P802.1Qdd/D0.8
December 5, 2023

(Amendment to IEEE Std 802.1Q-2022 as amended by IEEE Std 802.1Qcz-2023, IEEE Std 802.1Qcw-2023, and IEEE Std 802.1Qcj-2023)

Draft Standard for Local and metropolitan area networks—

Bridges and Bridged Networks

Amendment: Resource Allocation Protocol

Prepared by the

Time-Sensitive Networking (TSN) Task Group of IEEE 802.1

Sponsor

LAN/MAN Standards Committee
of the
IEEE Computer Society

This and the following cover pages are not part of the draft. They provide revision and other information for IEEE 802.1 Working Group members and will be updated as convenient. **New participants: Please read these cover pages**, they contain information that should help you contribute effectively to this standards development project. The Introduction to the current draft and previous drafts should be useful to all readers.

The text proper of this draft begins with the Title page.

Participation in 802.1 standards development

All participants in IEEE 802.1 activities should be aware of the Working Group Policies and Procedures, and their obligations under the IEEE Patent Policy, the IEEE Standards Association (SA) Copyright Policy, and the IEEE SA Participation Policy. For information on these policies see 1.ieee802.org/rules/ and the slides presented at the beginning of each of our Working Group and Task Group meeting.

The IEEE SA PAR (Project Authorization Request) and CSD (Criteria for Standards Development established by IEEE 802) are summarized in these cover pages and links are provided to the full text of both PAR and CSD. As part of the IEEE 802® process, the text of the PAR and CSD of each project is reviewed regularly to ensure their continued validity. A vote of "Approve" on this draft is also an affirmation that the PAR and CSD for this project are still valid.

Comments on this draft are encouraged. NOTE: All issues related to IEEE standards presentation style, formatting, spelling, etc. are routinely handled between the 802.1 Editor and the IEEE Staff Editors prior to publication, after balloting and the process of achieving agreement on the technical content of the standard is complete. Readers are urged to devote their valuable time and energy only to comments that materially affect either the technical content of the document or the clarity of that technical content. Comments should not simply state what is wrong, but also what might be done to fix the problem.

Full participation in the work of IEEE 802.1 requires attendance at IEEE 802 meetings. Information on 802.1 activities, working papers, and email distribution lists etc. can be found on the 802.1 Website:

http://ieee802.org/1/

Use of the email distribution list is not presently restricted to 802.1 members, and the working group has a policy of considering comments from all who are interested and willing to contribute to the development of the draft. Individuals not attending meetings have helped to identify sources of misunderstanding and ambiguity in past projects. The email lists exist primarily to allow the members of the working group to develop standards, and are not a general forum. All contributors to the work of 802.1 should familiarize themselves with the IEEE patent policy and anyone using the email distribution list will be assumed to have done so. Information can be found at http://standards.ieee.org/db/patents/

Comments on this draft may be sent to the 802.1 email exploder, to the Editors, or to the Chairs of the 802.1 Working Group and Time-Sensitive Networking (TSN) Task Group.

Feng Chen
Editor, P802.1Qdd
Email:chen.feng@siemens.com

Mick Seaman
Editor, IEEE Std 802.1Q
Email:mickseaman@gmail.com

Janos Farkas
Chair, 802.1 TSN Task Group
Email:Janos.Farkas@ericsson.com

Glenn Parsons
Chair, 802.1 Working Group
+1 514-379-9037
Email: glenn.parsons@ericsson.com

NOTE: Comments whose distribution is restricted in any way cannot be considered, and may not be acknowledged.

All participants in IEEE standards development have responsibilities under the IEEE patent policy and should familiarize themselves with that policy, see http://standards.ieee.org/about/sasb/patcom/materials.html

As part of our IEEE 802 process, the text of the PAR and CSD (Criteria for Standards Development, formerly referred to as the 5 Criteria or 5C's) is reviewed on a regular basis in order to ensure their continued validity. A vote of "Approve" on this draft is also an affirmation by the balloter that the PAR is still valid.

PAR (Project Authorization Request) and CSD

As part of our IEEE 802 process, the text of the PAR and CSD should be reviewed on a regular basis in order to ensure their continued validity. A vote of “Approve” on this draft is assumed also to be an affirmation by the balloter that the text of the PAR and CSD are still valid.

The following information is taken from the PAR for P802.1Qdd that was approved by the IEEE Standards Association on September 27, 2018 and will expire on **December 31, 2025**. The full text of the PAR can be found at P802.1Qdd-PAR.

Scope of the Project:

This amendment specifies protocols, procedures, and managed objects for a Resource Allocation Protocol (RAP) that uses the Link-local Registration Protocol (LRP) and supports and provides backwards compatibility with the stream reservation and quality of service capabilities, controls and protocols specified in IEEE Std 802.1Q. RAP provides support for accurate latency calculation and reporting, can use redundant paths established by other protocols, and is not limited to bridged networks.

Need for the Project:

A signaling protocol that performs distributed and dynamic resource management and admission control is an essential component for automatic configuration in bridged LANs requiring latency and bandwidth guarantees. Current IEEE 802.1Q Multiple Stream Reservation Protocol (MSRP) is constrained by the capability of its underlying IEEE 802.1Q Multiple Registration Protocol (MRP) and does not efficiently support a large reservation database. For use in distributed stream reservation, IEEE 802.1Q MSRP does not make use of all available Quality of Service provisions and does not support reservation for the streams in need of high availability by use of the technologies specified in IEEE Std 802.1CB. The proposed amendment will address these issues.

The following information is taken from the CSD that were approved by 802.1 and the 802 EC at PAR submission and can be found at P802.1Qdd-CSD.

CSD broad market potential [extract]:

The original version of IEEE 802.1Q Multiple Stream Reservation Protocol (MSRP) has been successfully and widely accepted by the professional, industrial, consumer, and automotive markets as an essential tool to realize automatic stream setup with dynamic resource allocation. The success of IEEE 802.1Q MSRP has expanded the requirements on that protocol beyond that capability. RAP addresses the expanded markets.

