5 in this Recommendation.

8.8.8. Throughput Test

For further study

8.8.9. Route Tracing (RT)

For further study

8.8.10. LCK/AIS Reception

The LCK/AIS ReceptionExtract Process handles received LKR and AIS packets, and signals the LCK, AIS and SSF defects. The behaviour is shown in Figure 8-xx28a.

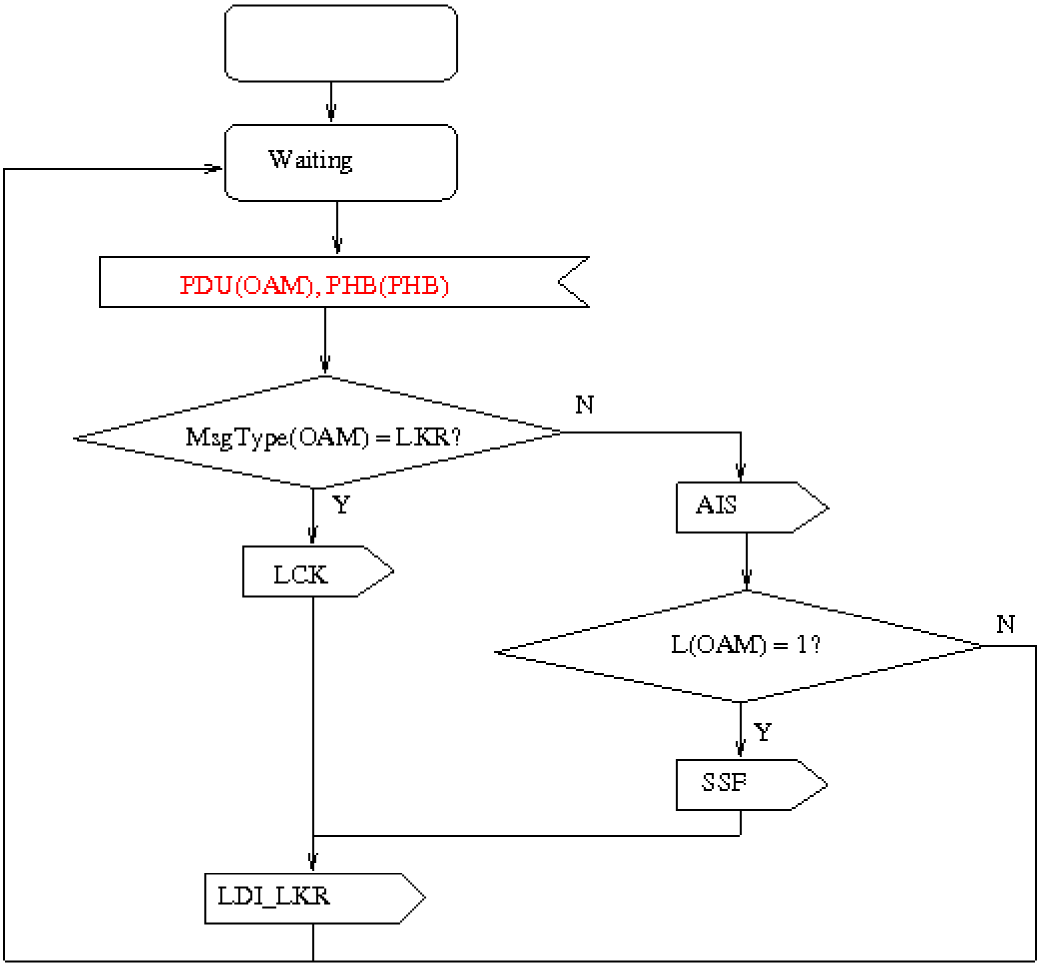


Figure 8-~~xx~~28a/G.8121.2/Y.1381.2 LCR/AIS Reception~~Extract~~ behaviour

8.8.11. Lock Instruct processes

An overview of the processes relating to the Lock Instruct mechanism is shown in Figure 8-29 ~~the figure below~~.

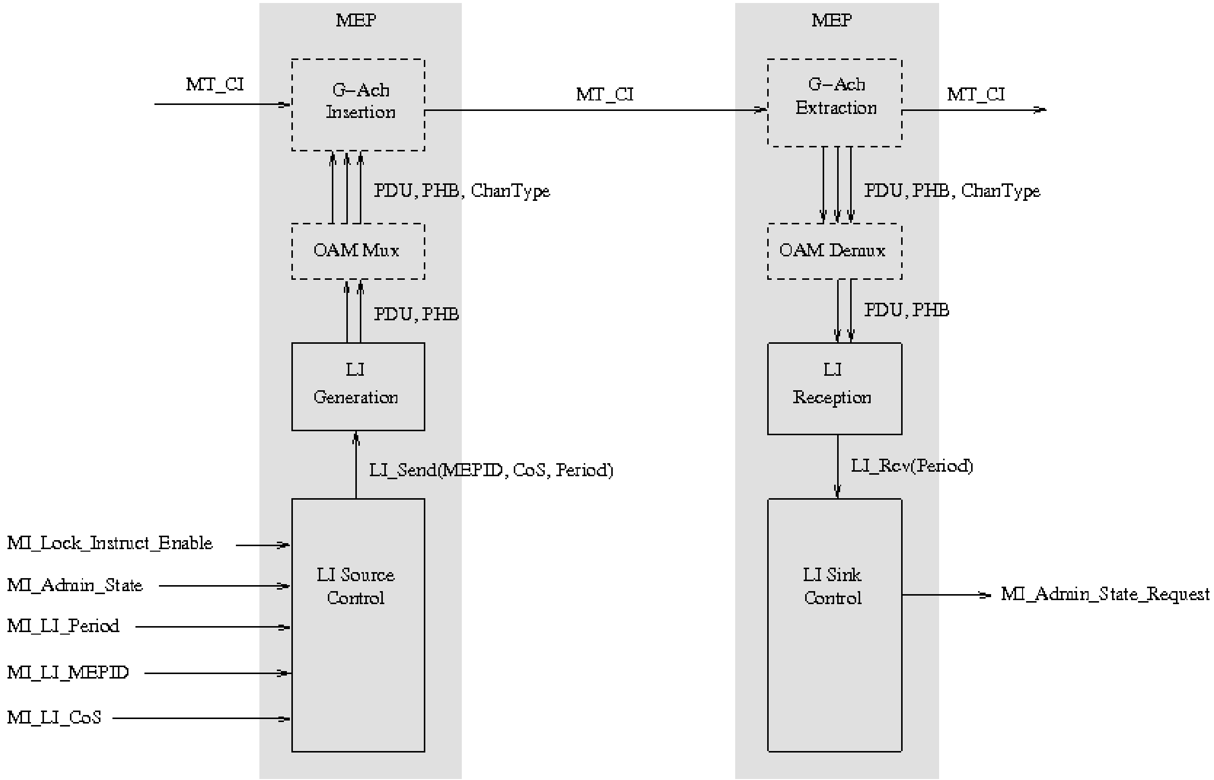


Figure 8-29x/G.8121.2/Y.1381.2 - Overview of Lock Instruct mechanism

The LI Source Control process controls sending LI messages when the admin state is "Locked" and MI_Lock_Instruct_Enable is set. The period at which to send is determined by MI_LI_Period, and the source MEP ID value is set by MI_LI_MEPID to one of the three values described in [RFC6435].

The LI Generation process formats LI messages and passes them to the OAM Mux process and hence to the G-Ach Insertion process.

The LI Reception process handles received LI messages and checks them for correctness.

The LI Sink Control process monitors received LI messages to determine whether a Lock Instruct condition exists, and signals this to the EMF via MI_Admin_State_Request.

8.8.11.1 LI Source Control Process

~~LI Source Control Process (8.8.11.1)~~

The LI Source Control Process is described in ~~the following f~~Figure 8-30.

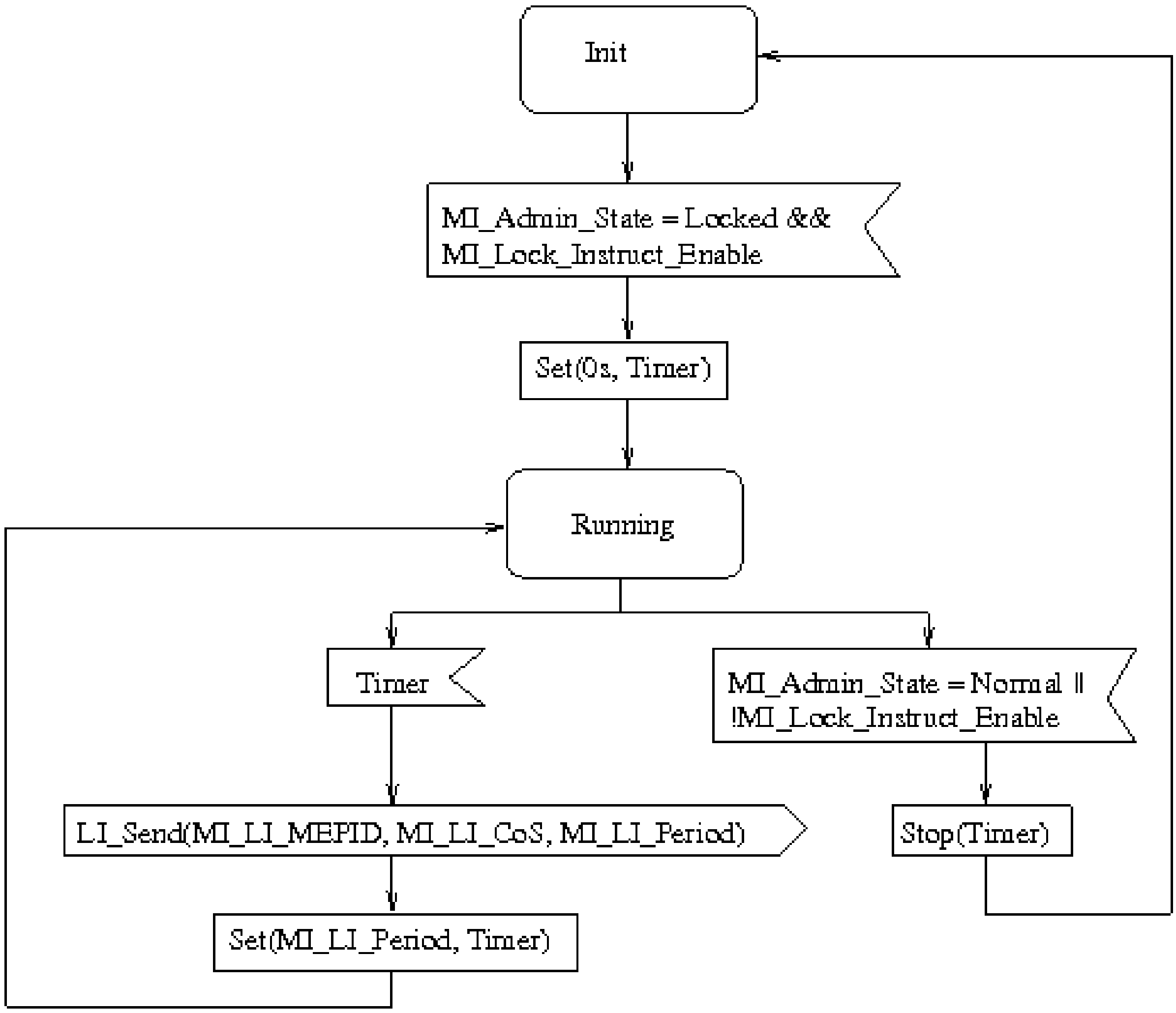


Figure 8-x30/G.8121.2/Y.1381.2 - LI Source Control Process

8.8.11.21 LI Generation Process (.2)

The LI Generation Process is described in the following fFigure 8-31.

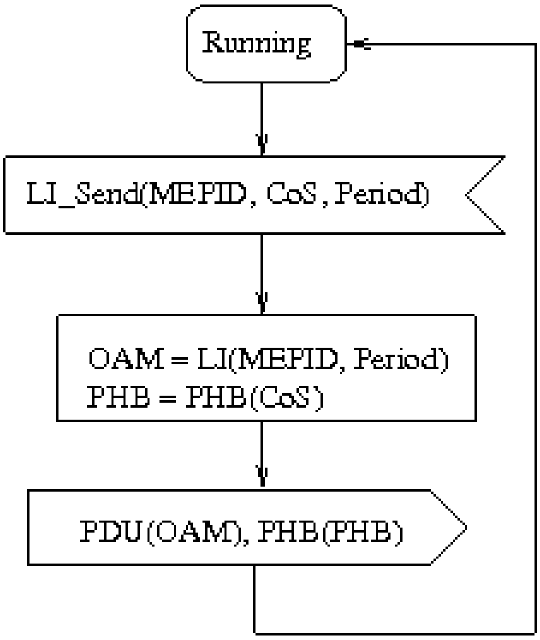


Figure 8-x31/G.8121.2/Y.1381.2 - LI Generation Process

The LI(MEPID, Period) function formats an LI PDU according to [RFC6435], as follows:

- The version is set to 1.
- The reserved field is set to 0.
- The refresh timer field is set to the Period.
- The MEPID is copied into the MEP Source ID TLV.

The PHB(CoS) function returns the PHB with the lowest drop precedence within the Class of Service defined by the CoS input parametered0ed02

8.8.11.2 LI Source Control Process

The LI Generation Process is described in the following fFigure 8-31.

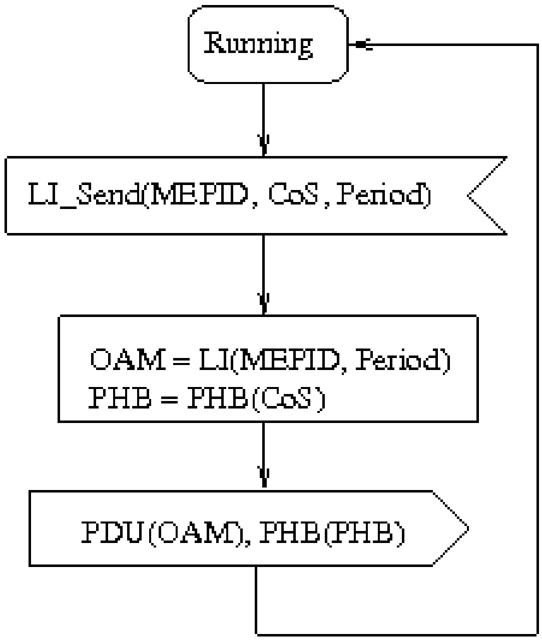


Figure 8-x31/G.8121.2/Y.1381.2 – LI Generation Process

The LI(MEPID, Period) function formats an LI PDU according to [RFC6435], as follows:

- The version is set to 1.
- The reserved field is set to 0.
- The refresh timer field is set to the Period.
- The MEPID is copied into the MEP Source ID TLV.

The PHB(CoS) function returns the PHB with the lowest drop precedence within the Class of Service defined by the CoS input parameterced0cd02

8.8.11.3 LI Reception Process

The LI Reception Process is described in the following fFigure 8-32.

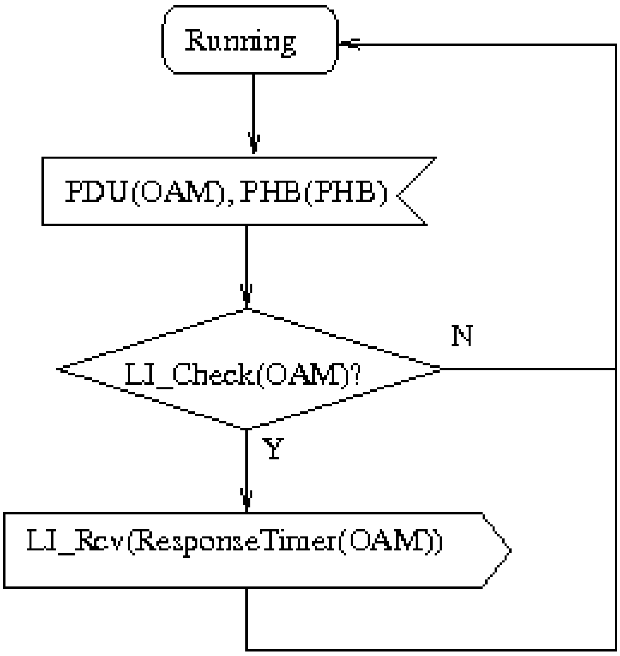


Figure 8-32x/G.8121.2/Y.1381.2 - LI Reception Process

The LI_Check(OAM) function performs implementation-specific checks, including those described in [RFC6435], and returns true if the OAM is valid and false otherwise.ed02

8.8.11.4 LI Sink Control Proces

The LI Sink Control process is described in Figure 8-x33.

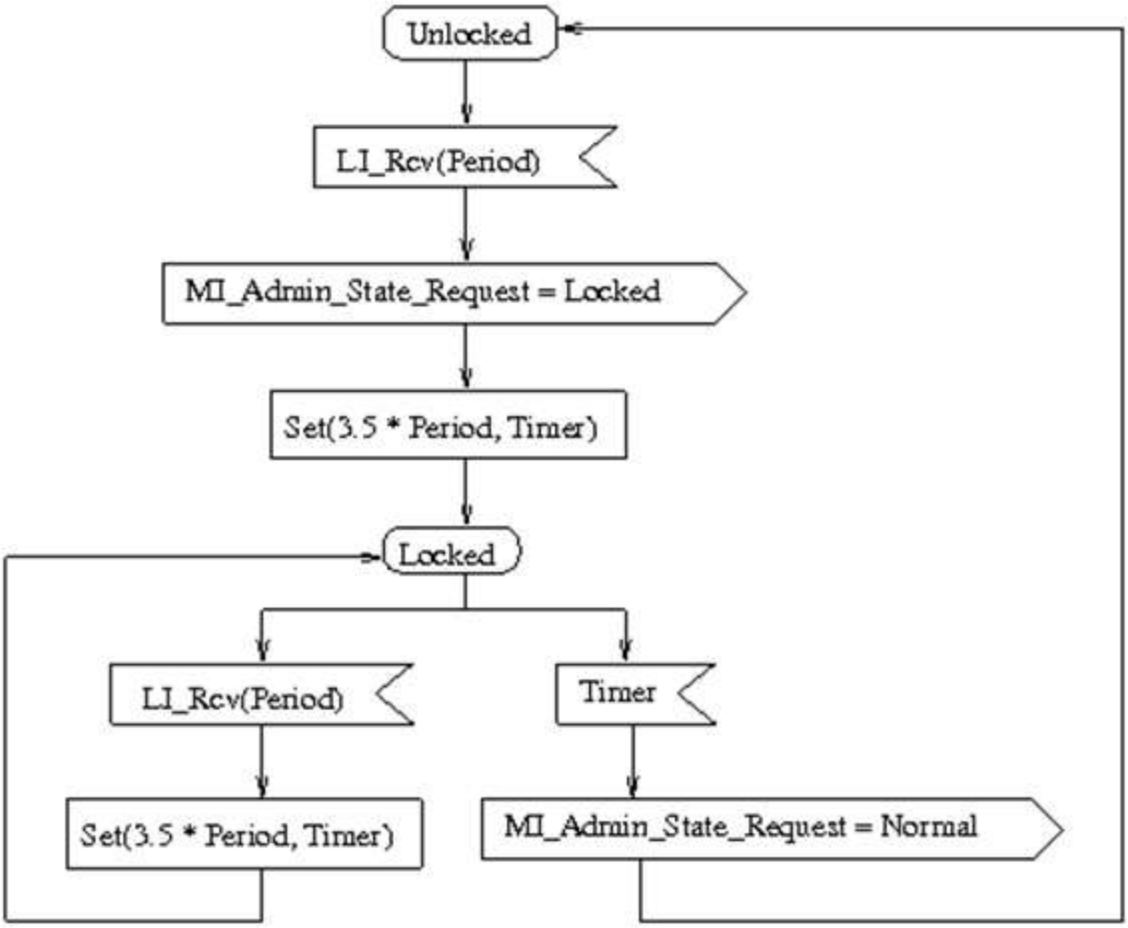


Figure 8-x33/G.8121.2/Y.1381.2 - LI Sink Control Process

See clause 8.8.11 in [ITU-T G.8121] ()

An overview of the performance monitoring processes for a single PM session is shown in the figure below: Either the proactive PM control process or the on-demand PM control process is used, to perform proactive or on-demand measurements respectively. For simplicity, both are shown in the figure below.

Either the Proactive or On-demand PM control process controls the session, including scheduling request packets, and processing responses to calculate performance metrics.

The PM generation process generates requests and responses for the five different types of PM PDUs: ILM, DLM, DM, ILM+DM and DLM+DM. It also counts data traffic (including test packets) and is responsible for writing the counters and/or timestamps into the outgoing PM PDUs. The location of the counter part is shown on the figure above for illustration only; the exact set of packets to be counted is implementation-specific, as described in [RFC6374].