Multiple vendors and users for industrial automation, professional audio-video, automotive and other systems requiring a protocol to signal the resource reservation along the end-to-end paths of streams for time-sensitive applications will participate in the development of the project.

Economic feasibility [extract]:

The well-established balance between infrastructure and attached stations will not be changed by the proposed amendment.

The amendment will specify an application for LRP and add no additional hardware costs to bridges and end stations beyond the minimal and firmly bounded resources consumed by LRP.

The cost factors, including installation and operational costs are well-known from existing IEEE 802.1Q MSRP that is built on IEEE 802.1Q MRP. The proposed amendment will specify an application running over LRP that supports a larger database with fewer message exchanges and thus will provide better economic feasibility than IEEE 802.1Q MSRP built on IEEE 802.1Q MRP.

Introduction to the current draft and previous drafts

This introduction is not part of the draft, and should not be the subject of ballot comments.

D0.8 (current)

Draft 0.8 was prepared based on the results of comment resolution for the TG ballot on D0.7. The major changes from D0.7 include:

— Added support for seamless redundancy.
— A new informative Annex Y for resource allocation examples.
— Added support for stream rank.
— Added event and procedures to handle resource changes.
— Added in 51.9 definition of RAP Failure Codes
— Restructured per-stream managed objects.
— RAP Architecture changed back to Baggy Pants style.
— Added in 51.3.9 relationship to MSRP

D0.7

Draft 0.7 was prepared based on the results of comment resolution for the TG ballot on D0.6. The major changes from D0.6 include:

— Defined managed objects for priority regeneration in 12.35.6.
— Architecture in 51.3.1 changed to incorporate the change made to RAP Participant.
— Model of operation in 51.3 reworked to provide a more comprehensive introduction.
— Defined Ingress Blocking in 51.3.4.1.3 to take Ingress Filtering into account.
— RESI (RSI in D0.6) primitives reworked to avoid using attribute TLVs as parameters.
— RAP Participant component (incl. RPSI) redefined to be on a per-device basis, while RAP Participant state machine remains on a per-Port basis.
— Five RAP Participant state machines in D0.6 combined into a single one.
— Incorporated new ECP managed objects and MIBs for Maintenance item #0248.

D0.6

Draft 0.6 was prepared based on the results of comment resolution for the TG ballot on D0.5. The major changes from D0.5 include:

— Inserted a subclause for conventions in 99.2.
— Aligned variable naming in 99.5 and 99.6 to the conventions defined in 99.2.
— Reworked RAP Participant in 99.7 and RAP Propagator in 99.8 (see related presentation in dd-chen-rework-rap-propagator-0222-v01.pdf).
— Added in 99.8 procedures for computation of per-hop worst-case latency for SP and ATS.
— Integrated the RAAI functions defined by D0.5 into RAP Propagator in 99.8.
— In clause 12, added managed objects for RAP Participant, and reworked the ones for RAP Propagator in accordance with the reworked RAP Propagator in 99.8.
— Updated Annex Z.

D0.5

Draft 0.5 was prepared based on the results of comment resolution for the TG ballot on D0.4. The major changes from D0.4 include:

— Managed objects for RAP Propagator added to clause 12.
— More text and figures added to 99.2.

— In 99.4, Failure Information sub-TLV removed from Redundancy Control sub-TLV and Listener Attach attribute, RA class attribute restructured, Accumulated Min Latency added to Talker Announce attribute, Interval parameter added to MSRP TSpec.
— Specification of RAP Service Interface added to 99.5.
— RAP Propagator variables, signaling and reservation functions with support for Single-context TA and Multi-context TA (only E2E FRER) added to 99.7.
— APIs for interaction with FDB and queue resources added to 99.8.

D0.4

Draft 0.4 was prepared for a 3rd Task Group ballot, as a result of discussion on Draft 0.3 at the IEEE 802.1 November 2020 Plenary Session. Since D0.3 was not reviewed by a Task Group balloting, D0.4 retains change bars contained in D0.3. The major changes from D0.3 include:

— Combined 99.2.1 and 99.2.2, and added a table to illustrate the RAP system classes and the associated RAP Instance types. Added two subclasses for RAP Proxy system. Clause 3 also reworked, by removing terms RAP Bridges and RAP end stations, and moving RAP instance related terms into 99.2.1.
— RAP architecture diagrams in 99.2.1 reworked.
— A note added in Clause 99.6.3.3.1 to indicate that the MAC_Operational value contained in the localTargetPortOper variable is also used by a CFM MEP to indicate a continuity fault.

D0.3

Draft 0.3 was prepared based on the results of comment resolution for the 2nd TG ballot on D0.2. The major changes from D0.2 are as follows:

— General terms for relays and end systems removed, making RAP focus on 802.1 systems (bridges and end stations) only, because many protocol functions and procedures in RAP, in particular attributes encoding and processing, are tied to bridged networks, such as VLANs, priority regeneration.
— RAP architecture in 99.2.2 reworked, in both text and figures.
— A RA Class Template Information sub-TLV added to include RTID and other RA class operational parameters if required by that RTID for exchanges in RA class attributes.
— Specification of RAP Participant provided in 99.6. Please also see the presentation for an introduction to this subclause << dd-chen-D0-3-rap-participant-1120-v01.pdf >>.
— Refer to change bars for other changes due to restructuring and rewording.