The PM reception process handles received requests and responses. Like the PM generation process, it counts the appropriate packets and writes the counters and/or timestamps into the received PM PDUs. Again, the location of the counter part is shown on the figure above for illustration only; the exact set of packets to be counted is implementation-specific, as described in [RFC6374].

The PM responder is responsible for replying to received PM request packets.

Multiple PM sessions can be used simultaneously, by instantiating multiple instances of the PM control, PM reception, PM generation and PM responder processes. Each PM session (proactive or on-demand) must have a unique test ID. For each test ID, the control process is associated with a corresponding instance of the PM reception and PM generation processes. Similarly, the responder process for a given session is associated with a corresponding instance of the PM reception and PM generation processes. The PM Mux process multiplexes PM packets for different sessions, while the PM Demux process demultiplexes them based on the Test ID (Session ID) and R (response) flag.

Note therefore that a given instance of the PM reception process is associated with exactly one other process to which it passes received packets. Depending on how it is instantiated, this could be the proactive PM control process, the on-demand PM control process, or the PM responder process.

8.8.3.1. Proactive PM Control Process

The Proactive PM Control process operates a single proactive PM session. Multiple sessions can be supported by instantiating multiple instances of the process, along with corresponding instances of the PM generation and PM reception processes.

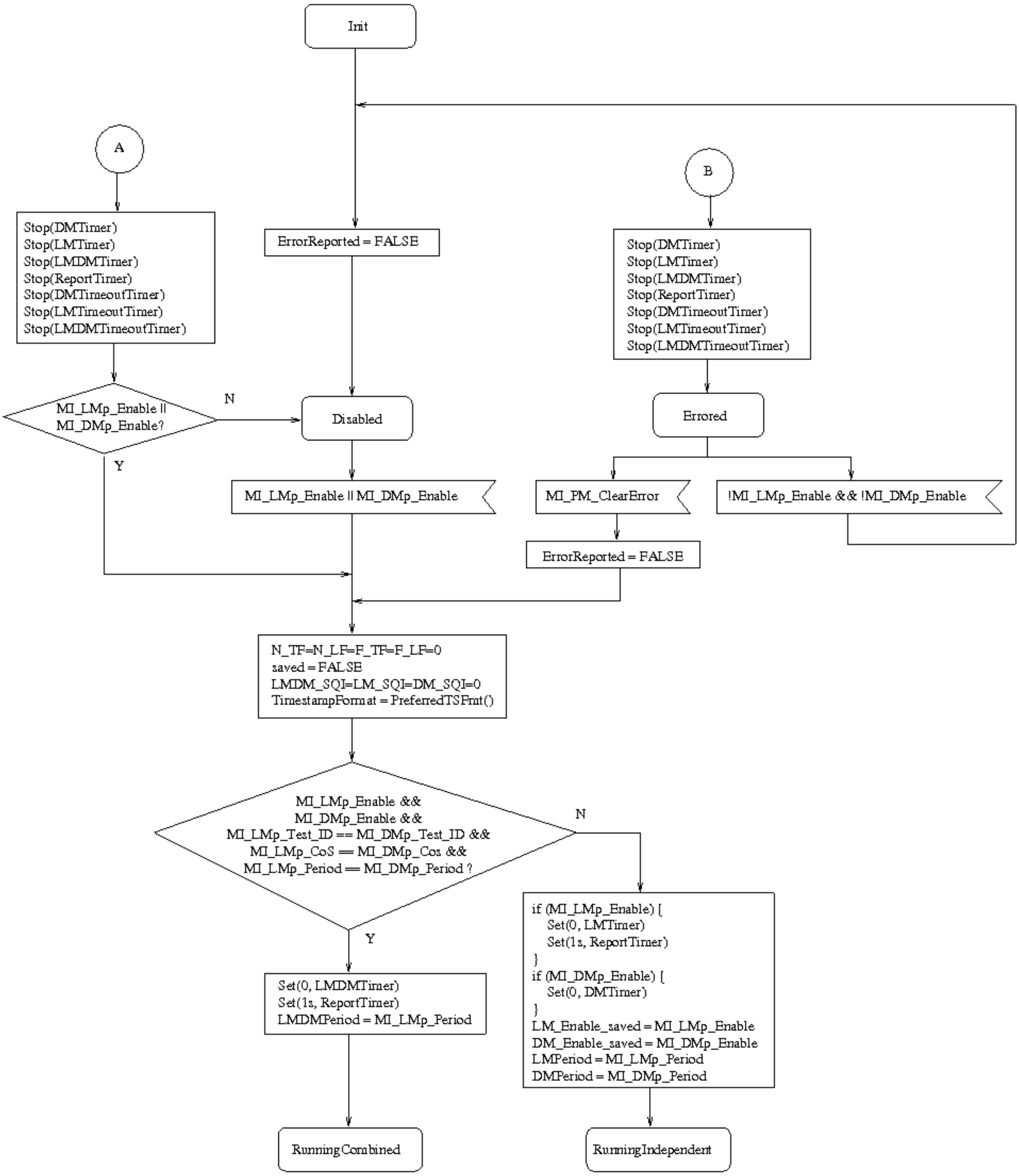
The proactive PM control process performs delay measurements when MI_DMp_Enable is true, and performs loss measurements when MI_LMp_Enable is true. If both are enabled, then where possible, the same PDUs are used to make both measurements (ie ILM+DM or DLM+DM PDUs). Otherwise, separate PDUs are used for loss (ILM or DLM) and delay (DM). The type of PDU used for loss is determined by MI_LMp_LMType, and can be “ILM” for inferred (synthetic) loss or “DLM” for direct (data traffic) loss measurement.

If an error is detected while the session is running, this is reported via MI_DMp_ReportError or MI_LMp_ReportError, and the session is disabled until MI_PM_ClearError is set.

The PM protocol includes a mechanism to negotiate the packet sending period with the responder. If the period is changed from that specified by the management information (MI_DMp_Period or MI_LMp_Period), this is signalled via MI_DMp_PeriodChanged or MI_LMp_PeriodChanged.

MI_LMp_CoS and MI_DMp_CoS specify the CoS (traffic class) to use for the measurement. In the case of MI_LMp_CoS, this can either be a specific value, or the special value “ALL” indicating that loss across all traffic classes should be measured.

The proactive PM control process is described in the following three figuresFigure 8-19, 8-20, and 8-21.



~~Figure 8-x~~19/G.8121.2/Y.1381.2 ~~proactive PM control process~~

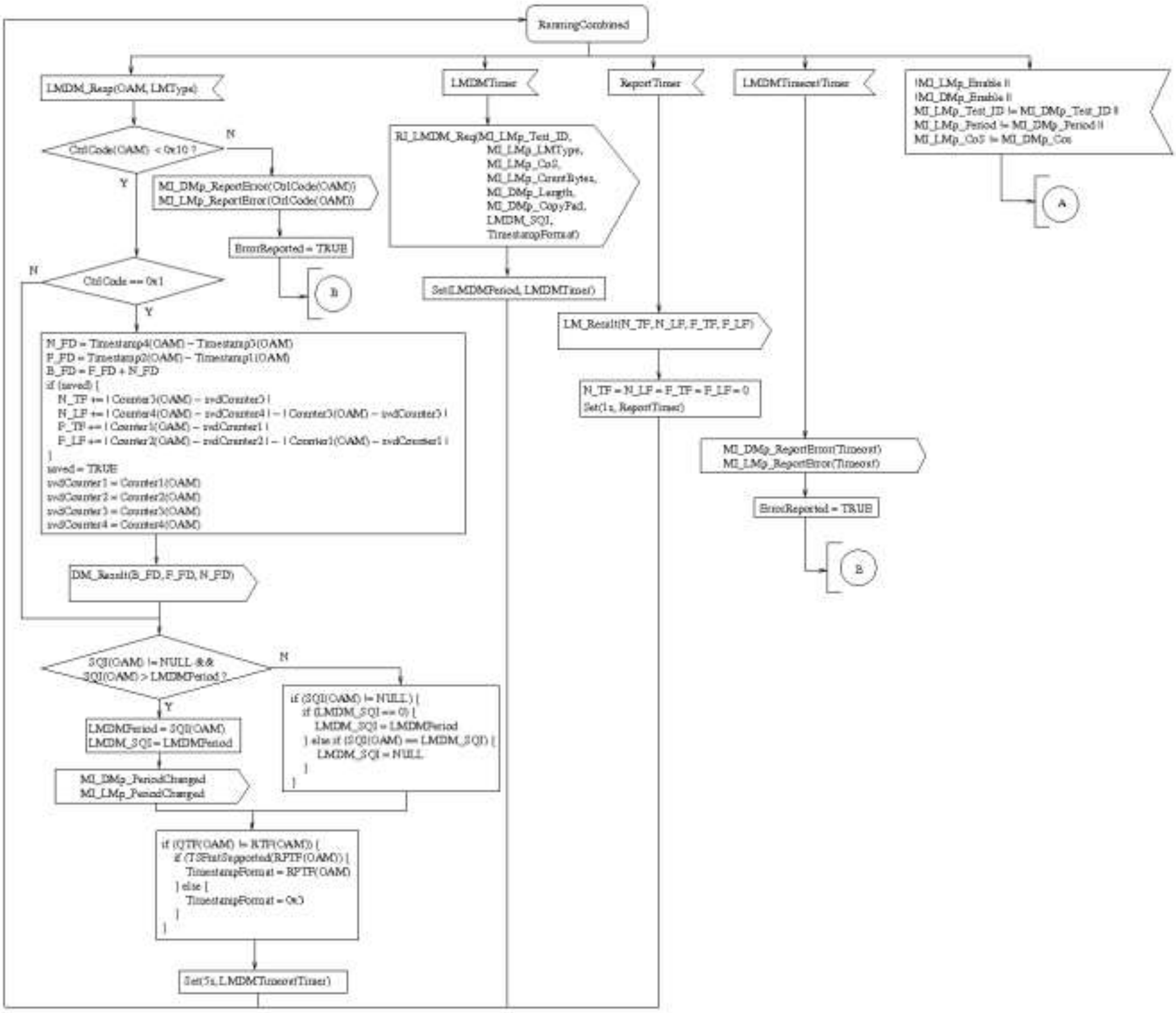


Figure 8-x20/G.8121.2/Y.1381.2 proactive PM control process

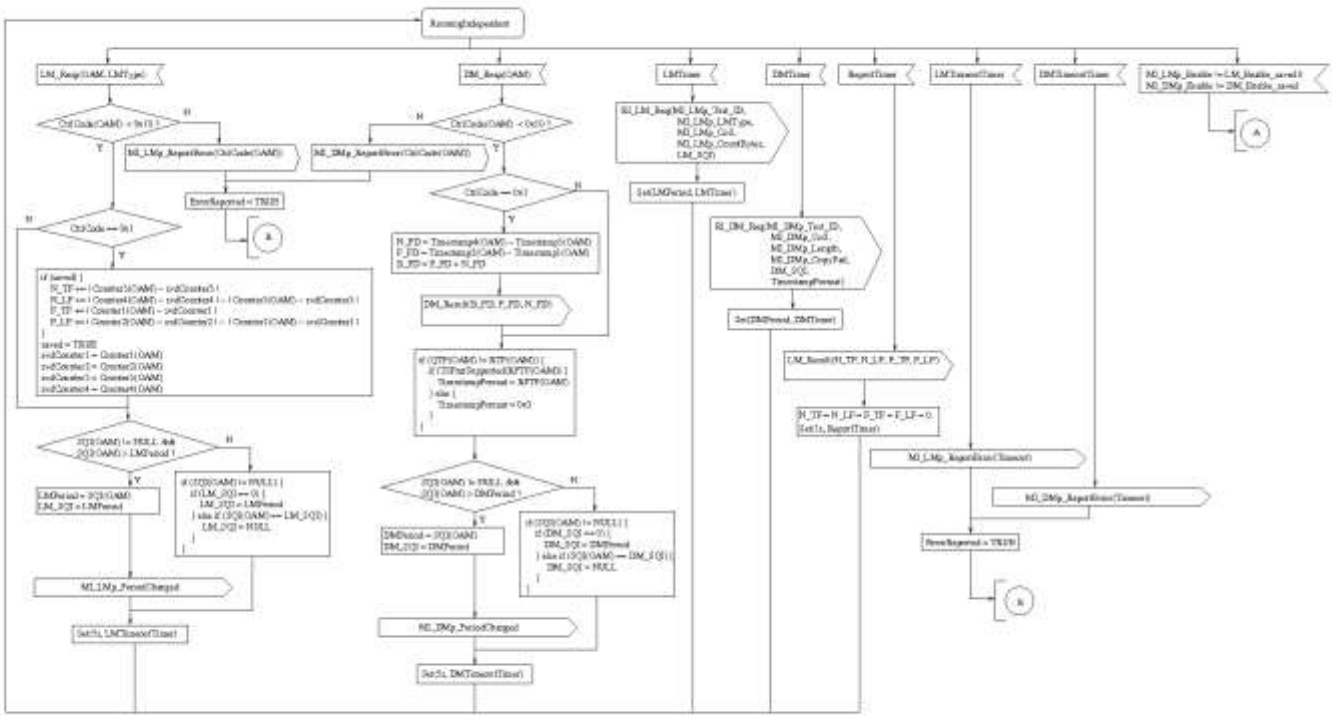

LMTimer
DMTimer
ReportTimer
LMTimeoutTimer
DMTimeoutTimer
ErrorReported = TRUE
A

Figure 8-x21/G.8121.2/Y.1381.2 proactive PM control process

The TSFmtSupported() function determines whether the specified timestamp format, from [RFC 6374], is supported by the implementation, while the PreferredTSFmt() function returns the timestamp format that is preferred by the implementation, as described in [RFC 6374].

Note that both the period and the timestamp format are negotiated with the responder. The period is negotiated by setting the SQI appropriately, while the timestamp format is negotiated via the QTF, RTF and RPTF fields. Initially, the implementation's preferred timestamp is used. If the responder does not respond to the first request using the same timestamp format, then the responder's preferred timestamp format is used if it is supported, otherwise the IEEE 1588v1 format is used as described in [RFC6374]. Note that support for this format is mandatory.

8.8.3.2. On-Demand PM Control Process

The On-Demand PM Control process operates a single on-demand PM session. Multiple sessions can be supported by instantiating multiple instances of the process, along with corresponding instances of the PM generation and PM reception processes.

The on-demand PM control process performs either delay measurement (via MI_DMo_Start/MI_DMo_Terminate), loss measurement (via MI_LMo_Start/MI_LMo_Terminate) or both simultaneously (via MI_LMDMo_Start/MI_LMDMo_Terminate). The type of loss measurement to perform is specified by the LMType parameter, and can be "ILM" for inferred (synthetic) loss or "DLM" for direct (data traffic) loss.

Results are reported via MI_DMo_Result and MI_LMo_Result.

If an error is detected while the session is running, this is reported via MI_DMo_ReportError or MI_LMo_ReportError, and the session is terminated automatically. The results collected up to that point are reported.

The PM protocol includes a mechanism to negotiate the packet sending period with the responder. If the period is changed from that specified when the session was started, this is signalled via MI_DMo_PeriodChanged or MI_LMo_PeriodChanged.

The CoS parameter of MI_LMo_Start, MI_DMo_Start or MI_LMDMo_Start specifies the CoS (traffic class) to use for the measurement. In the case of MI_LMo_Start, this can either be a specific value, or the special value "ALL" indicating that loss across all traffic classes should be measured.

The on-demand PM control process is described in Figure 8-22, Figure 8-23, Figure 8-24 and Figure 8-25the following four figures.