D0.2

Draft 0.2 was prepared for the 2nd TG ballot, based on the results of comment resolution for the first TG ballot on D0.1. As D0.2 is the first draft created with 802.1Q FM templates and contains many changes from D0.1, the editor decided not to show the change bars in this draft and will provide them from next draft. The major changes from D0.1 are listed below:

— Clause 3 (definitions) reworked, in particular terms for relay and end system added
— Subclause 99.2 used to describe model of operation and basic concept for general relays/end systems without using conformance statements. Definitive specification for Bridges and end stations starting from 99.3.
— RAP architecture in 99.2.1 reworked, in particular a new component RAL added.
— Description of RA class added in 99.2.2, in particular RA class specification and RSID.
— Description of signaling processes added in 99.2.3, in particular the Multiple-context Talker Announce used by reservations for redundancy, which is explained with examples in the deck <<dd-chen-multiple-context-talker-announce-examples-0520-v01.pdf >>.
— RAP variable definitions in 99.6 reworked
— RAP attribute and TLV definitions in 99.7 reworked, in particular the Token bucket TSpec and the Redundancy control sub-TLV
— RA class protection procedures defined in 99.9.1.1

— VLAN context and Talker pruning defined under 99.10.1.

D0.1

Draft 0.1 was prepared for the first Task Group ballot.

The editor created this draft by using MS word, thus could not make it 100% comply with the format of the official 802.1Q drafts FM templates. As a result, cross-reference is not yet available in this draft (apologies from the editor for inconvenience caused to reviewers). As this is the 1st TG ballot, change bars are not shown for this version.

The major additions/changes made in D0.1 are included in the following (sub)clauses, on which comments and suggestions from reviewers are particularly expected:

— 3. Definitions
— 99.1.1 RAP terminology
— 99.1.2 RAP architecture
— 99.1.5 Reservation protection
— 99.4 Definitions of RAP parameters
— 99.5 RAP attributes and TLV encoding

Some notes about D0.1:

a) Many definitions added in Clause 3 and 99.1.1 (suggest reading them first!)
b) The new terms “RA class (3.x.1)” and “RAP Protection Port (3.x.2)” introduced to replace “SR class” and “RAP domain boundary port” originally used in D0.0.
c) “Domain” completely removed (Note: the editor withdrew the proposal about RAP domain detection presented in dd-chen-RAP-domain-0919-v01.pdf). Instead, a RAP protection port is detected based on whether the neighbor supports the same RA class priority as mine; no other RA class parameters like shaper will be checked. A RAP protection port status determines only where and how to apply priority regeneration and has no influence on stream reservation. This implies that RAP may allow a stream reservation to be made across a heterogeneous data plane, which is associated with the same priority but not necessarily with the same shaper in each bridge. Whether reservation for streaming over such a data plane can be made is decided on a per stream basis during each reservation, which is a matter of bridge-local decision.

D0.0

Draft 0.0 was prepared by the editor as an Editor’s first draft. Everything in this draft can be considered a contribution to the Time-Sensitive Networking Task Group by the editor; nothing has been approved by the Task Group or Working Group. This initial draft includes the following:

a) A list of the existing clauses/subclauses in 802.1Q (including some .1Q amendments not yet incorporated in 802.1Q-2018 but relevant to this project) to be amended and the new ones to be created by this project.
b) The main body of RAP is specified in a new clause with a temporary clause number 99, which includes in D0.0 the following contents:
    1) Subclause 99.2.2 shows a figure for the proposed RAP architecture and a short description for each component.
    2) Subclause 99.4 defines the parameters used by RAP, divided into 4 groups.
    3) Subclause 99.5 specifies the RAP attributes and their encoding in TLV format.
c) Annex Z documents the objectives and non-objectives proposed so far and includes links to previous contributions.

P802.1Qdd/D0.8

December 5, 2023

(Amendment to IEEE Std 802.1Q-2022 as amended by IEEE Std 802.1Qcz-2023, IEEE Std 802.1Qcw-2023, and IEEE Std 802.1Qcj-2023)

Draft Standard for Local and metropolitan area networks—

Bridges and Bridged Networks

Amendment: Resource Allocation Protocol

Prepared by the

Time-Sensitive Networking (TSN) Task Group of IEEE 802.1

Sponsor

LAN/MAN Standards Committee
of the
IEEE Computer Society

Abstract: This amendment to IEEE Std 802.1Q™-2022 specifies a Resource Allocation Protocol (RAP) that uses the Link-local Registration Protocol (LRP) and supports and provides backwards compatibility with the stream reservation and quality of service capabilities, controls and protocols specified in IEEE Std 802.1Q.

Keywords: Bridged Local Area Networks, Local Area Networks (LANs), MAC Bridges, Metropolitan Area Networks, Virtual Bridged Local Area Networks (virtual LANs), Time-Sensitive Networking (TSN), IEEE Std 802.1CS™-2020, Link-local Registration Protocol (LRP), IEEE Std 802.1CB™-2017, Frame Replication and Elimination for Reliability (FRER), resource reservation.

PDF: ISBN 978-X-XXX-XXX-X STDXXXXX
Print: ISBN 978-X-XXX-XXX-X STDPDXXXXX

Important Notices and Disclaimers Concerning IEEE Standards Documents

IEEE Standards documents are made available for use subject to important notices and legal disclaimers. These notices and disclaimers, or a reference to this page (https://standards.ieee.org/ipr/disclaimers.html), appear in all standards and may be found under the heading "Important Notices and Disclaimers Concerning IEEE Standards Documents."

Notice and Disclaimer of Liability Concerning the Use of IEEE Standards Documents

IEEE Standards documents are developed within IEEE Societies and subcommittees of IEEE Standards Association (IEEE SA) Board of Governors. IEEE develops its standards through an accredited consensus development process, which brings together volunteers representing varied viewpoints and interests to achieve the final product. IEEE Standards are documents developed by volunteers with scientific, academic, and industry-based expertise in technical working groups. Volunteers are not necessarily members of IEEE or IEEE SA and participate without compensation from IEEE. While IEEE administers the process and establishes rules to promote fairness in the consensus development process, IEEE does not independently evaluate, test, or verify the accuracy of any of the information or the soundness of any judgments contained in its standards.

IEEE makes no warranties or representations concerning its standards, and expressly disclaims all warranties, express or implied, concerning this standard, including but not limited to the warranties of merchantability, fitness for a particular purpose and non-infringement. In addition, IEEE does not warrant or represent that the use of the material contained in its standards is free from patent infringement. IEEE standards documents are supplied "AS IS" and "WITH ALL FAULTS."

Use of an IEEE standard is wholly voluntary. The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard.

In publishing and making its standards available, IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity, nor is IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing any IEEE Standards document, should rely upon his or her own independent judgment in the exercise of reasonable care in any given circumstances or, as appropriate, seek the advice of a competent professional in determining the appropriateness of a given IEEE standard.

IN NO EVENT SHALL IEEE BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO: THE NEED TO PROCURE SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE PUBLICATION, USE OF, OR RELIANCE UPON ANY STANDARD, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE AND REGARDLESS OF WHETHER SUCH DAMAGE WAS FORESEEABLE.

Translations

The IEEE consensus development process involves the review of documents in English only. In the event that an IEEE standard is translated, only the English version published by IEEE is the approved IEEE standard.

Official statements

A statement, written or oral, that is not processed in accordance with the IEEE SA Standards Board Operations Manual shall not be considered or inferred to be the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal position of IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that the presenter's views should be considered the personal views of that individual rather than the formal position of IEEE, IEEE SA, the Standards Committee, or the Working Group. Statements made by volunteers may not represent the formal position of their employer(s) or affiliation(s).

Comments on standards

Comments for revision of IEEE Standards documents are welcome from any interested party, regardless of membership affiliation with IEEE or IEEE SA. However, **IEEE does not provide interpretations, consulting information, or advice pertaining to IEEE Standards documents**.

Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Since IEEE standards represent a consensus of concerned interests, it is important that any responses to comments and questions also receive the concurrence of a balance of interests. For this reason, IEEE and the members of its Societies and subcommittees of the IEEE SA Board of Governors are not able to provide an instant response to comments, or questions except in those cases where the matter has previously been addressed. For the same reason, IEEE does not respond to interpretation requests. Any person who would like to participate in evaluating comments or in revisions to an IEEE standard is welcome to join the relevant IEEE working group. You can indicate interest in a working group using the Interests tab in the Manage Profile & Interests area of the IEEE SA myProject system.[a] An IEEE Account is needed to access the application.

Comments on standards should be submitted using the Contact Us form.[b]

Laws and regulations

Users of IEEE Standards documents should consult all applicable laws and regulations. Compliance with the provisions of any IEEE Standards document does not constitute compliance to any applicable regulatory requirements. Implementers of the standard are responsible for observing or referring to the applicable regulatory requirements. IEEE does not, by the publication of its standards, intend to urge action that is not in compliance with applicable laws, and these documents may not be construed as doing so.

Data privacy

Users of IEEE Standards documents should evaluate the standards for considerations of data privacy and data ownership in the context of assessing and using the standards in compliance with applicable laws and regulations.

Copyrights

IEEE draft and approved standards are copyrighted by IEEE under US and international copyright laws. They are made available by IEEE and are adopted for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making these documents available for use and adoption by public authorities and private users, neither IEEE nor its licensors waive any rights in copyright to the documents.

[a] Available at: https://development.standards.ieee.org/myproject-web/public/view.html#landing.
[b] Available at: https://standards.ieee.org/content/ieee-standards/en/about/contact/index.html.

[c] Available at: https://ieeexplore.ieee.org/browse/standards/collection/ieee.
[d] Available at: https://standards.ieee.org/standard/index.html.
[e] Available at: https://standards.ieee.org/about/sasb/patcom/materials.html.

IMPORTANT NOTICE

Participants

<<The following lists will be updated in the usual way prior to publication>>

At the time this standard was submitted to the IEEE-SA Standards Board for approval, the IEEE 802.1 Working Group had the following membership:

Introduction

This introduction is not part of IEEE Std 802.1Qdd™-20XX, IEEE Standard for Local and metropolitan area networks— Bridges and Bridged Networks—Amendment: Resource Allocation Protocol.

This amendment to IEEE Std 802.1Q-2022 specifies a Resource Allocation Protocol (RAP) that uses the Link-local Registration Protocol (LRP) and supports and provides backwards compatibility with the stream reservation and quality of service capabilities, controls and protocols specified in IEEE Std 802.1Q.

This standard contains state-of-the-art material. The area covered by this standard is undergoing evolution. Revisions are anticipated within the next few years to clarify existing material, to correct possible errors, and to incorporate new related material. Information on the current revision state of this and other IEEE 802 standards may be obtained from

Secretary, IEEE-SA Standards Board
445 Hoes Lane
Piscataway, NJ 08854-4141
USA

Contents

Figures

Tables

IEEE Standard for Local and metropolitan area networks—

Bridges and Bridged Networks

Amendment: Resource Allocation Protocol

[This amendment is based on IEEE Std 802.1Q™-2022 as amended by IEEE Std 802.1Qcz™-2023, IEEE Std 802.1Qcw™-2023 and IEEE Std 802.1Qcj™-2023.]

NOTE—The editing instructions contained in this amendment define how to merge the material contained here into the base document and its other amendments to form the new comprehensive standard.

Editing instructions are shown in ***bold italic***. Four editing instructions are used: change, delete, insert, and replace. ***Change*** is used to make corrections in existing text or tables. The editing instruction specifies the location of the change and describes what is being changed either by using ~~strikethrough~~ (to remove old material) and underscore (to add new material). ***Delete*** removes existing material. ***Insert*** adds new material without disturbing the existing material. Insertions may require renumbering. If so, renumbering instructions are given in the editing instruction. ***Replace*** is used to make changes in figures or equations by removing the existing figure or equation and replacing it with a new one. Editing instructions, change markings, and this NOTE will not be carried over into future editions because the changes will be incorporated into the base standard.