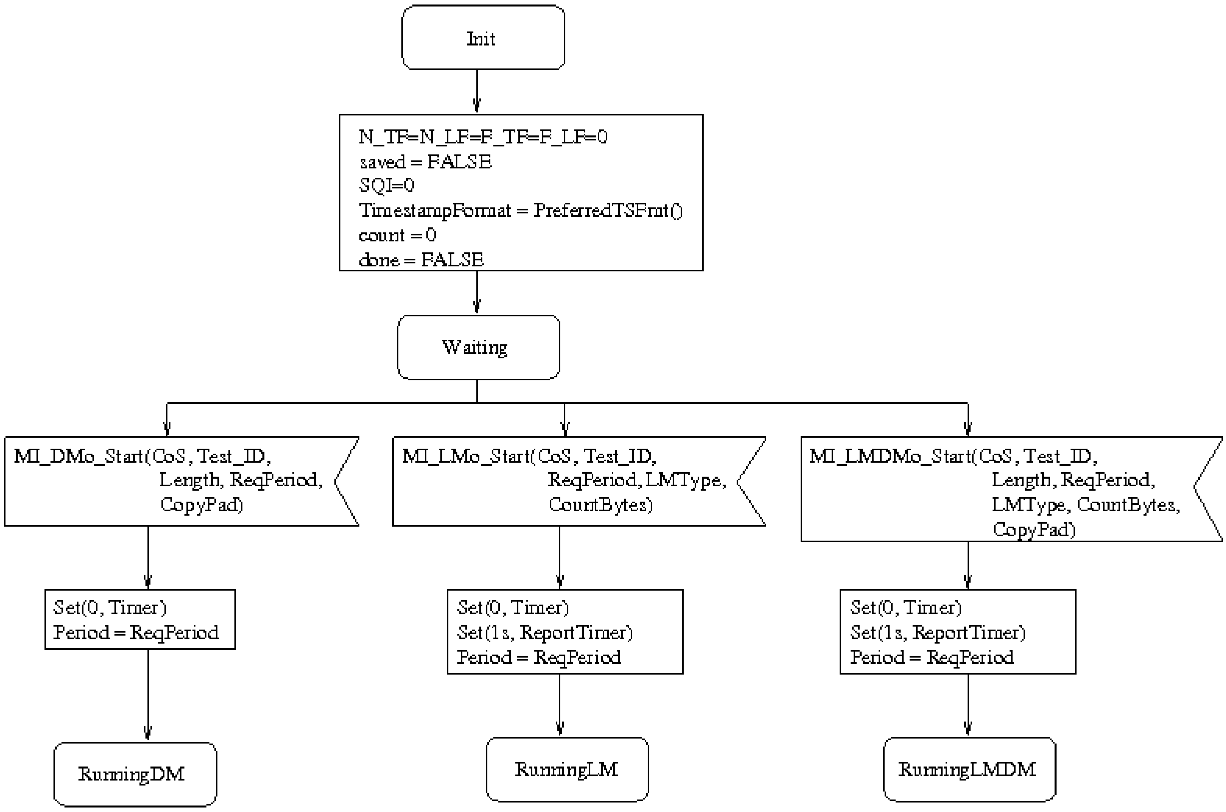


Figure 8-22x/G.8121.2/Y.1381.2 On-demand PM control process

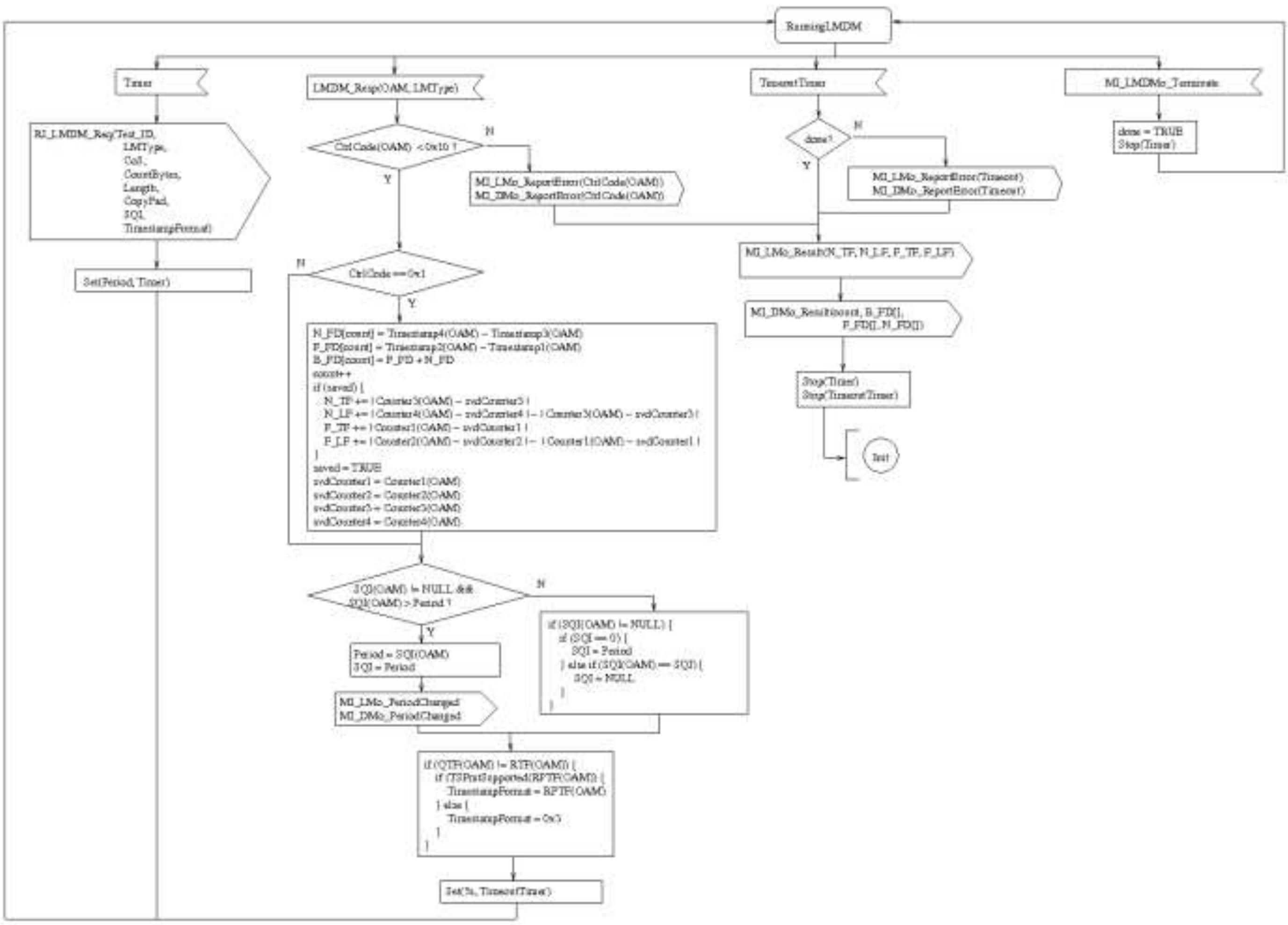
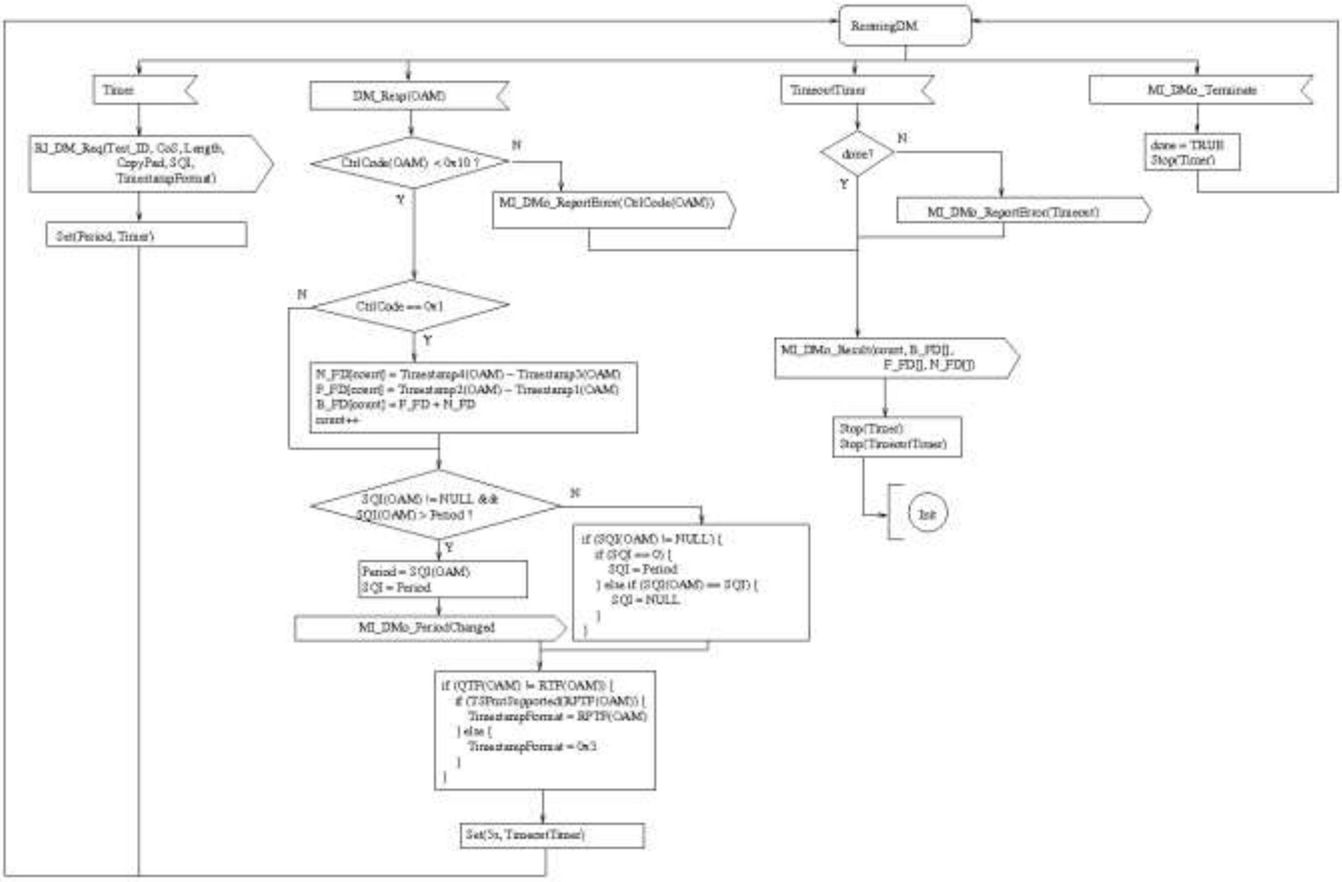


Figure 8-23x/G.8121.2/Y.1381.2 On-demand PM control process



~~Figure 8-24x/G.8121.2/Y.1381.2 On-demand PM control process~~

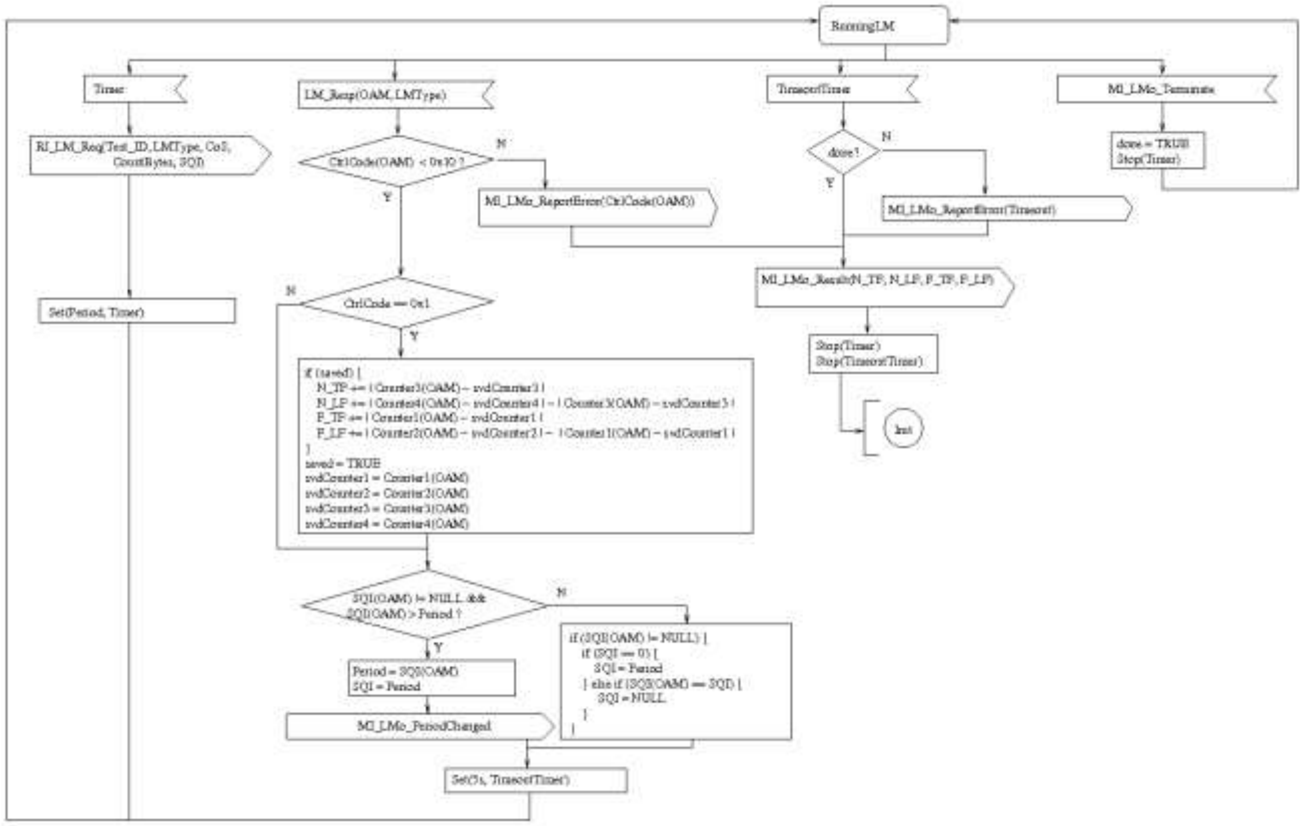


Figure 8-x25/G.8121.2/Y.1381.2 On-demand PM control process

As in the proactive PM control process, the period and timestamp format are negotiated with the responder, as described in the previous section.

8.8.3.3. PM Generation Process

The PM Generation process generates PM requests when it receives the RI_DM_Req, RI_LM_Req or RI_LMDM_Req signals from the corresponding Proactive or On-demand control process, and generates PM responses when it receives the RI_DM_Resp, RI_LM_Resp or RI_LMDM_Resp signals from the corresponding PM responder process.

For delay measurement, it writes the packet send time into the PDU, using the requested timestamp format.

For loss measurement, it counts the appropriate traffic depending on the type of loss measurement, and writes the counters into the transmitted PM PDUs. The packets to count are dependent on the LMType (ILM or DLM) and the CoS (which may be a particular value, or the special value "ALL"). If the CountBytes parameter is set, the number of bytes in each matching packet is counted, otherwise the count is simply incremented for each matching packet.

In the RI_DM_Req, RI_LM_Req and RI_LMDM_Req signals, the SQI parameter specifies the value to place in the SQI TLV. If it is set to NULL, no SQI TLV is included. The TSFmt parameter specifies the timestamp format to use when writing timestamps. The Length parameter specifies the length of padding to include in the PDU. If set to 0, no Padding TLV is included.

The PM generation process is described in Figure 8-xx26:

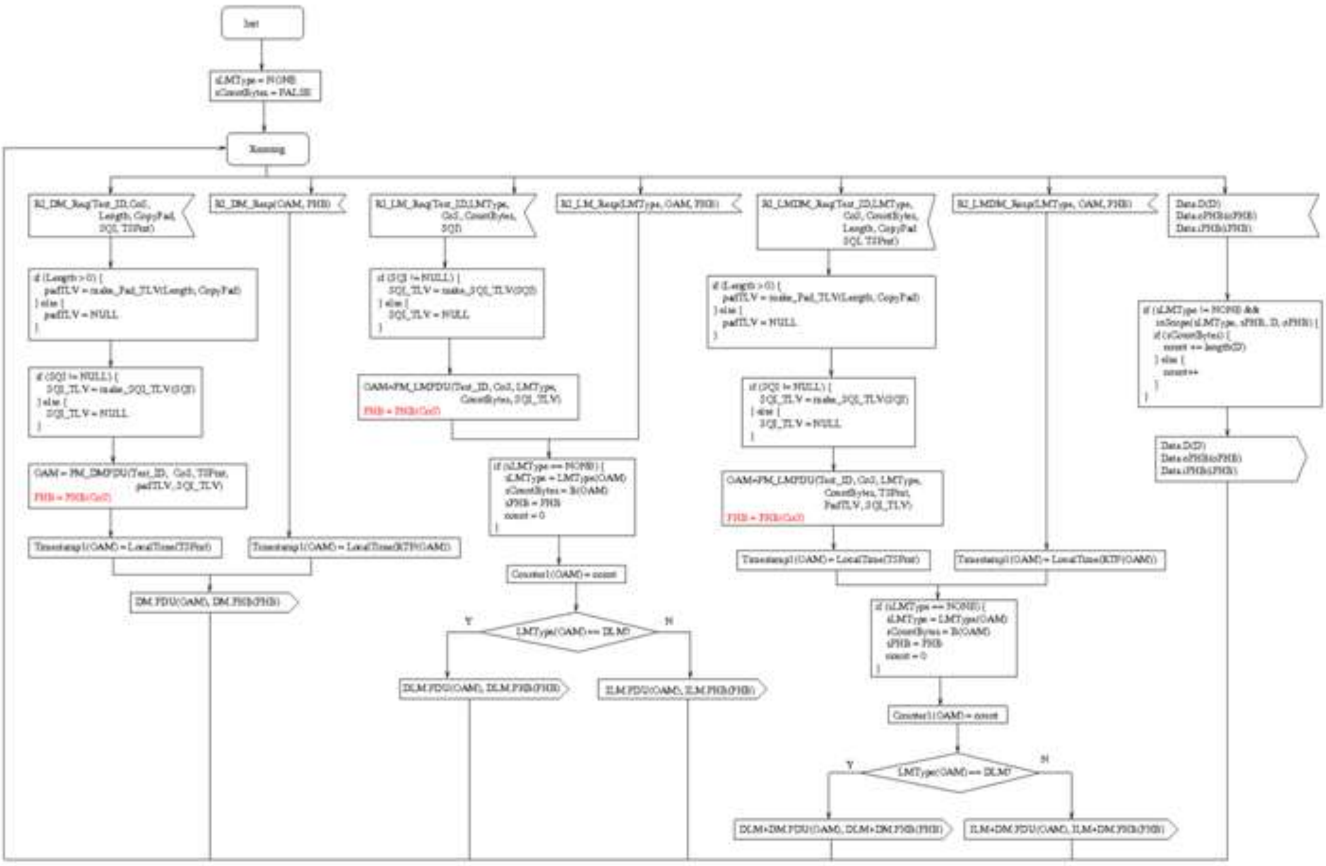


Figure 8-xx26/G.8121.2//Y.1381.2 – PM Generation Process
[updated per Annex IX, C.2025]

The make_Pad_TLV(Length, CopyPad) function creates a Padding TLV as specified in [RFC6374], as follows:

- If CopyPad is set, the Type is set to 0, otherwise it is set to 128.
- The Length field is set to Length
- The Value field is set to all 0s.

The make_SQI_TLV(SQI) function creates an SQI TLV as specified in [RFC6374], as follows:

- The Type is set to 2
- The Length field is set to 4.
- The Value field is set to SQI.

The PM_DMPDU(Test_ID, TSFmt, CoS, padTLV, SQI_TLV) function creates a DM PDU as specified in [RFC6374], as follows:

- The version is set to 0
- The R flag is unset; the T flag is set; and the rest of the flags field is set to 0.
- The control code is set to 0. Other values for the control code are FFS.

- The message length is set to the total length of the PDU.
- The QTF field is set to TSFmt, the RTF and RPTF fields are set to 0
- The reserved field is set to 0.
- The session ID and DS fields are set to Test_ID and CoS respectively.
- The timestamp fields are all set to 0.
- The pad TLV and SQI TLV, if not NULL, are appended to the message. The use of other TLVs is FFS.

The PM_LMPDU() function creates an ILM or DLM PDU as specified in [RFC6374], as follows:

- The version is set to 0
- The R flag is unset; the T flag is set if a specific CoS value has been specified and is unset if the CoS is set to "ALL"; and the rest of the flags field is set to 0.
- The control code is set to 0. Other values for the control code are FFS.
- The message length is set to the total length of the PDU.
- In the Dflag field, the X flag is set appropriate depending whether the implementation writes 32 or 64 bit counters; the B flag is set if CountBytes is set, and is unset otherwise; and the rest of the field is set to 0.
- The OTF field is set to the implementations preferred timestamp format.
- The reserved field is set to 0.
- The session ID and DS fields are set to Test_ID and CoS respectively. If the CoS is "ALL", the DS field is set to 0.
- The origin timestamp field is set to the local time-of-day, using the format specified in the OTF field.
- The counter fields are all set to 0.
- The SQI TLV, if not NULL, is appended to the message. The use of other TLVs is FFS.

The PHB(CoS) function returns the PHB with the lowest drop precedence within the Class of Service defined by the CoS input parameter.

The PM_LMDMPDU() function creates an ILM+DM or DLM_DM PDU as specified in [RFC6374], in a similar way to the DM and LM cases described above.

The InScope() function determines whether a given data packet should be counted, depending on the LM Type (ILM or DLM) and the CoS/PHB (a specific TC value or "ALL").

The LocalTime(TSFmt) function returns the local time-of-day, in the format specified.

8.8.3.4. PM Reception Process

The PM Reception process receives PM message for a given Test ID, and passes them to the corresponding Proactive or On-demand control process or PM Responder process.

For delay measurement, it writes the packet receive time into the PDU. For loss measurement, it counts the appropriate traffic depending on the type of loss measurement, and writes the counters into the received PM PDUs. The packets to count are dependent on the LMType (ILM or DLM) and the CoS (which may be a particular value, or the special value "ALL"). If the CountBytes bit is set, the number of bytes in each matching packet is counted, otherwise the count is simply incremented for each matching packet.

The PM reception process is described in Fthe following figure 8-27:

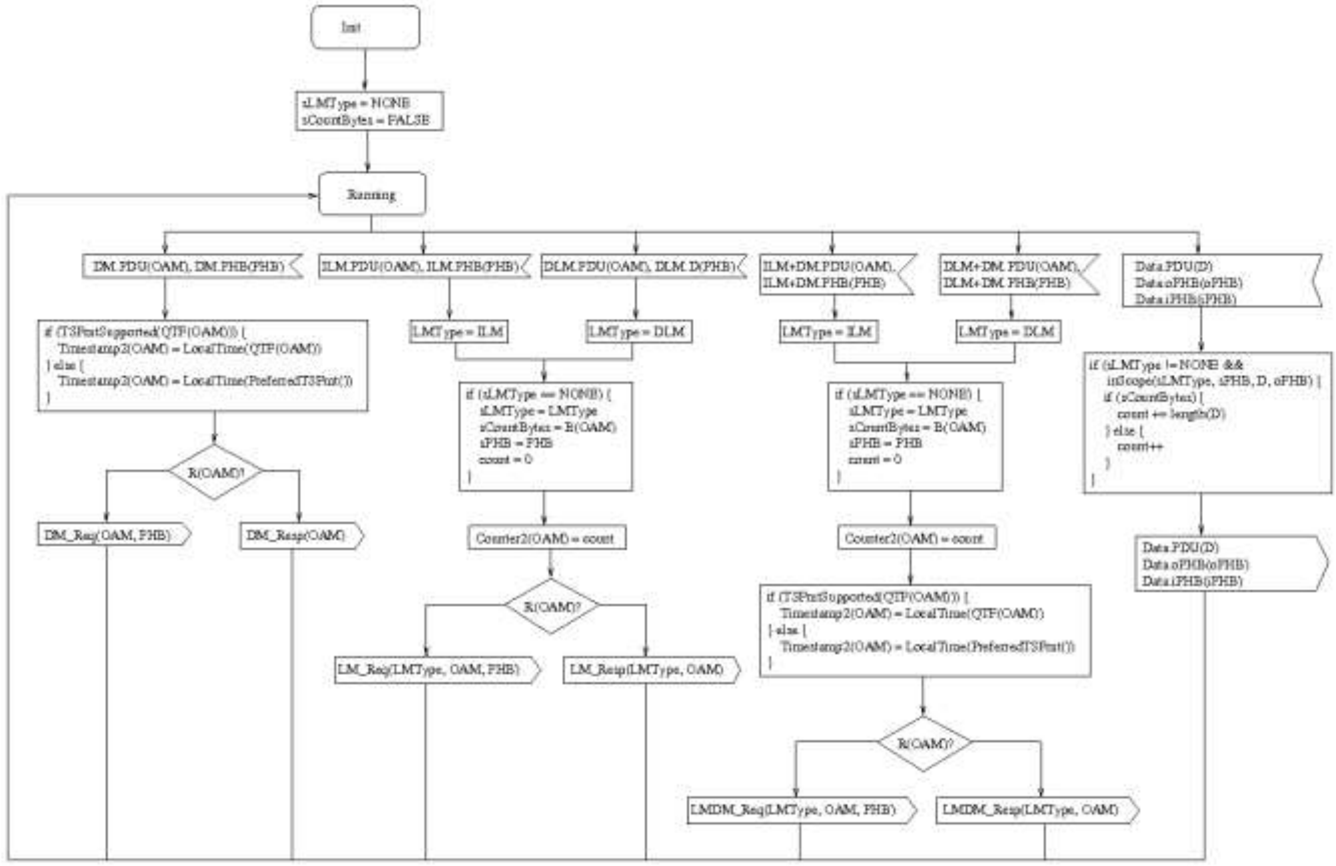


Figure 8-x27/G.8121.2/Y.1381.2 – PM reception process

8.8.3.5. PM Responder Process

The PM responder process responds to PM messages for a single PM session. Multiple sessions can be supported by instantiating multiple instances of the process, along with corresponding instances of the PM generation and PM reception processes.