1. Overview

1.3 Introduction

Insert the following text at the end of subclause 1.3:

This standard specifies protocols, procedures, and managed objects for a Resource Allocation Protocol (RAP) that uses the Link-local Registration Protocol (LRP), as specified in IEEE Std 802.1CS, and supports and provides backwards compatibility with the stream reservation and quality of service capabilities, controls and protocols specified in IEEE Std 802.1Q. RAP provides support for accurate latency calculation and reporting, can use redundant paths established by other protocols, and is not limited to bridged networks.

2. Normative references

Insert the following reference in alphanumeric order:

IEEE Std 802.1CS™, IEEE Standard for Local and metropolitan area networks—Link-local Registration Protocol.

3. Definitions

Insert the following text after "-station":

This standard makes use of the terms defined in IEEE Std 802.1CS:

— Controlled system
— Native system
— Portal
— Proxy system
— record
— target port
— local target port
— neighbor target port

This standard makes use of the terms defined in IEEE Std 802.1CB:

— Compound Stream
— Member Stream

Insert the following definitions in alphabetic order, renumbering as appropriate:

<< Editor's note: The temporal numbering of the items defined in this clause uses continuous numbers starting from 3.x.1, just for ease of reference within the current draft. >>

3.1 FRER-capable RAP Bridge: A RAP Native or RAP Controlled Bridge supporting the Stream identification function and the FRER functions of IEEE Std 802.1CB.

3.2 FRER-capable RAP end station: A RAP Native or RAP Controlled end station supporting the Stream identification function and the FRER functions of IEEE Std 802.1CB.

3.3 resource allocation (RA) class: A priority of a traffic class or a set of traffic classes whose bandwidth and queue resources are reserved by the Resource Allocation Protocol (RAP) for time-sensitive streams.

3.4 Resource Allocation Protocol (RAP): A protocol providing resource reservation for transmission of time-sensitive streams.

3.5 RAP Bridge Proxy: A RAP Proxy for Bridges.

3.6 RAP Controlled Bridge: A Virtual Local Area Network (VLAN) Bridge operating as a Controlled system from a RAP Proxy.

3.7 RAP Controlled end station: An end station operating as a Controlled system from a RAP Proxy.

3.8 RAP Controlled station: A RAP Controlled end station or a RAP Controlled Bridge.

3.9 RAP End instance: A RAP instance for an end station.

3.10 RAP Endpoint: The component of a RAP End Instance that is responsible for generation and processing of RAP attributes for an end station.

3.11 RAP End Station Proxy: A RAP Proxy for end stations.

3.12 RAP instance: An application instance of the Link-local Registration Protocol (LRP) that provides RAP functionality for an end station or Bridge.

3.13 RAP Native Bridge: A Virtual Local Area Network (VLAN) Bridge operating as a Native system using the Resource Allocation Protocol (RAP).

3.14 RAP Native end station: An end station operating as a Native system using the Resource Allocation Protocol (RAP).

3.15 RAP Native station: A RAP Native Bridge or a RAP Native end station.

3.16 RAP Participant: The component of a RAP End Instance or RAP Relay Instance that provides per-Port attribute declaration and registration services to the RAP Endpoint or the RAP Propagator, respectively, and uses the services provide by the underly LRP.

3.17 RAP Propagator: The component of a RAP Relay Instance that is responsible for generation, processing and propagation of RAP attributes for a Bridge.

3.18 RAP Proxy: A Proxy system using the Resource Allocation Protocol (RAP).

3.19 RAP Relay instance: A RAP instance for a Bridge.

4. Abbreviations

Insert the following abbreviations in alphabetic order:

FRER	IEEE Std 802.1CB Frame Replication and Elimination for Reliability
LRP	IEEE Std 802.1CS Link-local Registration Protocl
RA	Resource Allocation
RAP	Resource Allocation Protocol
RESI	RAP Endpoint Service Interface (51.6.2)
RPSI	RAP Participant Service Interface (51.7.2)
RTID	Resource Allocation Class Template Identifier

5. Conformance

5.4 VLAN Bridge component requirements

5.4.1 VLAN Bridge component options

Insert the following item at the end of the letter list in 5.4.1:

a) Support Resource Allocation Protocol (RAP), as specified in Clause 51.

Insert the following new subclause at the end of 5.4 using the next available subclause number:

5.4.x Resource Allocation Protocol (RAP) requirements

<< Editor's note: a placeholder for specifying conformance for RAP Bridges. >>

Insert the following new subclause at the end of Clause 5 using the next available subclause number:

5.34 End station requirements—RAP

<< Editor's note: a placeholder for specifying conformance for RAP end stations. >>

6. Support of the MAC Service

6.9.4 Regenerating priority

Insert the following text at the end of 6.9.4:

For Bridges that supports RAP, an *RAP Priority Regeneration Override Table (12.35.7)* is maintained for each reception Port. This table contains an entry for each received priority value that is associated with an RA class (51.3.2) supported by the Bridge, and specifies the priority overriding value to be used as the regenerated priority, in preference to the value in the Priority Regeneration table, when the domainBoundaryStatus parameter [item c) in 51.8.4.8] associated with that RA class on that Port has the value TRUE. Any priority overriding value contained in such a table shall not be associated with any RA class supported by the Bridge.

12. Bridge management

12.27 Edge control Protocol (ECP) management

Change 12.27.1, including Table 12-31, as follows:

12.27.1 ECP table entry

The management operations that can be performed on the ECP table entry managed object is as follows:

a) Read ECP table entry
b) Update ECP table entry

ECP table entries (Table 12-31) are created or deleted implicitly as a result of the creation or deletion of other port objects.

Table 12-31—ECP table entry

Name	Data type	Operations supported[a]	Conformance[b]	References
ecpComponentID	ComponentID	R	BE	12.4.1.5
ecpPortNumber	Port Number	R	BE	12.4.2
ecpOperAckTimerInit	timer exp	R	BE	D.2.12, ~~43.3.7.1~~43.3.6.2
ecpOperMaxRetries	unsigned [0...7]	R	BE	D.2.12, 43.3.7.4
ecpTxFrameCount	counter	R	BE	Clause 43
ecpTxRetryCount	counter	R	BE	Clause 43
ecpTxFailures	counter	R	BE	Clause 43
ecpRxFrameCount	counter	R	BE	Clause 43
ecpProposedR	unsigned [0...7]	RW	BE	Clause 43
ecpProposedRTE	timer exp	RW	BE	Clause 43
ecpDestinationAddress	MAC Address	RW	BE	Clause 43

[a] R = read only access; RW = Read/Write access.
[b] B = required for an EVB Bridge system; E = required for an EVB station system.

Insert the following subclause 12.35 at the end of Clause 12:

12.35 Resource Allocation Protocol (RAP) management

The Bridge enhancements for support of RAP are defined in Clause 51.

This managed resource comprises the following objects:

a) RAP Participant Table in 12.35.1.
b) RAP Propagator Bridge Table in 12.35.2.
c) RAP Propagator Port Table in 12.35.3.
d) RA Class Bridge Table in 12.35.4.
e) RA Class Port Table in 12.35.5.
f) RA Class Port Pair Table in 12.35.6.
g) RAP Priority Regeneration Override Table in 12.35.7.
h) RAP Redundancy Context Table in 12.35.8.
i) RAP Talker Announce Registration Port Table in 12.35.9.
j) RAP Talker Announce Declaration Port Table in 12.35.10.
k) RAP Listener Attach Registration Port Table in 12.35.11.
l) RAP Listener Attach Declaration Port Table in 12.35.12.

12.35.1 RAP Participant Table

There is one RAP Participant Table per Port of a Bridge component. The table contains a set of parameters for the operation of a RAP Participant associated with a Bridge Port, as detailed in Table 12-42.

Table 12-42—RAP Participant Table

Name	Data type	Operations supported[a]	Conformance[b]	References
participantEnabled	Boolean	RW	B	51.7.4.2
neighborDiscoveryMode	Enumerated	RW	B	51.7.4.3
helloTime	integer	RW	B	51.7.4.4
completeListTimerReset	integer	RW	B	51.7.4.5
exploreHelloRecvEnabled	Boolean	RW	B	51.7.4.6
localTargetPort	TargetPort	RW	B	51.7.4.7.1 51.7.4.7.2
staticNeighorTargetPort	TargetPort	RW	B	51.7.4.7.1 51.7.4.7.3
lldpNeighborTargetPort	TargetPort	R	B	51.7.4.7.1 51.7.4.7.4
neighborMismatch	Boolean	R	B	51.7.4.8
portalConnected	Boolean	R	B	51.7.4.12

[a] R = read only access; RW = Read/Write access.
[b] B = required for Bridge or Bridge component support of RAP.

12.35.2 RAP Propagator Bridge Table

There is one RAP Propagator Bridge Table per Bridge component. The table contains a set of parameters for the operation of the RAP Propagator associated with a Bridge, as detailed in Table 12-43.

Table 12-43—RAP Propagator Bridge Table

Name	Data type	Operations supported[a]	Conformance[b]	References
frerCapable	Boolean	R	B	51.8.4.11
maxProcessingDelay	integer	R	B	51.8.4.12
minProcessingDelay	integer	R	B	51.8.4.13

[a] R = read only access; RW = Read/Write access.
[b] B = required for Bridge or Bridge component support of RAP.

12.35.3 RAP Propagator Port Table

There is one RAP Propagator Port Table per Port of a Bridge component. The table contains a set of parameters for the operation of the RAP Propagator on a Bridge Port, as detailed in Table 12-44.

Table 12-44—RAP Propagator Port Table

Name	Data type	Operations supported[a]	Conformance[b]	References
streamDaPruningEnabled	Boolean	RW	B	51.8.4.7 b)
maxInterferingFrameSize	integer	R	B	51.8.4.7 e)
maxPropagationDelay	integer	R	B	51.8.4.7 f)
minPropagationDelay	integer	R	B	51.8.4.7 g)

[a] R = read only access; RW = Read/Write access.
[b] B = required for Bridge or Bridge component support of RAP.

12.35.4 RA Class Bridge Table

There is one RA Class Bridge Table per Bridge component. Each table row contains a set of parameters associated with an RA class supported by the Bridge, as detailed in Table 12-45.

Table 12-45—RA Class Bridge Table row elements

Name	Data type	Operations supported[a]	Conformance[b]	References
raClassId	integer	RW	B	51.3.2.1
raClassPriority	integer	RW	B	51.3.2.2
rtid	integer	RW	B	51.3.2.3
templatedDefinedData	octet string	RW	B	51.5.2.1.5

[a] R = read only access; RW = Read/Write access.
[b] B = required for Bridge or Bridge component support of RAP.

12.35.5 RA Class Port Table

There is one RA Class Port Table per Port of a Bridge component. Each table row contains a set of parameters associated with an RA class supported by the Bridge and configured in the RA Class Base Table (12.35.4), as detailed in Table 12-46.

Table 12-46—RA Class Port Table row elements

Name	Data type	Operations supported[a]	Conformance[b]	References
raClassId	integer	R	B	51.8.4.8 b)
domainBoundaryStatus	Boolean	R	B	51.8.4.8 c)
maxStreamFrameSize	integer	RW	B	51.8.4.8 d)
minStreamFrameSize	integer	RW	B	51.8.4.8 e)
maxBandwidth	integer	RW	B	51.8.4.8 f)
allocatedBandwidth	integer	R	B	51.8.4.8 g)
proposedMaxHopLatency	integer	RW	B	51.8.4.8 h)

[a] R = read only access; RW = Read/Write access.
[b] B = required for Bridge or Bridge component support of RAP.

12.35.6 RA Class Port Pair Table

There is one RA Class Port Pair Table per reception transmission Port pair of a Bridge component. Each table row contains a set of parameters associated with an RA supported by the Bridge and configured in the RA Class Base Table (12.35.4), as detailed in Table 12-47.