The PM responder process is described in Fthe following figure 8-28:

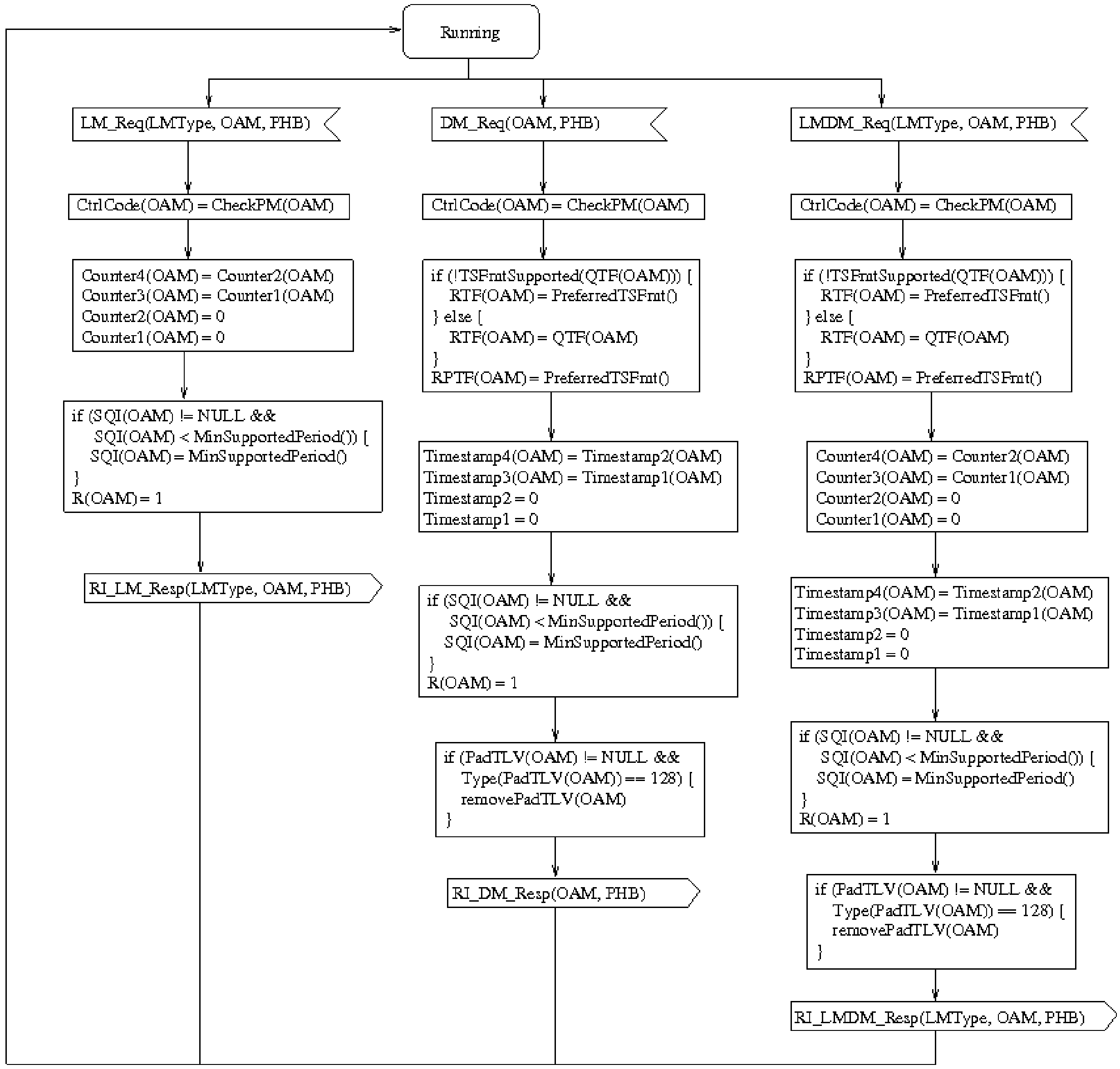


Figure 8-x28/G.8121.2/Y.1381.2 PM responder process

The CheckPM() function checks the received PDU and returns an appropriate control code, as described in [RFC6374]. In particular, it returns 0x19 (Administrative Block) if MI_PM_Responder_Enable is not set, and 0x2 (Data Format Invalid) if the QTF in a DM, ILM+DM or DLM_DM message is not supported.

Note that when MI_PM_Responder_Enable is not set, responses are still sent, with the above error.

The PM responder process also unsets the X flag in LM messages if the implementation does not support 64 bit counters.

8.8.4. Lock Instruct Processes *[updated per cd02]*

An overview of the processes relating to the Lock Instruct mechanism is shown in Figure 8-29 the figure below.

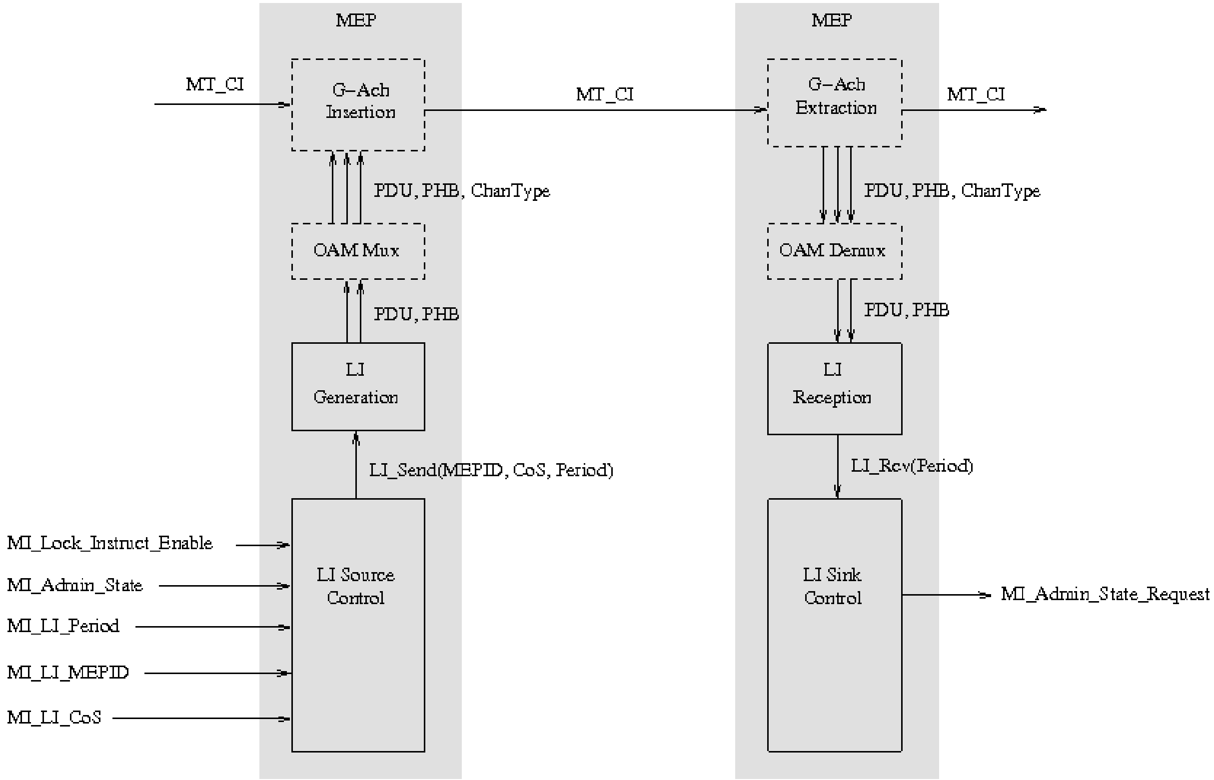


Figure 8-29x/G.8121.2/Y.1381.2 - Overview of Lock Instruct mechanism

The LI Source Control process controls sending LI messages when the admin state is "Locked" and MI_Lock_Instruct_Enable is set. The period at which to send is determined by MI_LI_Period, and the source MEP ID value is set by MI_LI_MEPID to one of the three values described in [RFC6435].

The LI Generation process formats LI messages and passes them to the OAM Mux process and hence to the G-Ach Insertion process.

The LI Reception process handles received LI messages and checks them for correctness.

The LI Sink Control process monitors received LI messages to determine whether a Lock Instruct condition exists, and signals this to the EMF via MI_Admin_State_Request.

8.10.5.1 LI Source Control Process

The LI Source Control Process is described in the following fFigure 8-30.

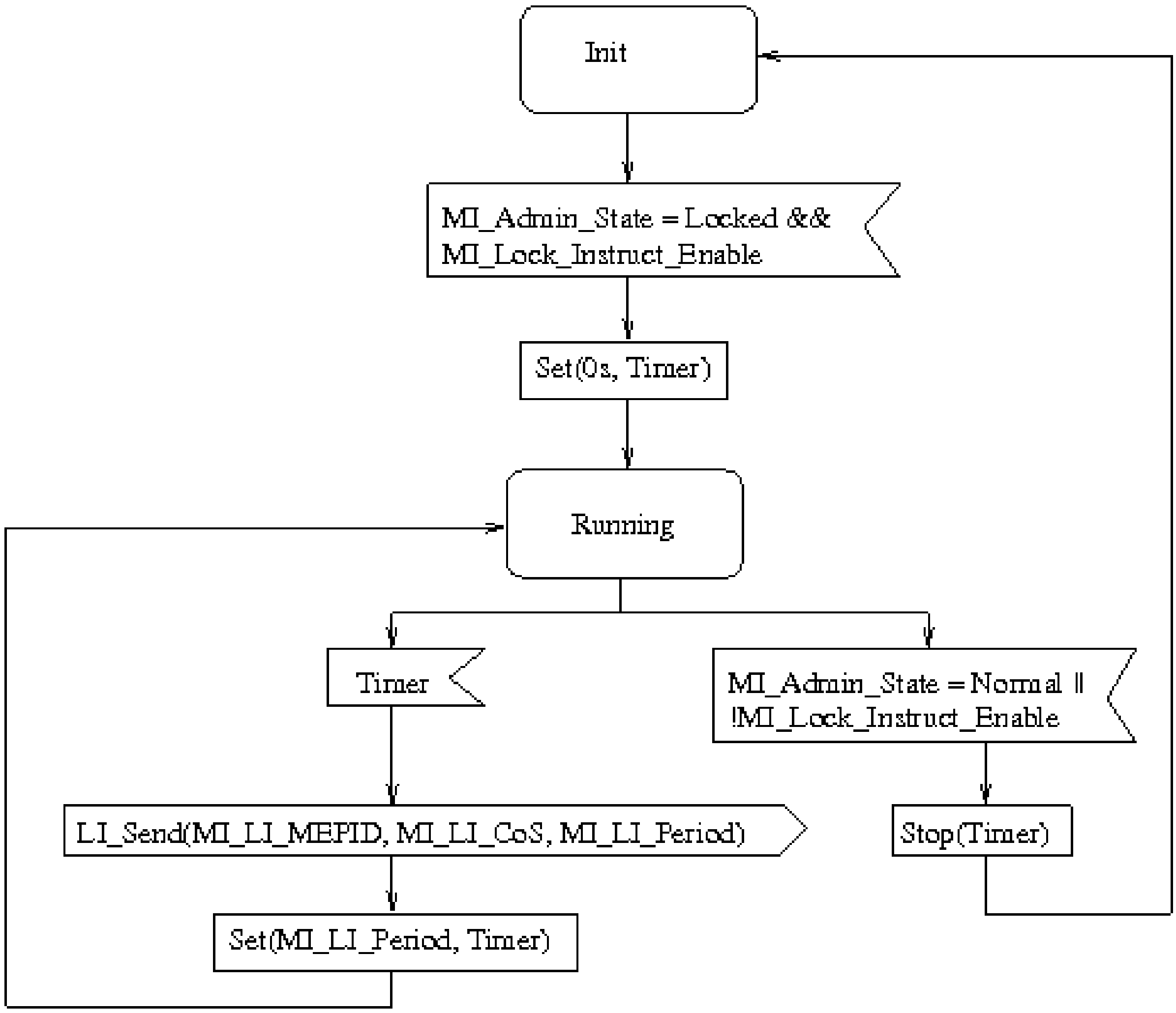


Figure 8-x30/G.8121.2/Y.1381.2 - LI Source Control Process

8.10.5.2 LI Generation Process

The LI Generation Process is described in the following fFigure 8-31.

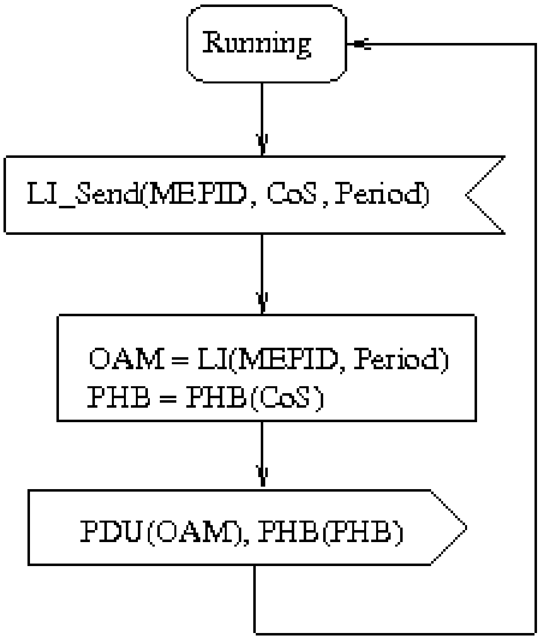


Figure 8-x31/G.8121.2/Y.1381.2 - LI Generation Process

The LI(MEPID, Period) function formats an LI PDU according to [RFC6435], as follows:

- The version is set to 1.
- The reserved field is set to 0.
- The refresh timer field is set to the Period.
- The MEPID is copied into the MEP Source ID TLV.

The PHB(CoS) function returns the PHB with the lowest drop precedence within the Class of Service defined by the CoS input parametercd0cd02

8.10.5.3 LI Reception Process

The LI Reception Process is described in the following fFigure 8-32.

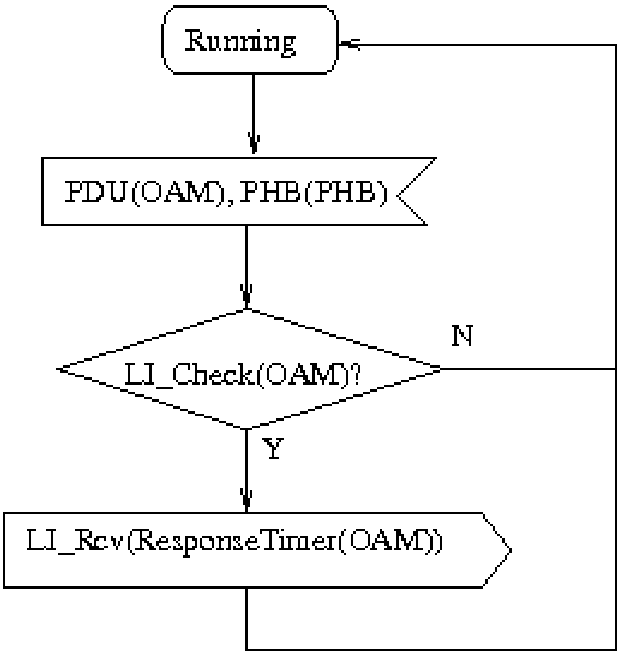


Figure 8-8-32x/G.8121.2/Y.1381.2 - LI Reception Process

The LI_Check(OAM) function performs implementation-specific checks, including those described in [RFC6435], and returns true if the OAM is valid and false otherwise.cd02

8.10.5.4 LI Sink Control Process

The LI Sink Control process is described in Figure 8-x33.

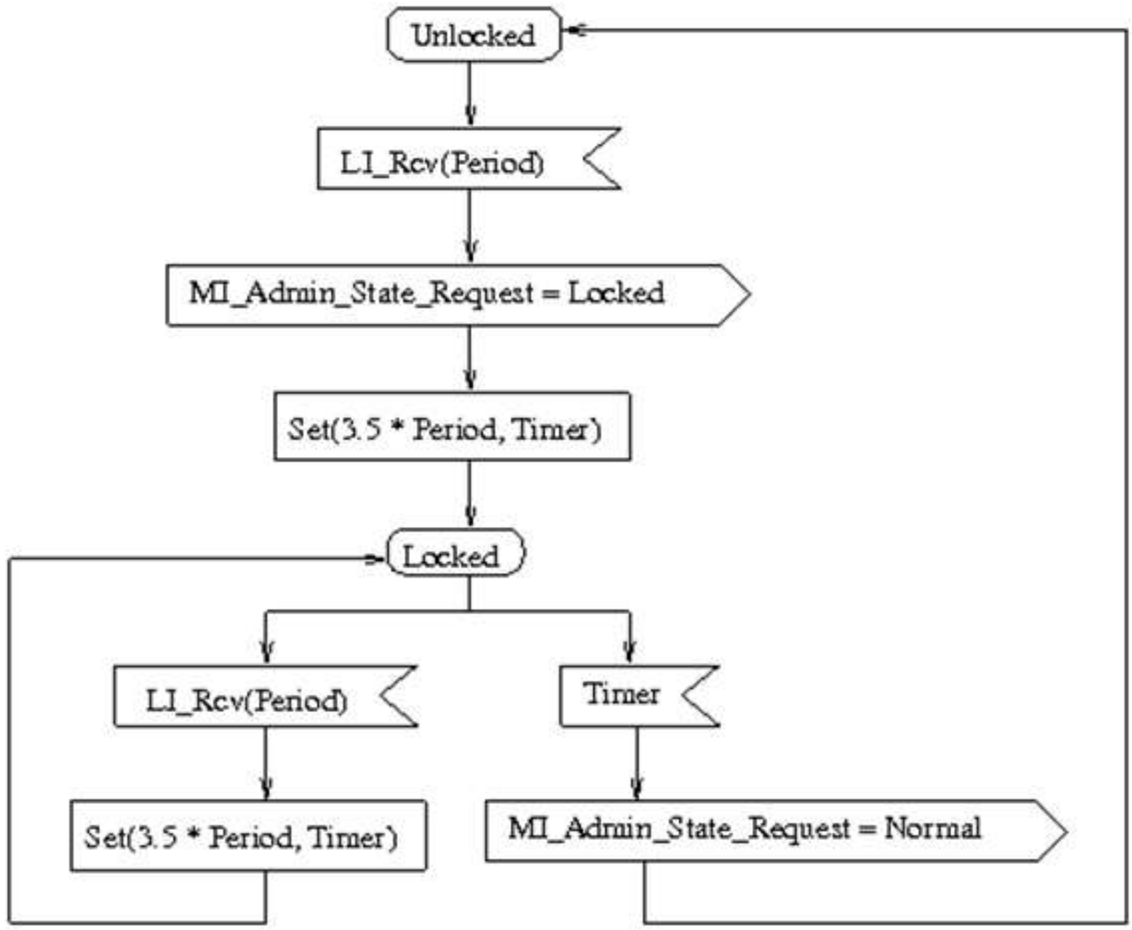


Figure 8-x33/G.8121.2/Y.1381.2 – LI Sink Control Process

8.9 Dataplane Loopback Processes

See clause 8.9 in [ITU-T G.8121]

9 MPLS-TP layer functions

9.1 Connection Functions (MT_C)

Connection Functions are described in in [ITU-T G.8121].

9.2 Termination Functions

9.2.1 MPLS-TP Trail Termination function (MT_TT)

The bidirectional MPLS-TP Trail Termination (MT_TT) function terminates the MPLS-TP OAM to determine the status of the MPLS-TP (sub)layer trail. The MT_TT function is performed by a co-located pair of the MPLS-TP trail termination source (MT_TT_So) and sink (MT_TT_Sk) functions as shown in the figure belowFigure 9-1.

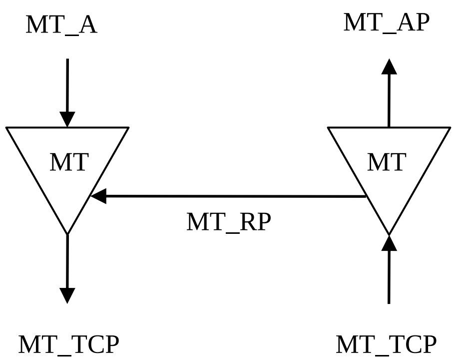


Figure 9-~~x~~1/G.8121.2/Y.1381.2 MT_TT

9.2.1.1 MPLS-TP Trail Termination Source function (MT_TT_So)

The MT_TT_So function determines and inserts the TTL value in the shim header TTL field and adds MPLS-TP OAM to the MT_AI signal at its MT_AP.

The information flow and processing of the MT_TT_So function is defined with reference to in Figure 9-2~~the figure below~~.