Table 12-47—RA Class Port Pair Table row elements

Name	Data type	Operations supported[a]	Conformance[b]	References
raClassId	integer	R	B	51.8.4.9 c)
maxHopLatency	integer	RW	B	51.8.4.9 d)

[a] R = read only access; RW = Read/Write access.
[b] B = required for Bridge or Bridge component support of RAP.

12.35.7 RAP Priority Regeneration Override Table

There is one RAP Priority Regeneration Override Table per Port of a Bridge component. Each table row contains a set of parameters for a received priority value that is associated with an RA class configured in the RA Class Base Table (12.35.4), as detailed in Table 12-48.

Table 12-48—RAP Priority Regeneration Override Table row elements

Name	Data type	Operations supported[a]	Conformance[b]	References
receivedPriority	integer [0..7]	R	B	6.9.4
regeneratedPriority	integer [0..7]	RW	B	6.9.4

[a] R = read only access; RW = Read/Write access.
[b] B = required for Bridge or Bridge component support of RAP.

12.35.8 RAP Redundancy Context Table

There is one RAP Redundancy Context Table per Bridge component. Each table row contains a set of parameters associated with a Redundancy Context (51.3.8.2) supported by the Bridge, as detained in Table 12-49.

Table 12-49—RAP Redundancy Context Table row elements

Name	Data type	Operations supported[a]	Conformance[b]	References
redundancyContextId	unsigned integer [0..4094]	RW	B	51.3.8.2, 51.8.4.10
vlanContextList	a sequence of unsigned integer [0..4094]	RW	B	51.3.8.2, 51.8.4.10

[a] R = read only access; RW = Read/Write access.
[b] B = required for Bridge or Bridge component support of RAP.

12.35.9 RAP Talker Announce Registration Port Table

There is one RAP Talker Announce Registration Port Table per Port of a Bridge component. Each table row in the table associated with a Port corresponds to a registration of the Talker Announce attribute (51.5.3) on that Port, and contains a set of parameters as detained in Table 12-50. Rows in the table are created, modified and deleted dynamically in the operation of resource allocation.

Table 12-50—RAP Talker Announce Registration Port Table row elements

Name	Data type	Operations supported[a]	Conformance[b]	References
streamId	octet string(size(8))	R	B	51.5.3.1
streamRank	unsigned integer [0..1]	R	B	51.5.3.2
accumulatedMaxLatency	unsigned integer	R	B	51.5.3.3
accumulatedMinLatency	unsigned integer	R	B	51.5.3.4
destinationMacAddress	MAC address	R	B	51.5.3.5.1
priority	unsigned integer [0..7]	R	B	51.5.3.5.2
vid	unsigned integer [0..4094]	R	B	51.5.3.5.3
tokenBucketTSpec	Boolean	R	B	12.35.9.1
tokenBucketTSpecValue	octet string	R	B	12.35.9.2
msrpTSpec	Boolean	R	B	12.35.9.3
msrpTSpecValue	octet string	R	B	12.35.9.4
redundancyControl	Boolean	R	B	12.35.9.5
redundancyControlValue	octet string	R	B	12.35.9.6
failureInfo	Boolean	R	B	12.35.9.7
failureInfoValue	octet string	R	B	12.35.9.8
orgDefinedInfo	Boolean	R	B	12.35.9.9
orgDefinedInfoValue	octet string	R	B	12.35.9.10

[a] R = read only access; RW = Read/Write access.
[b] B = required for Bridge or Bridge component support of RAP.

12.35.9.1 tokenBucketTSpec

The tokenBucketTSpec parameter returns a Boolean value that indicates whether a Token Bucket TSpec sub-TLV (51.5.3.6) is contained in the Talker Announce attribute (TRUE) or not (FALSE).

12.35.9.2 tokenBucketTSpecValue

When tokenBucketTSpec is TRUE, the tokenBucketTSpecValue parameter returns an octet string that is copied from the whole of the Value field of a Token Bucket TSpec sub-TLV (51.5.3.6) contained in the Talker Announce attribute.

When tokenBucketTSpec is FALSE, the tokenBucketTSpecValue parameter returns the empty octet string.

12.35.9.3 msrpTSpec

The msrpTSpec parameter returns a Boolean value that indicates whether a Token Bucket TSpec sub-TLV is contained in the Talker Announce attribute (TRUE) or not (FALSE).

12.35.9.4 msrpTSpecValue

When msrpTSpec is TRUE, the msrpTSpecValue parameter returns an octet string that is copied from the whole of the Value field of a MSRP TSpec sub-TLV (51.5.3.7) contained in the Talker Announce attribute.

When msrpTSpec is FALSE, the msrpTSpecValue parameter returns the empty octet string.

12.35.9.5 redundancyControl

The redundancyControl parameter returns a Boolean value that indicates whether a Redundancy Control sub-TLV (51.5.3.8) is contained in the Talker Announce attribute (TRUE) or not (FALSE).

12.35.9.6 redundancyControlValue

When redundancyControl is TRUE, the redundancyControlValue parameter returns an octet string that is copied from the whole of the Value field of a Redundancy Control sub-TLV (51.5.3.8) contained in the Talker Announce attribute.

When redundancyControl is FALSE, the redundancyControlValue parameter returns the empty octet string.

12.35.9.7 failureInfo

The failureInfo parameter returns a Boolean value that indicates whether a Failure Information sub-TLV (51.5.3.10) is contained in the Talker Announce attribute (TRUE) or not (FALSE).

12.35.9.8 failureInfoValue

When failureInfo is TRUE, the failureInfoValue parameter returns an octet string that is copied from the whole of the Value field of a Failure Information sub-TLV (51.5.3.10) contained in the Talker Announce attribute.

When failureInfo is FALSE, the failureInfoValue parameter returns the empty octet string.

12.35.9.9 orgDefinedInfo

The orgDefinedInfo parameter returns a Boolean value that indicates whether an Organizationally Defined sub-TLV (51.5.3.11) is contained in the Talker Announce attribute (TRUE) or not (FALSE).