• **Symbol:**

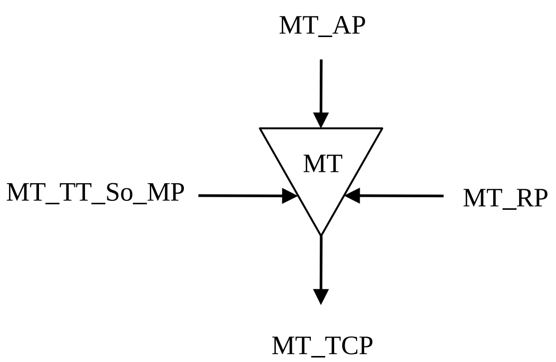


Figure 9-~~x~~2G.8121.2/Y.1381.2 MT_TT_So symbol

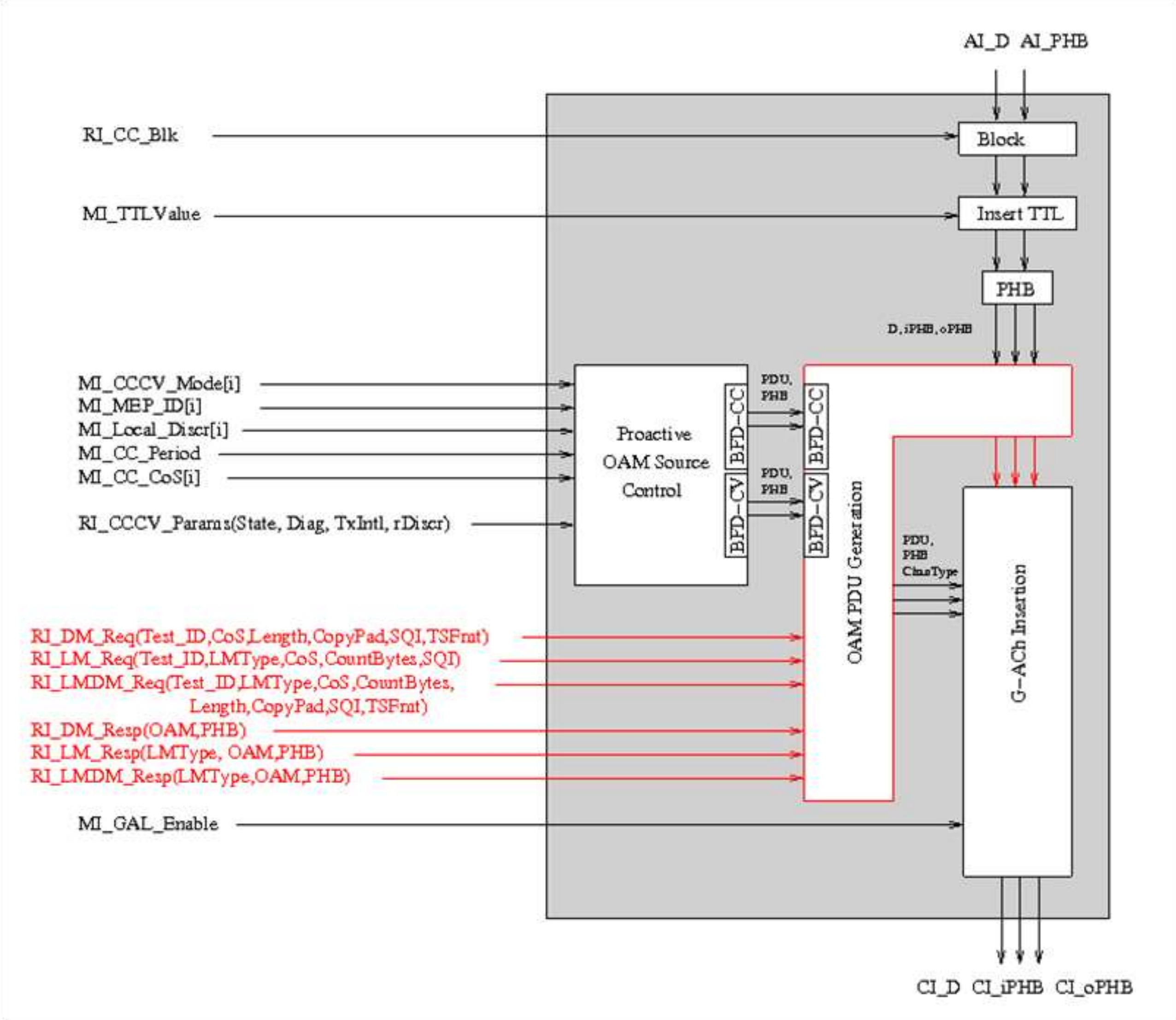
• **Interfaces:**

Table 9-x1/G.8121.2/Y.1381.2 – MT_TT_So inputs and outputs

Input(s)	Output(s)
MT_AP: MT_AI_D MT_AI_PHB **MT_RP:** MT_RI_CCCV_Params MT_RI_CC_Blk ~~MT_RI_DM_Req~~ ~~MT_RI_LM_Req~~ ~~MT_RI_LMDM_Req~~ MT_RI_DM_Resp MT_RI_LM_Resp MT_RI_LMDM_Resp MT_RI_PM_Error MT_RI_DM_Info MT_RI_LM_Info MT_RI_LMDM_Info **MT_TT_So_MP:** MTDe_TT_So_MI_ GAL_Enable MT_TT_So_MI_TTLValue MT_TT_So_MI_CCCV_Mode[] MT_TT_So_MI_MEPID[] MT_TT_So_MI_Local_Discr[] MT_TT_So_MI_CC_Period MT_TT_So_MI_CC_CoS[] MT_TT_So_MI_DMp_Enable[1...MDMp] MT_TT_So_MI_DMp_Test_ID[1...MDMp] MT_TT_So_MI_DMp_CoS[1...MDMp] MT_TT_So_MI_DMp_Length[1...MDMp] MT_TT_So_MI_DMp_CopyPad[1...MDMp] MT_TT_So_MI_DMp_Period[1...MDMp] MT_TT_So_MI_LMp_Enable[1...MLMp] MT_TT_So_MI_LMp_Test_ID[1...MLMp] MT_TT_So_MI_LMp_LMType[1...MLMp] MT_TT_So_MI_LMp_CoS[1...MLMp] MT_TT_So_MI_LMp_CountBytes[1...MLMp] MT_TT_So_MI_LMp_Period[1...MLMp]	**MT_CP:** MT_CI_D MT_CI_oPHB MT_CI_iPHB **MT_TT_So_MP:** MT_TT_So_MI_DMp_PeriodChanged[1...MDMp] MT_TT_So_MI_LMp_PeriodChanged[1...MLMp]

• Processes:

The processes associated with the MT_TT_So function are as depicted in Figure 9-~~X~~3. The sub-processes within each process, described in clause 8.8, are not shown separately


AI_D AI_PHB
RI_CC_Blk
Block
MI_TTLValue
Insert TTL
PHB
D,iPHB,oPHB
MI_CCCV_Mode[i]
MI_MEP_ID[i]
MI_Local_Discr[i]
MI_CC_Period
MI_CC_CoS[i]
RI_CCCV_Params(State, Diag, TxIntl, rDiscr)
Proactive OAM Source Control
BFD-CC
BFD-CV
PDU, PHB
OAM PDU Generation
PDU, PHB ClassType
G-ACh Insertion
RI_DM_Req(Test_ID,CoS,Length,CopyPad,SQI,TSFmt)
RI_LM_Req(Test_ID,LMType,CoS,CountBytes,SQI)
RI_LMDM_Req(Test_ID,LMType,CoS,CountBytes, Length,CopyPad,SQI,TSFmt)
RI_DM_Resp(OAM,PHB)
RI_LM_Resp(LMType, OAM,PHB)
RI_LMDM_Resp(LMType,OAM,PHB)
MI_GAL_Enable
CI_D CI_iPHB CI_oPHB

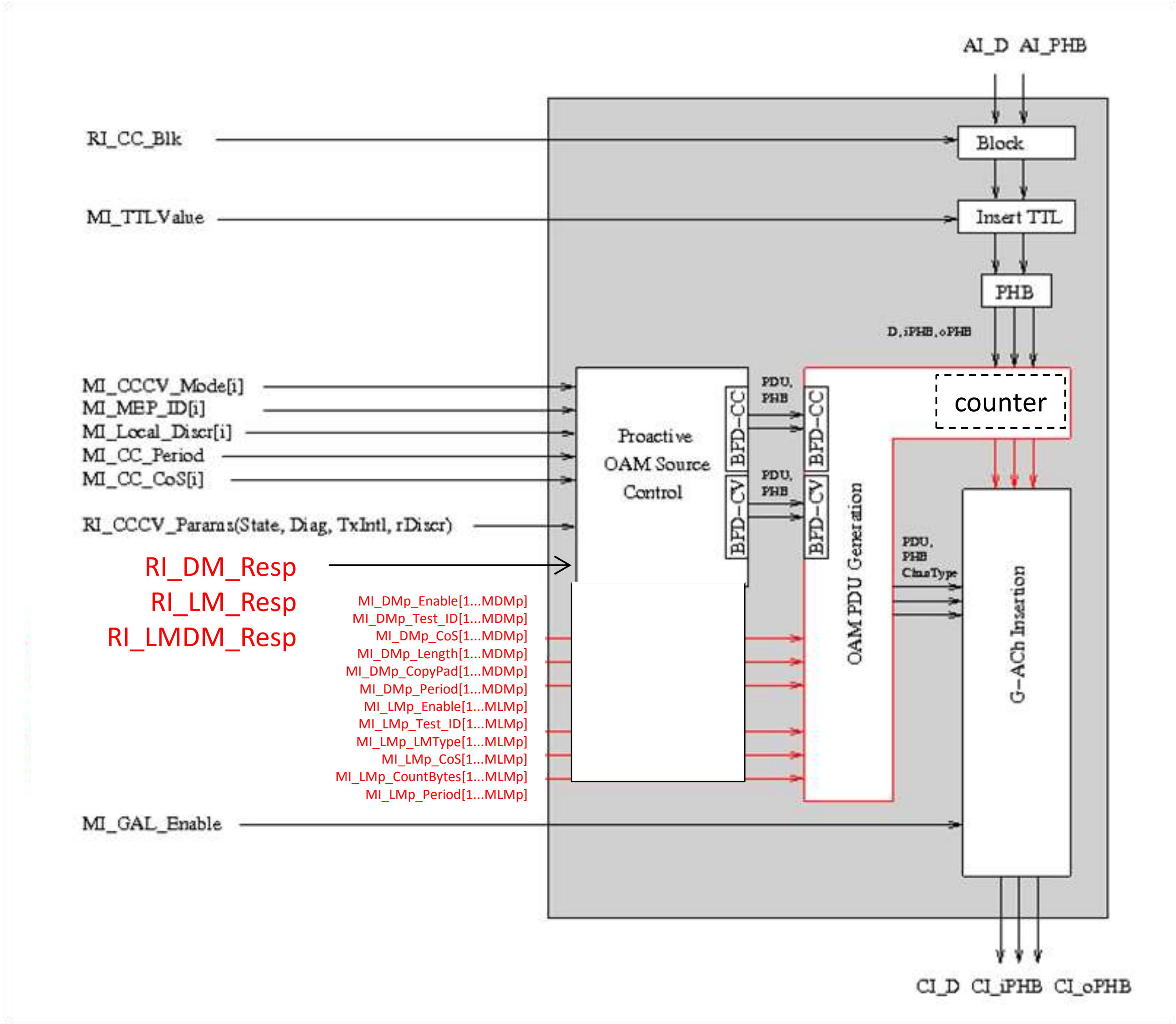


Figure 9-~~x~~3/G.8121.2/Y.1381.2 MT_TT_So process

~~*[Note: add "PM Mux" and "PM Generation" to the list of sub-processes contained in the OAM PDU Generation process.]*~~

PHB: See 9.2.1.1 in [ITU-T G.8121].The AI_PHB signal is assigned to both the CI_iPHB and CI_oPHB signals at the MT_TCP reference point.

Insert TTL: See 9.2.1.1 in [ITU-T G.8121]..

Block: See 9.2.1.1 in [ITU-T G.8121].

G-ACh Insertion: See 8.1.2 in [ITU-T G.8121].

OAM PDU Generation: This contains following sub-processes as described in clause 8.8~~ of [ITU-T G.8121]~~: PM Generation; OAM Mux; PM Mux.~~the OAM Mux sub-process; See 8.8~~

Proactive OAM Source Control Process: This contains following sub-processes as described in clause 8.8~~ of [ITU-T G.8121]~~: CCCV Generation ~~the CCCV Generation sub-process. See 8.8~~

~~[Note: Consistency with G.8121 should be verified]~~

The location of the counter part of the OAM PDU Generation process is shown for illustration only. The exact set of packets to be counted is implementation specific, as described in [RFC6374].

• **Defects:**

None.

• **Consequent actions:**

None.

• **Defect correlations:**

None.

• **Performance monitoring:**

None.

9.2.1.2 MPLS-TP Trail Termination Sink function (MT_TT_Sk)

The MT_TT_Sk function reports the state of the MPLS-TP Trail (Network Connection). It extracts MPLS-TP trail OAM from the MPLS-TP signal at its MT_TCP, detects defects, counts during 1-second periods errors and defects to feed Performance Monitoring when connected and forwards the defect information as backward indications to the companion MT_TT_So function.

Note – The MT_TT_Sk function extracts and processes one level of MPLS-TP OAM irrespective of the presence of more levels.

The information flow and processing of the MT_TT_Sk function is defined with reference to the Figure 9-4below.

• **Symbol:**

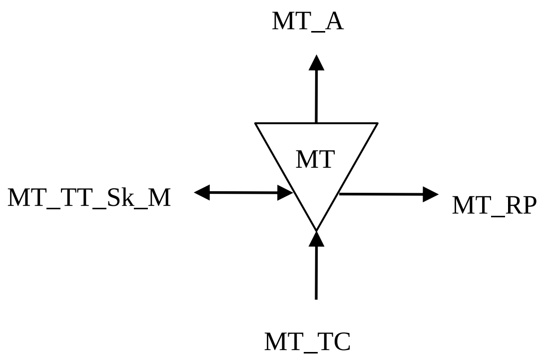


Figure 9-x4/G.8121.2/Y.1381.2 MT_TT_Sk function symbol

• **Interfaces:**

Table 9-x2/G.8121.2/Y.1381.2 – MT_TT_Sk inputs and outputs

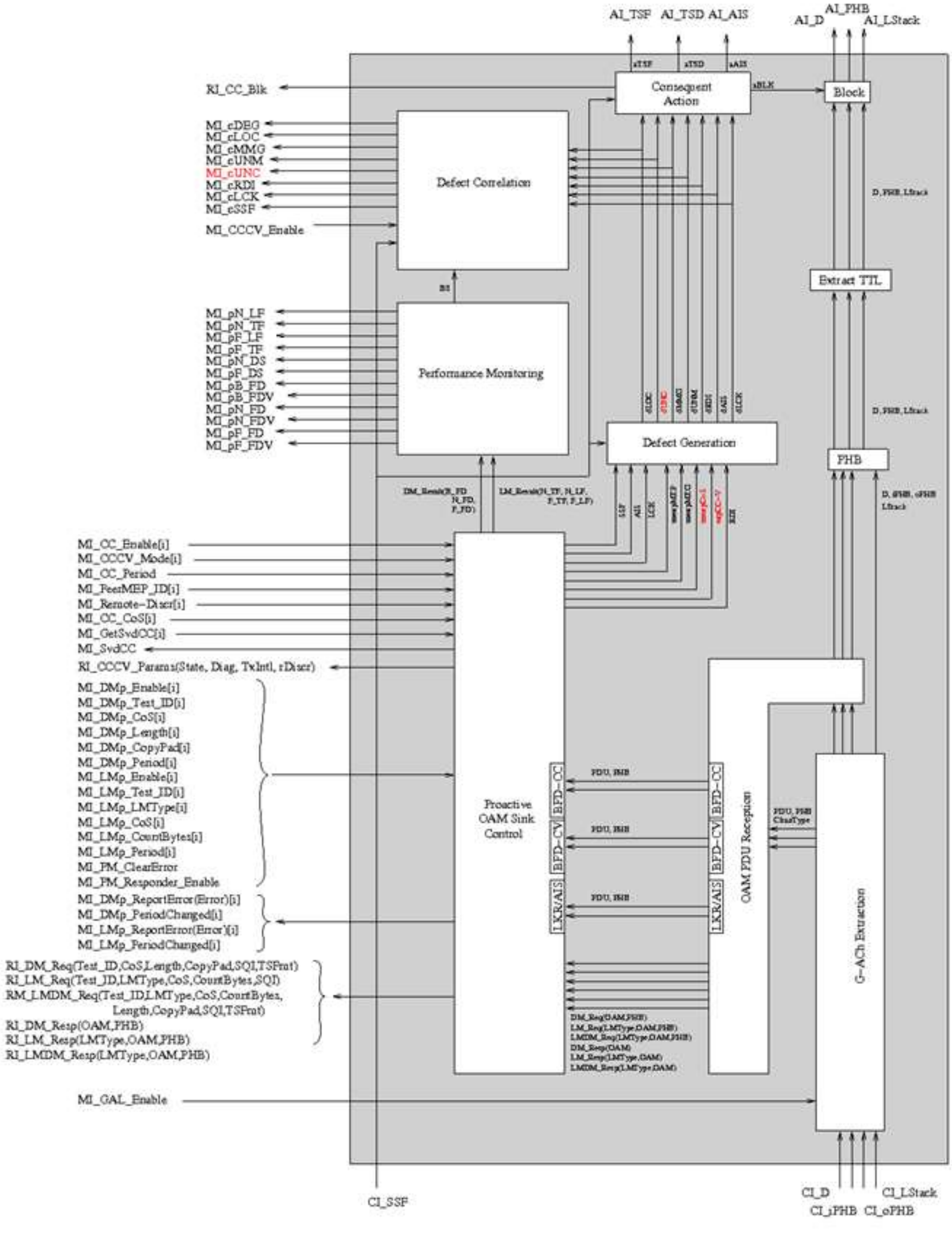
Input(s)	Output(s)
MT_TCP: MT_CI_D	**MT_AP:** MT_AI_D

Input(s)	Output(s)
MT_CI_iPHB MT_CI_oPHB MT_CI_SSF MT_CI_LStack **MT_TT_Sk_MP:** MT_TT_Sk_MI_GAL_Enable MT_TT_Sk_MI_CC_Enable[] MT_TT_Sk_MI_CCCV_Mode[] MT_TT_Sk_MI_CC_Period MT_TT_Sk_MI_Peer–MEPID[] MT_TT_Sk_MI_Remote_Discr[] MT_TT_Sk_MI_CC_CoS[] MT_TT_Sk_MI_GetSvdCC[] MT_TT_Sk_MI_SSF_Reported MT_TT_Sk_MI_RDI_Reported MT_TT_Sk_MI_DMp_Enable[1...MDMp] MT_TT_Sk_MI_LMp_Enable[1...MLMp] ~~MT_TT_Sk_MI_DMp_Enable[1...MDMp]~~ ~~MT_TT_Sk_MI_DMp_Test_ID[1...MDMp]~~ ~~MT_TT_Sk_MI_DMp_CoS[1...MDMp]~~ ~~MT_TT_Sk_MI_DMp_Length[1...MDMp]~~ ~~MT_TT_Sk_MI_DMp_CopyPad[1...MDMp]~~ ~~MT_TT_Sk_MI_DMp_Period[1...MDMp]~~ ~~MT_TT_Sk_MI_LMp_Enable[1...MLMp]~~ ~~MT_TT_Sk_MI_LMp_Test_ID[1...MLMp]~~ ~~MT_TT_Sk_MI_LMp_LMType[1...MLMp]~~ ~~MT_TT_Sk_MI_LMp_CoS[1...MLMp]~~ ~~MT_TT_Sk_MI_LMp_CountBytes[1...MLMp]~~ ~~MT_TT_Sk_MI_LMp_Period[1...MLMp]~~ MT_TT_Sk_MI_PM_ClearError MT_TT_Sk_MI_PM_Responder_Enable	MT_AI_PHB MT_AI_TSF MT_AI_TSD MT_AI_AIS MT_AI_LStack **MT_RP:** MT_RI_CCCV_Params MI_RI_CC_Blk ~~MT_RI_DM_Req~~ ~~MT_RI_LM_Req~~ ~~MT_RI_LMDM_Req~~ MT_RI_DM_Resp MT_RI_LM_Resp MT_RI_LMDM_Resp MT_RI_PM_Error MT_RI_DM_Info MT_RI_LM_Info MT_RI_LMDM_Info **MT_TT_Sk_MP:** MT_TT_Sk_MI_SvdCC MT_TT_Sk_MI_cSSF MT_TT_Sk_MI_cLCK MT_TT_Sk_MI_cLOC[] MT_TT_Sk_MI_cMMG MT_TT_Sk_MI_cUNM MT_TT_Sk_MI_cUNC MT_TT_Sk_MI_cRDI MT_TT_Sk_MI_DMp_ReportError(Error)[1...MDMp] ~~MT_TT_Sk_MI_DMp_PeriodChanged[1...MDMp]~~ MT_TT_Sk_MI_LMp_ReportError(Error)[1...MLMp] ~~MT_TT_Sk_MI_LMp_PeriodChanged[1...MLMp]~~ MT_TT_Sk_MI_pN_LF[1...P] MT_TT_Sk_MI_pN_TF[1...P] MT_TT_Sk_MI_pF_LF[1...P] MT_TT_Sk_MI_pF_TF[1...P] MT_TT_Sk_MI_pF_DS MT_TT_Sk_MI_pN_DS MT_TT_Sk_MI_pB_FD[1...P]

Input(s)	Output(s)
	MT_TT_Sk_MI_pB_FDV[1...P] MT_TT_Sk_MI_pN_FD[1...P] MT_TT_Sk_MI_pN_FDV[1...P] MT_TT_Sk_MI_pF_FD[1...P] MT_TT_Sk_MI_pF_FDV[1...P] MT_TT_Sk_MI_cDEG

• Processes:

The processes associated with the MT_TT_Sk function are as depicted in the figure Figure 9-5 below.


AI_TSF
AI_TSD
AI_AIS
AI_D
AI_PHB
AI_LStack
Consequent Action
Block
RI_CC_Blk
MI_cDEG
MI_cLOC
MI_cMMG
MI_cUNM
MI_cUNC
MI_cRDI
MI_cLCK
MI_cSSF
MI_CCCV_Enable
Defect Correlation
Extract TTL
MI_pN_LF
MI_pN_TF
MI_pF_LF
MI_pF_TF
MI_pN_DS
MI_pF_DS
MI_pB_FD
MI_pB_FDV
MI_pN_FD
MI_pN_FDV
MI_pF_FD
MI_pF_FDV
Performance Monitoring
Defect Generation
PHB
MI_CC_Enable[i]
MI_CCCV_Mode[i]
MI_CC_Period
MI_PeerMEP_ID[i]
MI_Remote-Discr[i]
MI_CC_CoS[i]
MI_GetSvdCC[i]
MI_SvdCC
RI_CCCV_Params(State, Diag, TxIntl, rDiscr)
MI_DMp_Enable[i]
MI_DMp_Test_ID[i]
MI_DMp_CoS[i]
MI_DMp_Length[i]
MI_DMp_CopyPad[i]
MI_DMp_Period[i]
MI_LMp_Enable[i]
MI_LMp_Test_ID[i]
MI_LMp_LMType[i]
MI_LMp_CoS[i]
MI_LMp_CountBytes[i]
MI_LMp_Period[i]
MI_PM_ClearError
MI_PM_Responder_Enable
MI_DMp_ReportError(Error)[i]
MI_DMp_PeriodChanged[i]
MI_LMp_ReportError(Error)[i]
MI_LMp_PeriodChanged[i]
RI_DM_Req(Test_ID,CoS,Length,CopyPad,SQI,TSFmt)
RI_LM_Req(Test_ID,LMType,CoS,CountBytes,SQI)
RM_LMDM_Req(Test_ID,LMType,CoS,CountBytes, Length,CopyPad,SQI,TSFmt)
RI_DM_Resp(OAM,PHB)
RI_LM_Resp(LMType,OAM,PHB)
RI_LMDM_Resp(LMType,OAM,PHB)
Proactive OAM Sink Control
BFD-CC
BFD-CV
LKR/AIS
OAM PDU Reception
G-ACh Extraction
MI_GAL_Enable
CI_SSF
CI_D
CI_iPHB
CI_oPHB
CI_LStack

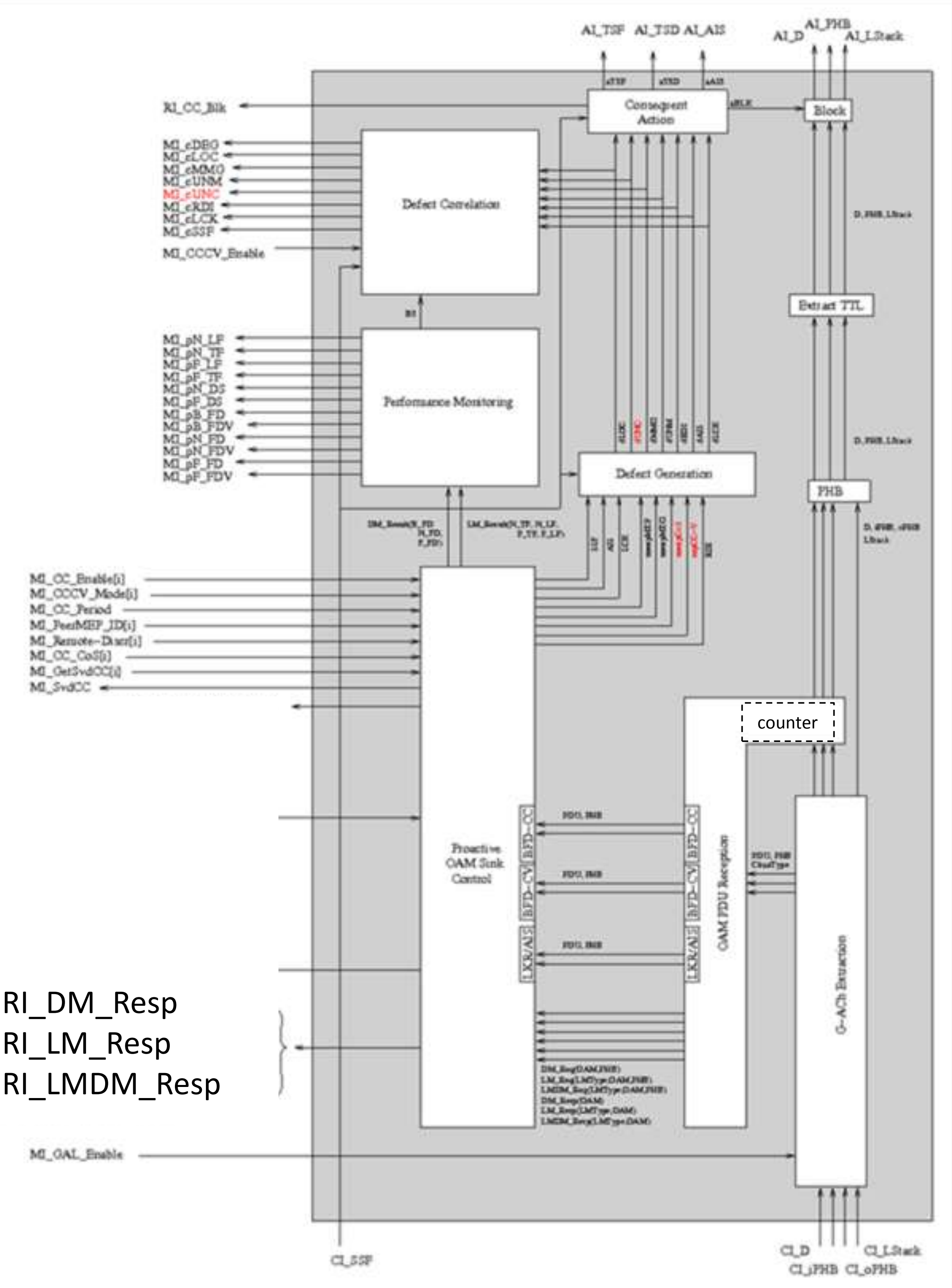


Figure 9-x-5/G.8121.2/G.8121.2/Y.1381.2 – MT_TT_Sk Process

[Annex VI, C.2025]

[Performance Monitoring process are FFS and are not depicted]

PHB: See 9.2.1.2 in [ITU-T G.8121].

Extract TTL: See 9.2.1.2 in [ITU-T G.8121].

Block: See 9.2.1.2 in [ITU-T G.8121].

G-ACh Extraction: see 8.1.3 in [ITU-T G.8121].

OAM PDU Reception: This contains following sub-processes as described in clause 8.8 of [ITU-T G.8121]: PM Reception; OAM Demux; Session Demux; PM Demux

the OAM Demux and Session Demux sub-processes. See 8.8

Proactive OAM Sink Control: This contains following sub-processes as described in clause 8.8 of [ITU-T G.8121]: CCCV Reception; LCK/AIS Reception; Proactive PM Sink Control; PM Responder.

the CCCV Reception and LCK/AIS Reception sub-processes. See 8.8

[Note: Consistency with G.8121 should be verified]

Performance Counter Process: See 9.2.1.2 in [ITU-T G.8121].

Defect Generation: See 9.2.1.2 in [ITU-T G.8121].

The location of the counter part of the OAM PDU Reception process is shown for illustration only. The exact set of packets to be counted is implementation specific, as described in [RFC6374].

• Defects:

See [ITU-T G.8121]

• Consequent actions:

See [ITU-T G.8121]

• Defect correlations:

See [ITU-T G.8121]

• Performance monitoring:

See [ITU-T G.8121]

9.3 Adaptation Functions

9.3.1 MPLS-TP to MPLS-TP Adaptation function (MT/MT_A)

This atomic functions are defined in clause 9.3.1 in [ITU-T G.8121]. They use the OAM protocol specific AIS insertion process and LCK generation process as defined in clause 8.6.2. For the MT/MT_A_Sk function, in addition to the MI shown in Table 9.5 in [ITU-T G.8121]. and Figure 9.11 in [ITU-T G.8121], there is an additional protocol-specific MI used by the AIS Insert Process defined in this document: MI_Local_Defect[1..M].

9.4 Diagnostic Functions

This clause describes Termination Functions and Adaptation Functions relating to OAM.

9.4.1 Diagnostic Functions for MEPs

~~Editor Note~~

~~….~~

~~Rest of the note in TD742/3 removed]~~

9.4.1.1 MT Diagnostic Trail Termination Functions for MEPs (MTDe_TT)

The bidirectional MTDe Flow Termination (MTDe_TT) function is performed by a co-located pair of MTDe flow termination source (MTDe_TT_So) and sink (MTDe_TT_Sk) functions as shown in ~~the figure below~~Figure 9-6:

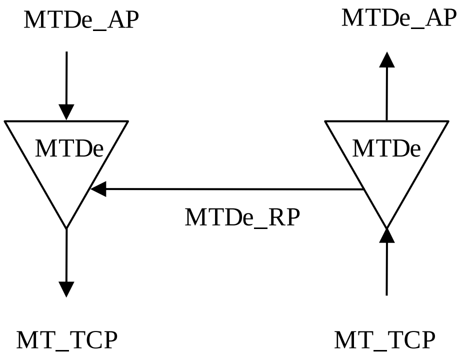


Figure 9-~~xx~~6/G.8121.2/Y.1381.2~~G.8121.2~~ - MTDe_TT

9.4.1.1.1 MT Diagnostic Trail Termination Source Function for MEPs (MTDe_TT_So)

The MTDe_TT_So Process diagram is shown in ~~the figure below~~Figure 9-7.

Symbol

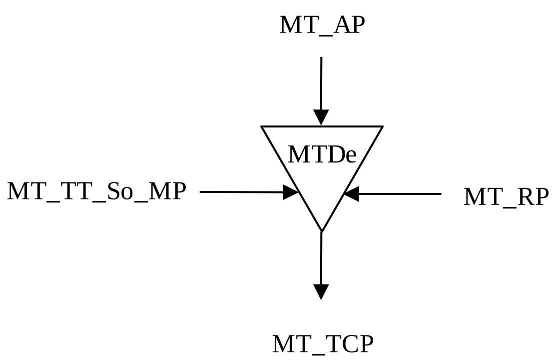


Figure 9-~~xx~~7/G.8121.2/Y.1381.2 - MTDe_TT_So_Symbol

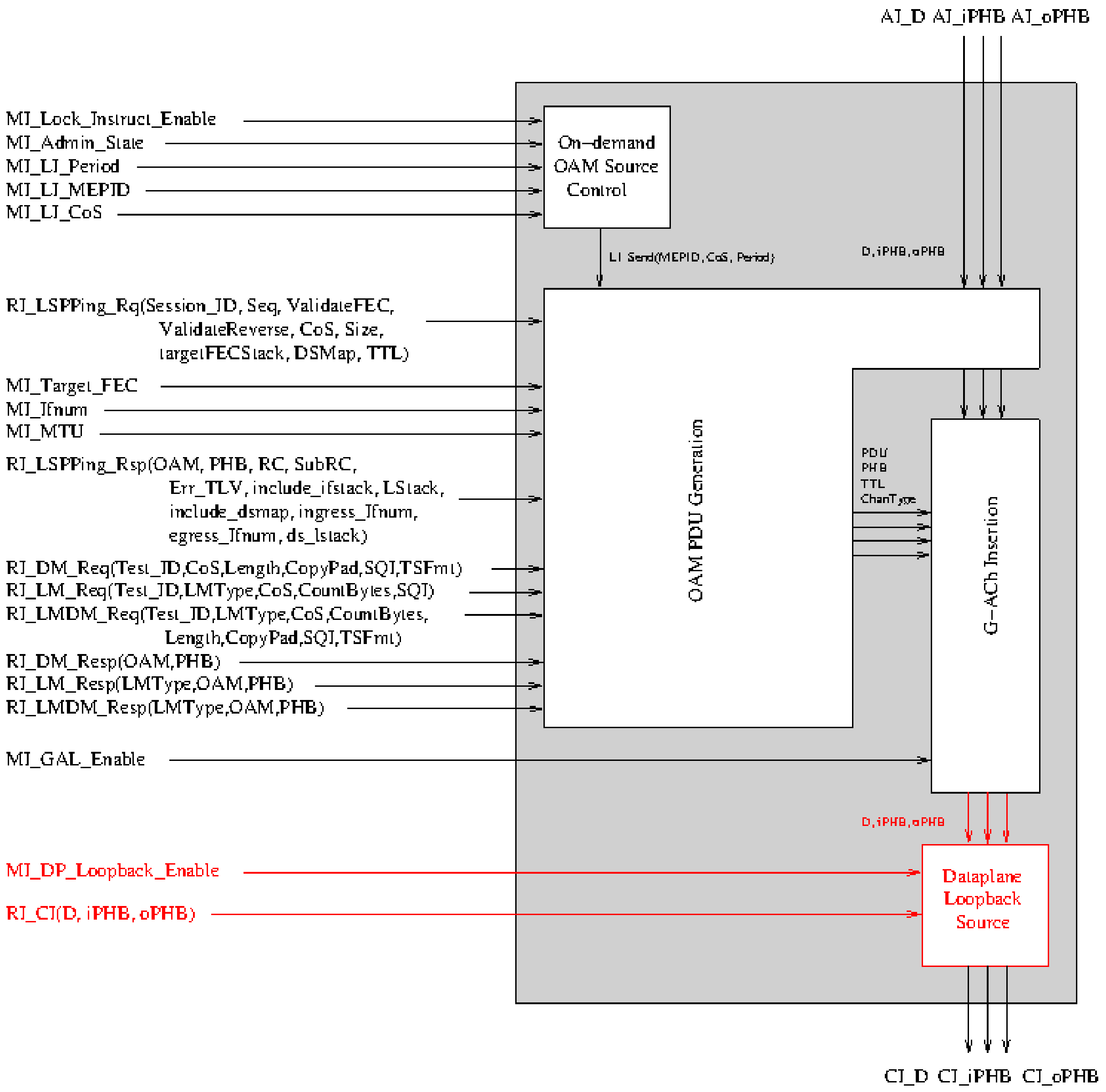
Interfaces

Table 9-*/-3/G.8121.2/Y.1381.2 – MTDe_TT_So interfaces

Input(s)	Output(s)
MT_AP: MTDe_AI_D MTDe_AI_iPHB MTDe_AI_oPHB **MTDe_RP:** ~~MTDe_RI_LSPPing_Rq~~ ~~MTDe_RI_LSPPing_Rsp~~ ~~MTDe_RI_DM_Req~~ ~~MTDe_RI_LM_Req~~ ~~MTDe_RI_LMDM_Req~~ ~~MTDe_RI_DM_Resp~~ ~~MTDe_RI_LM_Resp~~ ~~MTDe_RI_LMDM_Resp~~ ~~MTDe_RI_DM_Result~~ ~~MTDe_RI_LM_Result~~ ~~MTDe_RI_LMDM_Resu~~ MTDe_RI_LSPPing_Rq MTDe_RI_LSPPing_Rsp MTDe_RI_rDM_Reqsp MTDe_RI_rLM_Reqsp MTDe_RI_rLMDM_Reqsp MTDe_RI_DM_Resp MTDe_RI_LM_Resp MTDe_RI_LMDM_Resp MTDe_RI_CI **MTDe_TT_So_MP:** MTDe_TT_So_MI_ GAL_Enable MTDe_TT_So_MI_CV_Series MTDe_TT_So_MI_ODCV_Trace MTDe_TT_So_MI_FEC_Checking MTDe_TT_So_MI_Target_FEC MTDe_TT_So_MI_Ifnum MTDe_TT_So_MI_MTU MTDe_TT_So_MI_DMo_Start(CoS, Test_ID, Length, Period, CopyPad)[1...M$_{DMo}$] MTDe_TT_So_MI_LMo_Start(CoS, Test_ID, Period, LMType, CountBytes)[1...M$_{LMo}$] MTDe_TT_So_MI_LMDMo_Start(CoS, Test_ID, Length, Period, LMType, CountBytes, CopyPad)[1...M$_{LMDMo}$] MTDe_TT_So_MI_DMo_Terminate [1...M$_{DMo}$] MTDe_TT_So_MI_LMo_Terminate[1...M$_{LMo}$] MTDe_TT_So_MI_LMDMo_Terminate [1...M$_{LMDMo}$]	**MT_CP:** MT_CI_D MT_CI_oPHB MT_CI_iPHB **MTDe_RP:** MTDe_RI_rLSPPing_Rsp **MTDe_TT_So_MP:** MTDe_TT_Sok_MI_DMo_ReportError(Error) [1...M$_{DMo}$] MTDe_TT_Sko_MI_DMo_PeriodChanged [1...M$_{DMo}$] MTDe_TT_Sko_MI_LMo_ReportError(Error) [1...M$_{LMo}$] MTDe_TT_Sko_MI_LMo_PeriodChanged [1...M$_{LMo}$] MTDe_TT_Sko_MI_DMo_Result(count, B_FD[], F_FD[], N_FD[])[1...M$_{DMo}$] MTDe_TT_Sko_MI_LMo_Result(N_TF, N_LF, F_TF, F_LF)[1...M$_{LMo}$] ~~9.5~~ MTDe_TT_So_MI_CV_Series_Result MTDe_TT_So_MI_ODCV_Trace_Result MTDe_TT_So_MI_ODCV_FWErr MTDe_TT_So_MI_ODCV_BWErr

MTDe_TT_So_MI_Lock_Instruct_Enable MTDe_TT_So_MI_Admin_State MTDe_TT_So_MI_LI_Period MTDe_TT_So_MI_LI_MEPID MTDe_TT_So_MI_LI_CoS MTDe_TT_So_MI_DP_Loopback_Enable	

Processes


AI_D AI_iPHB AI_oPHB
MI_Lock_Instruct_Enable
MI_Admin_State
MI_LI_Period
MI_LI_MEPID
MI_LI_CoS
On–demand OAM Source Control
LI Send(MEPID, CoS, Period)
D, iPHB, oPHB
RI_LSPPing_Rq(Session_ID, Seq, ValidateFEC, ValidateReverse, CoS, Size, targetFECStack, DSMap, TTL)
MI_Target_FEC
MI_Ifnum
MI_MTU
RI_LSPPing_Rsp(OAM, PHB, RC, SubRC, Err_TLV, include_ifstack, LStack, include_dsmap, ingress_Ifnum, egress_Ifnum, ds_lstack)
RI_DM_Req(Test_ID,CoS,Length,CopyPad,SQI,TSFmt)
RI_LM_Req(Test_ID,LMType,CoS,CountBytes,SQI)
RI_LMDM_Req(Test_ID,LMType,CoS,CountBytes, Length,CopyPad,SQI,TSFmt)
RI_DM_Resp(OAM,PHB)
RI_LM_Resp(LMType,OAM,PHB)
RI_LMDM_Resp(LMType,OAM,PHB)
OAM PDU Generation
PDU
PHB
TTL
ChanType
G–ACh Insertion
MI_GAL_Enable
D, iPHB, oPHB
MI_DP_Loopback_Enable
RI_CI(D, iPHB, oPHB)
Dataplane Loopback Source
CI_D CI_iPHB CI_oPHB

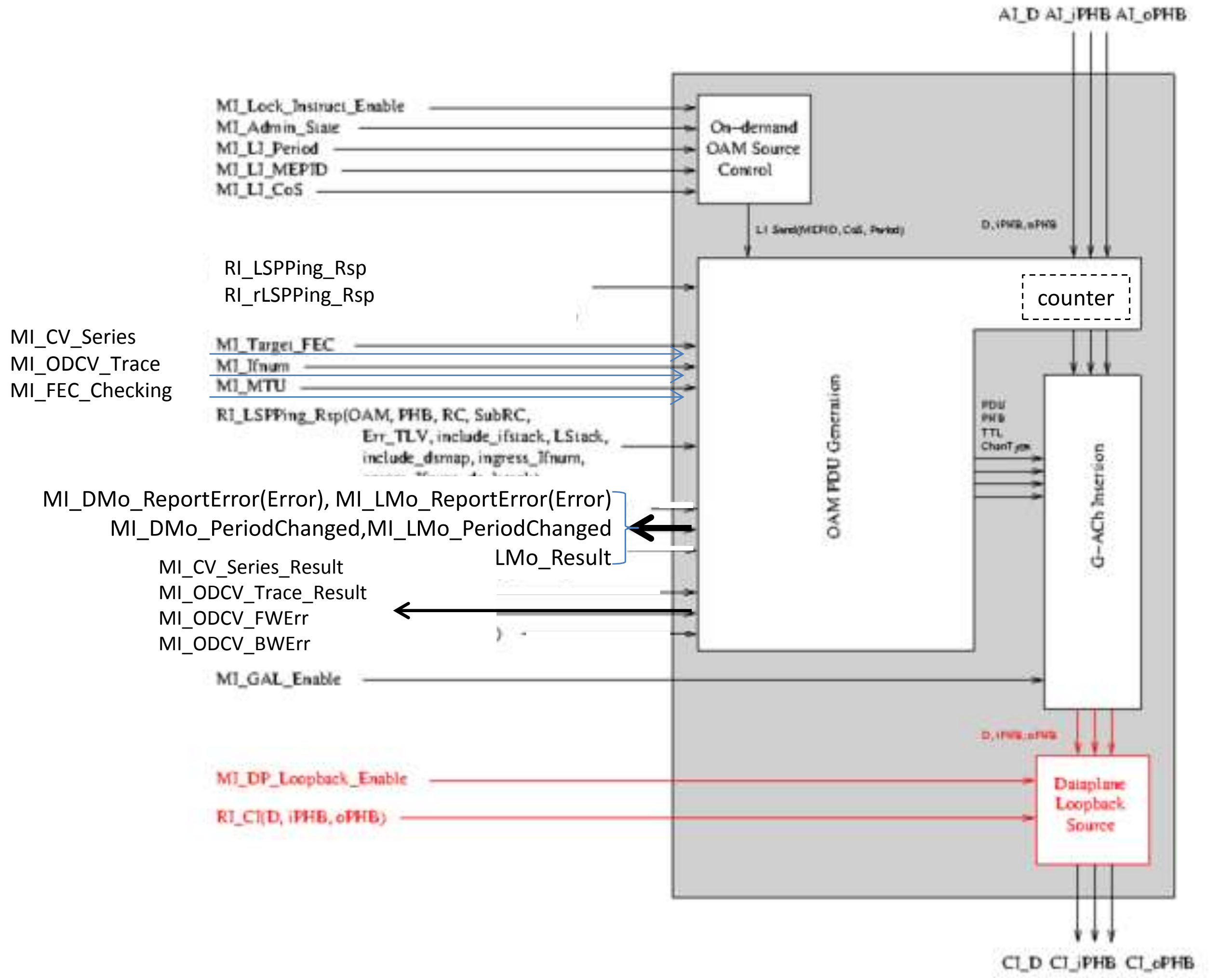


Figure 9-~~x~~8/G.8121.2/Y.1381.2 – MTDe_TT_So Process

~~[Ed note: Updated per cd02, Need to align with Table 9-**/G.8121.2]~~

PHB: See 9.4.1.1.1/G.8121.

G-ACh Insertion: See 8.1.2/G.8121

OAM PDU Generation: This contains following sub-processes as described in clause 8.8~~ of [ITU-T G.8121]~~: On-demand CV Request Generation; On-demand CV Response Generation; LI Generation; PM Generation; OAM Mux; PM Mux ~~the On-demand CV Request Generation, On-demand CV Response Generation, PM Generation, OAM Mux, PM Mux sub-processes and LI Generation sub-processes. See 8.8.~~

On-demand OAM Source Control Process: his contains the following sub-processes as described in clause 8.8~~ of [ITU-T G.8121]~~: LI Source Control; On-Demand PM Control; On-Demand CV Control~~See 8.8.~~

The location of the counter part of the OAM PDU Generation process is shown for illustration only. The exact set of packets to be counted is implementation specific, as described in [RFC 6374].

~~[Note: Consistency with G.8121 should be verified]~~

Dataplane Loopback Source process: see clause 8.9.1 in [ITU-T G.8121]

• **Defects:**

None.

• **Consequent actions:**

None.

• **Defect correlations:**

None.

• **Performance monitoring:**

None.

9.4.1.1.2 MT Diagnostic Trail Termination Sink Function for MEPs (MTDe_TT_Sk)

The MTDe_TT_Sk Process diagram is shown in ~~the figure below~~Figure 9-9.

Symbol

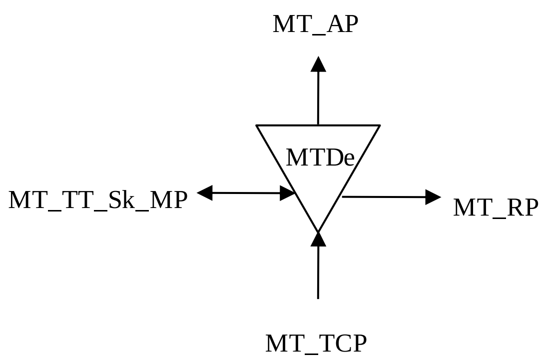


Figure 9-~~xx~~9/G.8121.2/Y.1381.2 - MTDe_TT_Sk_Symbol

[Ed note: Need to align with Fig 9-14/G.8121]

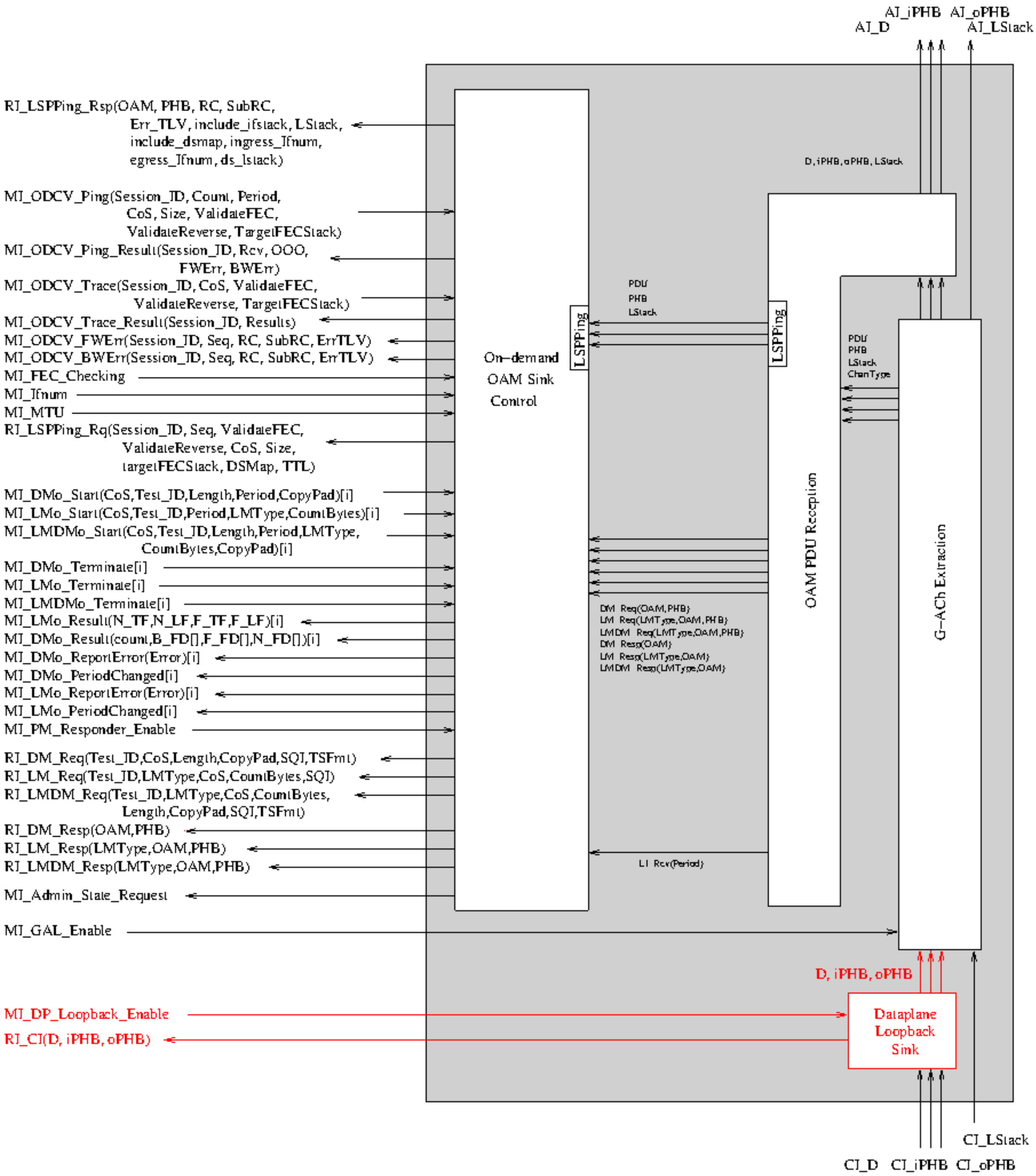
Interfaces

Table 9-4*/G.8121.2/Y.1381.2 – MTDe_TT_Sk interfaces

Input(s)	Output(s)
MT_CP: MT_CI_D MT_CI_iPHB MT_CI_oPHB MT_CI_LStack **MT_RP:** MTDe_RI_rLSPPing_Rsp **MTDe_TT_Sk_MP:** MTDe_TT_Sk_MI_GAL_Enable	**MTDe_AP:** MTDe_AI_D MTDe_AI_iPHB MTDe_AI_oPHB MTDe_AI_LStack **MTDe_RP:** ~~MTDe_RI_LSPPing_Rq~~ ~~MTDe_RI_LSPPing_Rsp~~ ~~MTDe_RI_DM_Req~~ ~~MTDe_RI_LM_Req~~ ~~MTDe_RI_LMDM_Req~~

Input(s)	Output(s)
MTDe_TT_Sk_MI_ODCV_Ping MTDe_TT_Sk_MI_ODCV_Trace MTDe_TT_Sk_MI_FEC_Checking MTDe_TT_Sk_MI_Ifnum MTDe_TT_Sk_MI_MTU MTDe_TT_Sk_MI_DMo_Start(CoS, Test_ID, Length, Period, CopyPad)[1...M_{DMo}] MTDe_TT_Sk_MI_LMo_Start(CoS, Test_ID, Period, LMType, CountBytes)[1...M_{LMo}] MTDe_TT_Sk_MI_LMDMo_Start(CoS, Test_ID, Length, Period, LMType, CountBytes, CopyPad)[1...M_{LMDMo}] MTDe_TT_Sk_MI_DMo_Terminate [1...M_{DMo}] MTDe_TT_Sk_MI_LMo_Terminate[1...M_{LMo}] MTDe_TT_Sk_MI_LMDMo_Terminate [1...M_{LMDMo}] MTDe_TT_Sk_MI_PM_Responder_Enable MTDe_TT_Sk_MI_DP_Loopback_Enable	MTDe_RI_DM_Resp MTDe_RI_LM_Resp MTDe_RI_LMDM_Resp MTDe_RI_DM_Result MTDe_RI_LM_Result MTDe_RI_LMDM_Result MTDe_RI_LSPPing_Rsp MTDe_RI_rDM_Reqsp MTDe_RI_rLM_Respq MTDe_RI_rLMDM_Respq MTDe_RI_DM_Resp MTDe_RI_LM_Resp MTDe_RI_LMDM_Resp MTDe_RI_CI **MTDe_FT_Sk_MP:** MTDe_TT_Sk_MI_Admin_State_Request MTDe_TT_Sk_MI_ODCV_Ping_Result MTDe_TT_Sk_MI_ODCV_Trace_Result MTDe_TT_Sk_MI_ODCV_FWErr MTDe_TT_Sk_MI_ODCV_BWErr MTDe_TT_Sk_MI_DMo_ReportError(Error) [1...M_{DMo}] MTDe_TT_Sk_MI_DMo_PeriodChanged [1...M_{DMo}] MTDe_TT_Sk_MI_LMo_ReportError(Error) [1...M_{LMo}] MTDe_TT_Sk_MI_LMo_PeriodChanged [1...M_{LMo}] MTDe_TT_Sk_MI_DMo_Result(count, B_FD[], F_FD[], N_FD[])[1...M_{DMo}] MTDe_TT_Sk_MI_LMo_Result(N_TF, N_LF, F_TF, F_LF)[1...M_{LMo}]

Processes


AI_iPHB
AI_oPHB
AI_D
AI_LStack
RI_LSPPing_Rsp(OAM, PHB, RC, SubRC, Err_TLV, include_ifstack, LStack, include_dsmap, ingress_Ifnum, egress_Ifnum, ds_lstack)
MI_ODCV_Ping(Session_ID, Count, Period, CoS, Size, ValidateFEC, ValidateReverse, TargetFECStack)
MI_ODCV_Ping_Result(Session_ID, Rcv, OOO, FWErr, BWErr)
MI_ODCV_Trace(Session_ID, CoS, ValidateFEC, ValidateReverse, TargetFECStack)
MI_ODCV_Trace_Result(Session_ID, Results)
MI_ODCV_FWErr(Session_ID, Seq, RC, SubRC, ErrTLV)
MI_ODCV_BWErr(Session_ID, Seq, RC, SubRC, ErrTLV)
MI_FEC_Checking
MI_Ifnum
MI_MTU
RI_LSPPing_Rq(Session_ID, Seq, ValidateFEC, ValidateReverse, CoS, Size, targetFECStack, DSMap, TTL)
MI_DMo_Start(CoS,Test_ID,Length,Period,CopyPad)[i]
MI_LMo_Start(CoS,Test_ID,Period,LMType,CountBytes)[i]
MI_LMDMo_Start(CoS,Test_ID,Length,Period,LMType, CountBytes,CopyPad)[i]
MI_DMo_Terminate[i]
MI_LMo_Terminate[i]
MI_LMDMo_Terminate[i]
MI_LMo_Result(N_TF,N_LF,F_TF,F_LF)[i]
MI_DMo_Result(count,B_FD[],F_FD[],N_FD[])[i]
MI_DMo_ReportError(Error)[i]
MI_DMo_PeriodChanged[i]
MI_LMo_ReportError(Error)[i]
MI_LMo_PeriodChanged[i]
MI_PM_Responder_Enable
RI_DM_Req(Test_ID,CoS,Length,CopyPad,SQI,TSFmt)
RI_LM_Req(Test_ID,LMType,CoS,CountBytes,SQI)
RI_LMDM_Req(Test_ID,LMType,CoS,CountBytes, Length,CopyPad,SQI,TSFmt)
RI_DM_Resp(OAM,PHB)
RI_LM_Resp(LMType,OAM,PHB)
RI_LMDM_Resp(LMType,OAM,PHB)
MI_Admin_State_Request
MI_GAL_Enable
MI_DP_Loopback_Enable
RI_CI(D, iPHB, oPHB)
On-demand
OAM Sink
Control
LSPPing
PDU
PHB
LStack
LSPPing
D, iPHB, oPHB, LStack
OAM PDU Reception
PDU
PHB
LStack
ChanType
G-ACh Extraction
DM Req(OAM,PHB)
LM Req(LMType,OAM,PHB)
LMDM Req(LMType,OAM,PHB)
DM Resp(OAM)
LM Resp(LMType,OAM)
LMDM Resp(LMType,OAM)
LI Rcv(Period)
D, iPHB, oPHB
Dataplane
Loopback
Sink
CI_LStack
CI_D
CI_iPHB
CI_oPHB

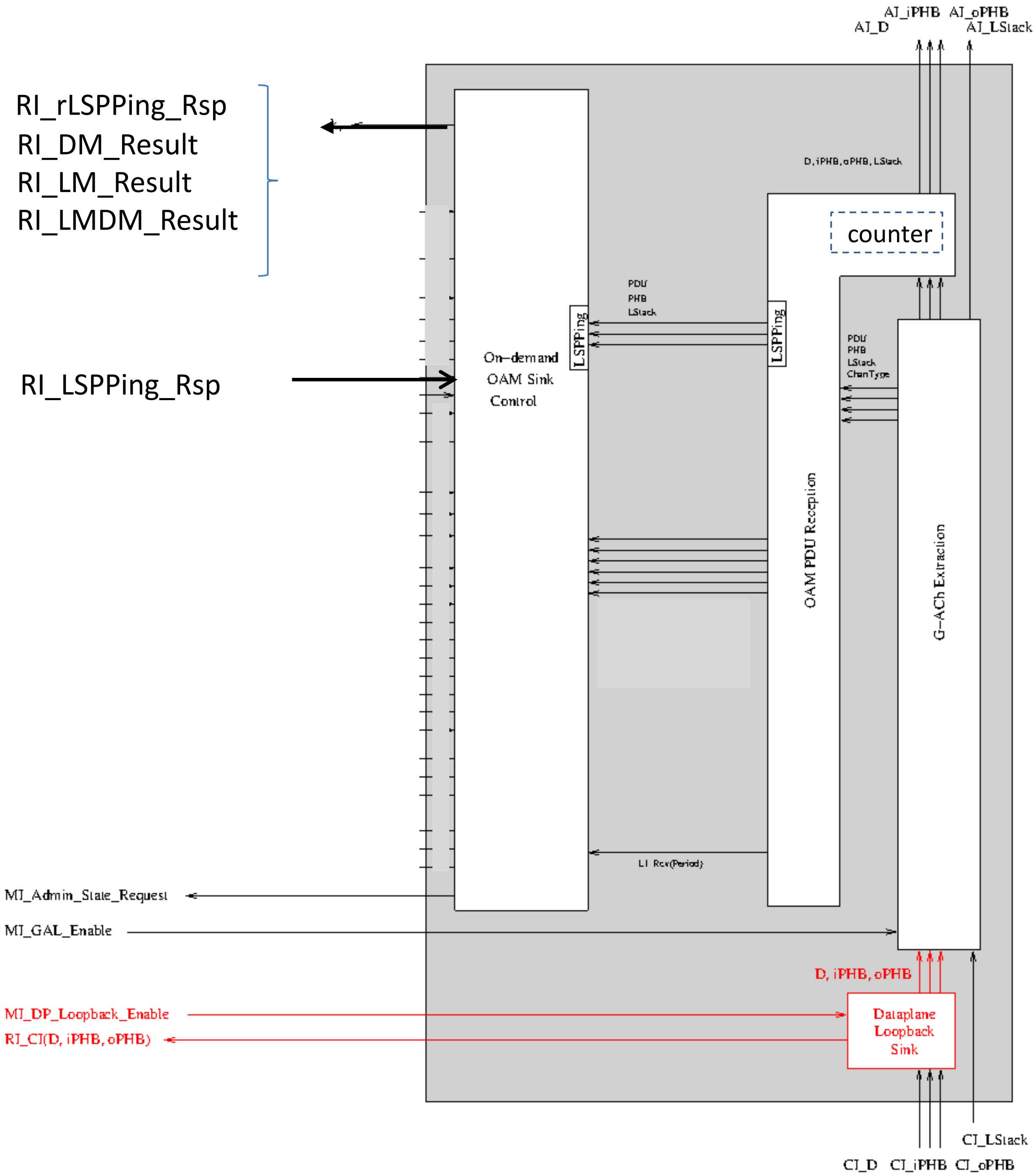


Figure 9-10x/G.8121.2/Y.1381.2 – MTDe_TT_Sk Process
[Need to be aligned with the table above]
[Ed note: Updated per Annex II, cd02, Need to align with Table 9-**/G.8121.2]

PHB: See 9.4.1.1.2 in [ITU-T G.8121].

G-Ach Extraction: see 8.1.3 in [ITU-T G.8121].

OAM PDU Reception: This contains following sub-processes as described in clause 8.8 of [ITU-T G.8121]: On-demand CV Reception; LI Reception; PM Reception; OAM Demux; PM Demux.

the OAM Demux, PM Demux, On-demand CV Reception, PM Reception processes and LI Reception process. See 8.8

On-demand OAM Sink Control: This contains following sub-processes as described in clause 8.8 of [ITU-T G.8121]: On-demand CV Control; MEP On-demand CV Responder; LI Sink Control; On-demand PM Control; PM Responderthe On-demand CV Control, MEP On-demand CV Responder, On-demand PM Control, PM Responder sub-processes, and LI Sink Control sub-processes. See 8.8

The location of the counter part of the OAM PDU Reception process is shown for illustration only. The exact set of packets to be counted is implementation specific, as described in [RFC6374].

[Note: Consistency with G.8121 should be verified]

Dataplane Loopback Sink process: see clause 8.9.2 in [ITU-T G.8121]

• Defects:

None.

• Consequent actions:

None.

• Defect correlations:

None.

• Performance monitoring:

None.

9.4.1.2 MTDe to MT Adaptation Functions (MTDe/MT_A)

The MPLS-TP MEP Diagnostic Adaptation Function (MTDe/MT_A) is described in Clause 9.4.1.2/G.8121.

9.5.19.4.2 Diagnostic Functions for MIPs

9.5.1.19.4.2.1 MPLS-TP MIP Diagnostic Trail Termination function (MTDi_TT)

The bidirectional MPLS-TP MIP DiagnosticTrail Termination (MTDi_TT) function is performed by a co-located pair of the MPLS-TP trail termination source (MTDi_TT_So) and sink (MTDi_TT_Sk) functions as shown in the figure belowFigure 9-11.

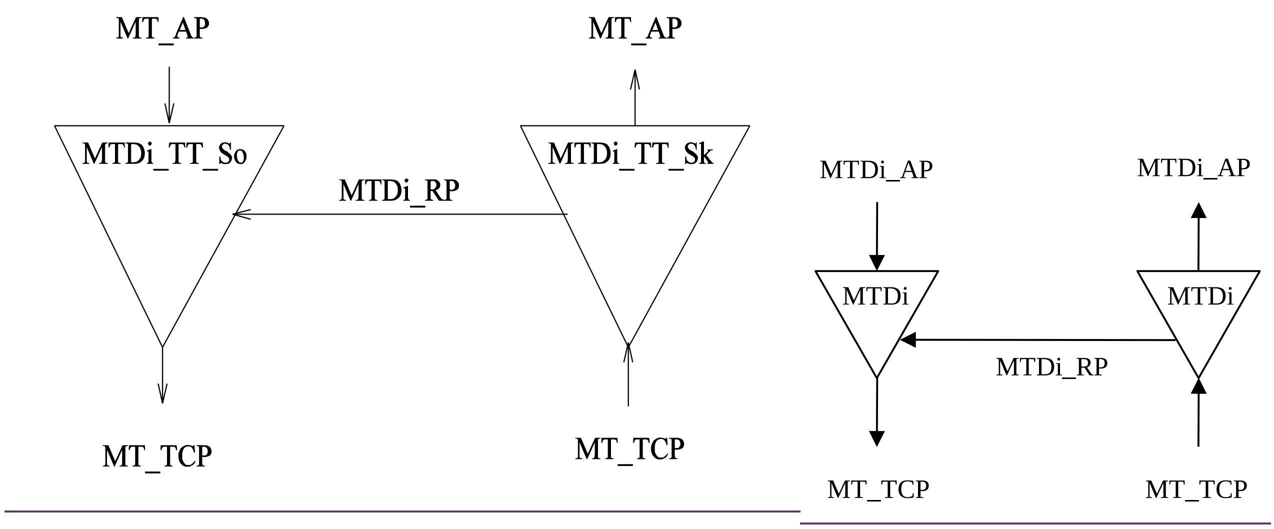


Figure 9-~~xx~~11/G.8121.2/Y.1381.2 - MTDi_TT
~~*[Note: MT_AP should be MTDi_AP]*~~

~~9.5.1.1.1~~9.4.2.1.1 MPLS-TP MIP Diagnostic Trail Termination Source function (MTDi_TT_So)

The MTDi_TT_So function adds MPLS-TP OAM to the MT_AI signal at its MT_AP.

The information flow and processing of the MTDi_TT_So function is defined with reference Figure 9-12 ~~to the figure below~~.

• **Symbol:**

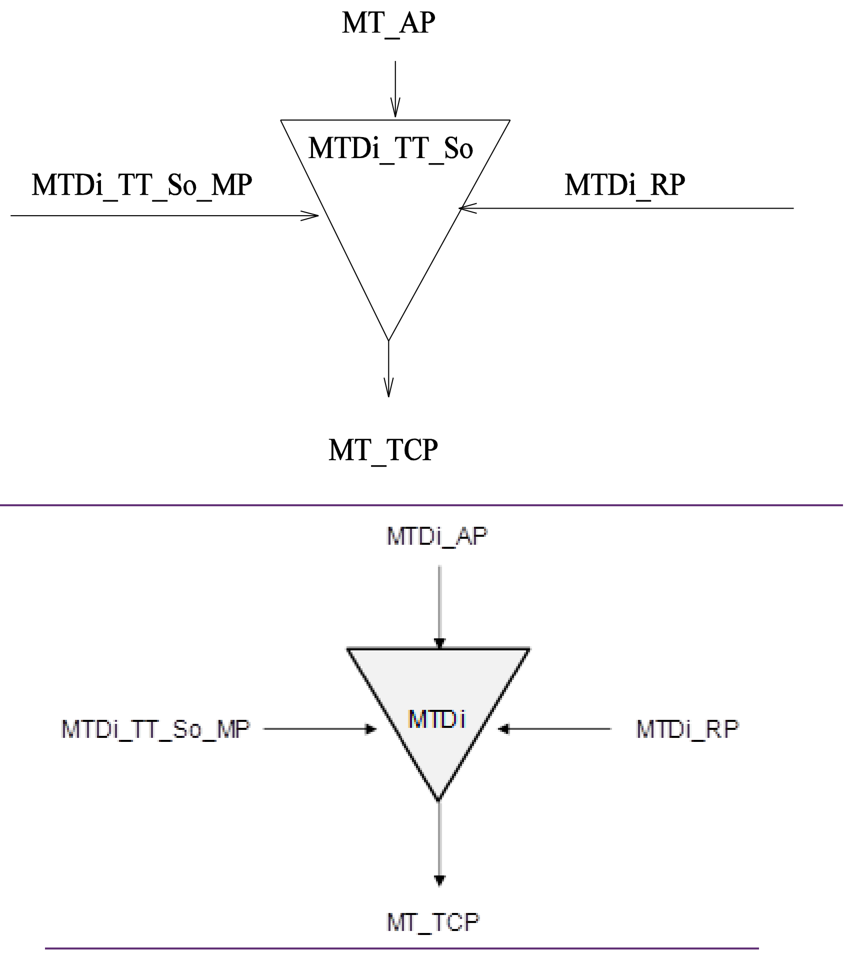


Figure 9-~~xx~~12/G.8121.2/Y.1381.2 - MTDi_TT_So symbol

~~*[Note: MT_AP should be MTDi_AP]*~~

• **Interfaces:**

Table 9-5*/G.8121.2/Y.1381.2 – MTDi_TT_So inputs and outputs

Input(s)	Output(s)
MTDi_AP: MTDi_AI_D MTDi_AI_iPHB MTDi_AI_oPHB **MTDi_RP:** MTDi_RI_LSPPing_Rsp MTDi_RI_CI **MTDi_TT_So_MP:** MTDi_TT_So_MI_Target_FEC MTDi_TT_So_MI_Ifnum MTDi_TT_So_MI_MTU MTDi_TT_Sk_MI_GAL_Enable MTDi_TT_So_MI_DP_Loopback_Enable	**MT_CP:** MT_CI_D MT_CI_oPHB MT_CI_iPHB

• **Processes:**

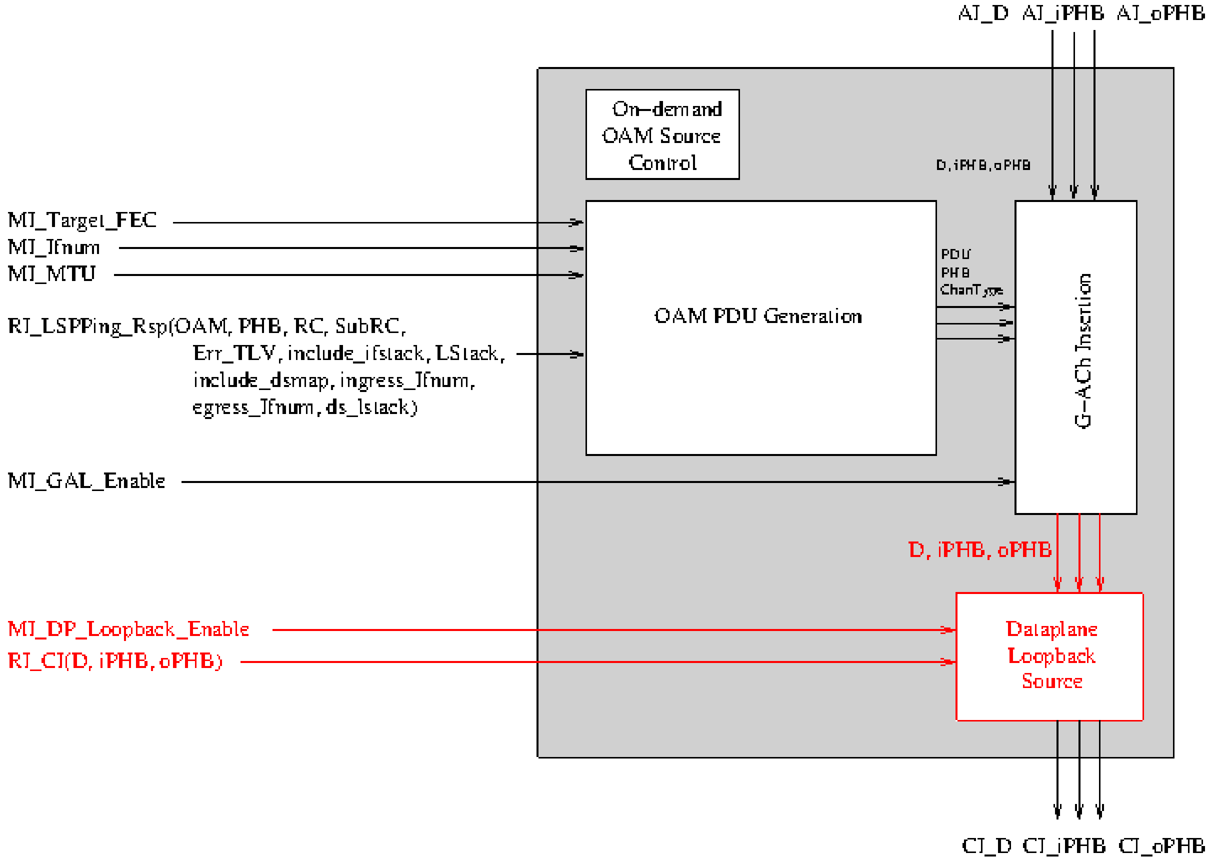


Figure 9-~~xx~~13/G.8121.2/Y.1381.2 - MTDi_TT_So Process
~~[Annex X, C.2025]~~

The processes associated with the MTDi_TT_So function are as depicted in the Figure below.

G-ACh Insertion: See 8.1.2 in [ITU-T G.8121].

OAM PDU Generation: This contains the following sub-processes as described in clause 8.8 [ITU-T G.8121]: On-demand CV Response Generation; OAM MuxOn-demand CV Response Generation and OAM Mux sub-processes. See 8.8

On-demand OAM Source Control: This process performs no operations.See 8.8

[Note: Consistency with G.8121 should be verified]

Dataplane Loopback Source process: see clause 8.9.1 in [ITU-T G.8121]

• **Defects:**

None.

• **Consequent actions:**

None.

• **Defect correlations:**

None.

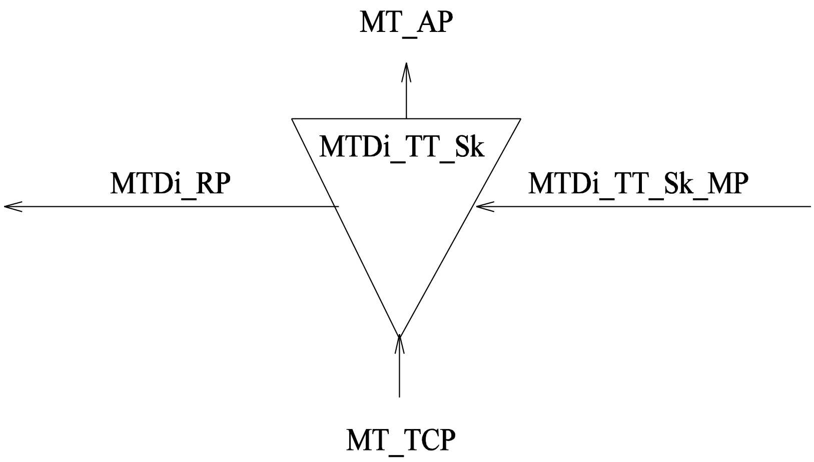
• **Performance monitoring:**

None.

9.5.1.1.29.4.2.1.2 MPLS-TP MIP Diagnostic Trail Termination Sink function (MTDi_TT_Sk)

The information flow and processing of the MTDi_TT_Sk function is defined with reference to the Figure 14below.

• **Symbol:**



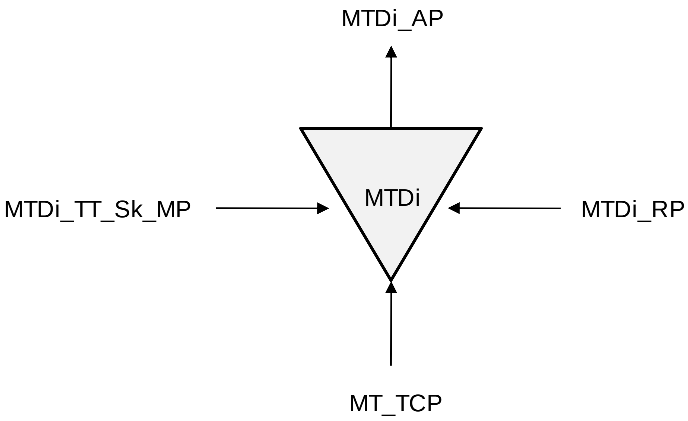


Figure 9-~~xx~~14/G.8121.2/Y.1381.2 - MTDi_TT_Sk symbol
~~[Note: MT_AP should be MTDi_AP]~~

• **Interfaces:**

Table ~~x~~9-6/G.8121.2/Y.1381.2 – MTDi_TT_Sk inputs and outputs

Input(s)	Output(s)
MT_TCP: MT_CI_D MT_CI_iPHB MT_CI_oPHB MT_CI_LStack **MTDi_TT_Sk_MP:** MTDi_TT_Sk_MI_FEC_Checking MTDi_TT_Sk_MI_GAL_Enable MTDi_TT_Sk_MI_DP_Loopback_Enable	**MTDi_AP:** MTDi_AI_D MTDi_AI_iPHB MTDi_AI_oPHB MTDi_AI_LStack **MTDi_RP:** MTDi_RI_LSPPing_Rsp MTDi_RI_CI

• **Processes:**

The processes associated with the MTDi_TT_Sk function are as depicted in ~~the figure below~~Figure 9-15.

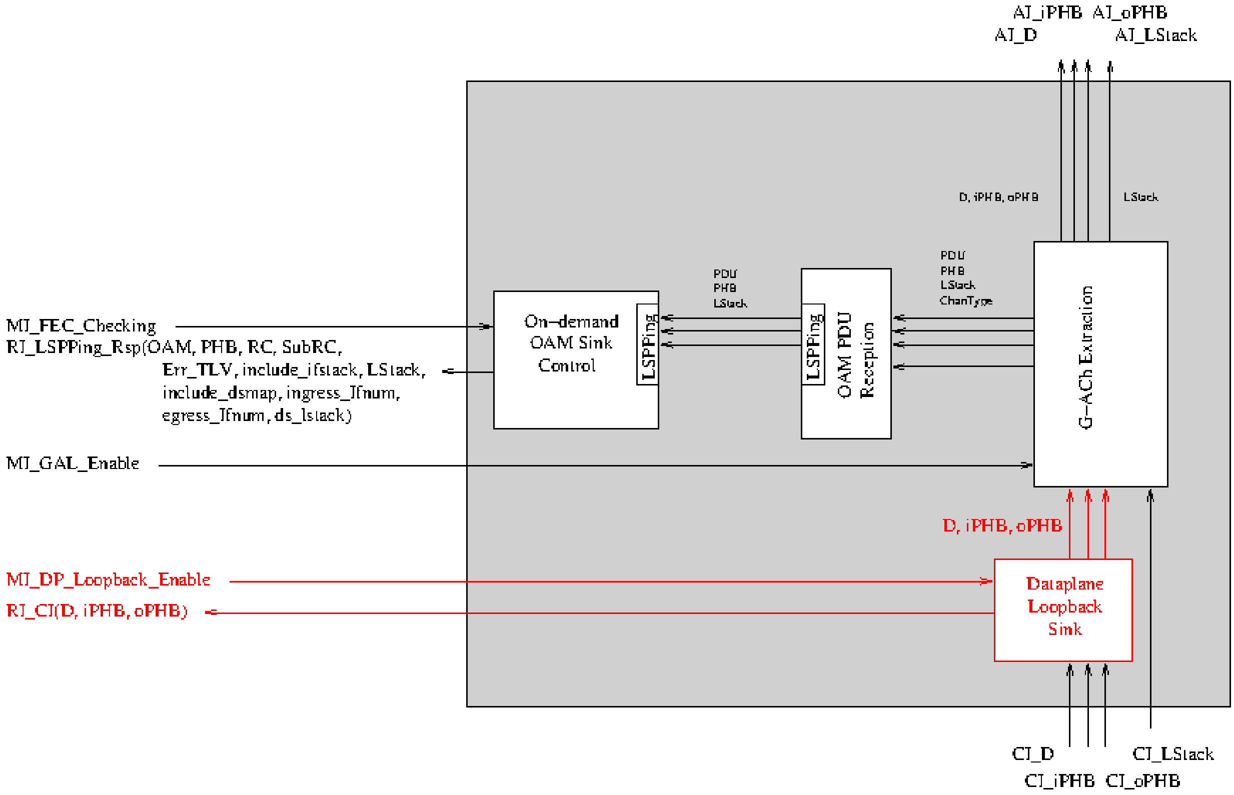


Figure 9-~~xx~~15/G.8121.2/Y.1381.2 - MTDi_TT_Sk process
~~**[updated per Annex XI, C.2025]**~~

G-ACh Extraction: see 8.1.3 in [ITU-T G.8121].

OAM PDU Reception: This contains the following sub-processes as described in clause 8.8 ~~of [ITU-T G.8121]~~: On-demand CV Reception;, OAM Demux ~~On-demand CV Reception and OAM Demux sub-processes. See 8.8~~

On-demand OAM Sink Control: This contains the following sub-processes as described in clause 8.8 of [ITU-T G.8121]: MIP On-demand CV Responder

~~MIP On-demand CV Responder sub-process. See 8.8~~

~~[Note: Consistency with G.8121 should be verified]~~

Dataplane Loopback Sink process: see clause 8.9.2 in [ITU-T G.8121]

• **Defects:**

None

• Consequent actions:

None

• Defect correlations:

None

• Performance monitoring:

None

9.5.1.29.4.2.2 MPLS-TP MIP Diagnostic Adaptation function (MTDi/MT_A)

The MPLS-TP MIP Diagnostic Adaptation Function (MTDi/MT_A) is described in Clause 9.4.2.2 of [ITU-T G.8121]/G.8121

10 MPLS-TP to Non-MPLS-TP client adaptation functions

This atomic functions are defined in clause 10 -in -[ITU-T G.8121]G.8121.

11 Non-MPLS-TP Server to MPLS-TP adaptation functions

These atomic functions are defined in clause 11 in [ITU-T G.8121]G.8121. They use the OAM protocol specific AIS insertion process and LCK generation process as defined in clause 8.6.2 and 8.6.3.

Appendix I

OAM Process and subprocesses

(This appendix does not form an integral part of this Recommendation)

Table I-1 indicates the relationship between processes and sub-processes and where these (sub-) processes are implemented to the termination functions (MT_TT, MTDe_TT, and MTDi_TT)

Table I-1/G.8121.2/Y.1381.2 OAM Process and subprocesses

Process	Sub-processes	MT_TT	MTDe_TT	MTDi_TT
Proactive OAM Source Control	CCCV Generation	Yes		
	Proactive PM Source Control	Yes		
	~~LI Source Control~~			
Proactive OAM Sink Control	CCCV Reception	Yes		
	LCK/AIS Reception	Yes		
	~~LI Sink Control~~	Yes		
	Proactive PM Sink Control	Yes		
	PM Responder			
On-demand OAM Source Control	On-demand CV Control		Yes	
	LI Source Control		Yes	
	On-demand PM Control		Yes	
On-demand OAM Sink Control	~~On-demand CV Control~~		Y	
	MIP On-demand CV Responder			Yes
	MEP On-demand CV Responder			
	LI Sink Control		Yes	
	~~On-demand PM Control~~		Yes	
	PM Responder		Y	
			Yes	
OAM PDU Generation	On-demand CV Request Generation		Yes	
	On-demand CV Response Generation		Yes	Yes
	OAM Mux	Yes	Yes	Yes
	LI Generation		Yes	
	PM Mux	Yes	Yes	
	PM Generation	Yes	Yes	

OAM PDU Reception	Session Demux	Yes		
	On-demand CV Reception		Yes	Yes
	OAM Demux	Yes	Yes	Yes
	LI Reception		Yes	
	PM Demux	Yes	Yes	
	PM Reception	Yes	Yes	