12.35.9.10 orgDefinedInfoValue

When orgDefinedInfo is TRUE, the orgDefinedInfoValue parameter returns an octet string that is copied from the whole of the Value field of an Organizationally Defined sub-TLV (51.5.3.11) contained in the Talker Announce attribute.

When orgDefinedInfo is FALSE, the orgDefinedInfoValue parameter returns the empty octet string.

12.35.10 RAP Talker Announce Declaration Port Table

There is one RAP Talker Announce Declaration Port Table per Port of a Bridge component. Each table row in the Table associated with a Port corresponds to a declaration of the Talker Announce attribute (51.5.3) on that Port, and contains the same set of parameters as specified in Table 12-50. Rows in the table are created, modified and deleted dynamically in the operation of resource allocation by RAP.

12.35.11 RAP Listener Attach Registration Port Table

There is one RAP Listener Attach Registration Port Table per Port of a Bridge component. Each table row in the Table associated with a Port corresponds to a registration of the Listener Attach attribute (51.5.4) on that Port and contains a set of parameters as detained in Table 12-51. Rows in the table are created, modified and deleted dynamically in the operation of resource allocation by RAP.

Table 12-51—RAP Listener Attach Registration Port Table row elements

Name	Data type	Operations supported[a]	Conformance[b]	References
streamId	octet string(size(8))	R	B	51.5.4.1
vid	unsigned integer [0..4094]	R	B	51.5.4.2
listenerAttachStatus	Enumerated	R	B	51.5.4.3
vlanContextStatus	Boolean	R	B	12.35.11.1
vlanContexStatusValue	octet string	R	B	12.35.11.2

[a] R = read only access; RW = Read/Write access.
[b] B = required for Bridge or Bridge component support of RAP; e = optional for end station support of RAP.

12.35.11.1 vlanContextStatus

The vlanContextStatus parameter returns a Boolean value that indicates whether a VLAN Context Status sub-TLV (51.5.4.4) is contained in the Listener Attach attribute (TRUE) or not (FALSE).

12.35.11.2 vlanContextStatusValue

When vlanContextStatus is TRUE, the vlanContextStatusValue parameter returns an octet string that is copied from the whole of the Value field of a a VLAN Context Status sub-TLV (51.5.4.4) contained in the Listener Attach attribute.

12.35.12 RAP Listener Attach Declaration Port Table

There is one RAP Listener Attach Declaration Port Table per Port of a Bridge component. Each table row in the Table associated with a Port corresponds to a declaration of the Listener Attach attribute (51.5.4) on that Port and contains the same set of parameters as specified in Table 12-51. Rows in the table are created, modified and deleted dynamically in the operation of resource allocation by RAP.

17. Management Information Base (MIB)

17.2 Structure of the MIB

Insert the following subclause at the end of 17.2 using the next available subclause number:

17.2.x Structure of the IEEE8021-ECP-MIB module

The IEEE8021-ECP-MIB module defines managed objects for management of Edge Control Protocol (12.27.1).

Table 17-52—IEEE8021-ECP-MIB structure

IEEE8021-ECP-MIB table/object	Reference
ieee8021BridgeEcpTable	12.27.1
ieee8021BridgeEcpIfIndex	ecpPortNumber, 12.4.2
ieee8021BridgeEcpAdminAckTimerInitExp	ecpProposedRTE, Clause 43
ieee8021BridgeEcpOperAckTimerInitExp	ecpOperAckTimerInit, 43.3.6.2
ieee8021BridgeEcpAdminMaxRetries	ecpProposedR, Clause 43
ieee8021BridgeEcpOperMaxRetries	ecpOperMaxRetries, 43.3.7.4
ieee8021BridgeEcpTxFrameCount	ecpTxFrameCount, Clause 43
ieee8021BridgeEcpTxRetryCount	ecpTxRetryCount, Clause 43
ieee8021BridgeEcpTxFailures	ecpTxFailures, Clause 43
ieee8021BridgeEcpRxFrameCount	ecpRxFrameCount, Clause 43
ieee8021BridgeEcpDestMacAddress	ecpDestinationAddress, Clause 43

17.3 MIB module relationships

Insert the following subclause at the end of 17.3 using the next available subclause number:

17.3.x Relationship of IEEE8021-ECP-MIB to other MIB modules

<< Editor's note: text contribution is required for this subclause. >>

17.4 MIB security considerations

Insert the following subclause at the end of 17.4 using the next available subclause number:

17.4.x Security considerations of the IEEE8021-ECP-MIB

<< Editor's note: text contribution is required for this subclause. >>

17.7 MIB Modules

Insert the following subclause at the end of 17.7 using the next available subclause number:

17.7.x Definitions for the IEEE8021-ECP-MIB module

```
IEEE8021-ECP-MIB DEFINITIONS ::= BEGIN

-- ==============================================================
-- IEEE 802.1Q MIB Edge Control Protocol
-- ==============================================================

IMPORTS
    MODULE-IDENTITY, OBJECT-TYPE,
    Counter32, Unsigned32
        FROM SNMPv2-SMI
    MacAddress
        FROM SNMPv2-TC
    ieee802dot1mibs
        FROM IEEE8021-TC-MIB
    InterfaceIndexOrZero
        FROM IF-MIB
    MODULE-COMPLIANCE, OBJECT-GROUP
        FROM SNMPv2-CONF;

ieee8021BridgeEcpMib MODULE-IDENTITY

    LAST-UPDATED "202209130824Z" -- September 13, 2022
    ORGANIZATION "IEEE 802.1 Working Group"
    CONTACT-INFO
        " WG-URL:   http://www.ieee802.org/1/
          WG-EMail: stds-802-1-l@ieee.org
          Contact:  IEEE 802.1 Working Group Chair
          Postal:   C/O IEEE 802.1 Working Group
                    IEEE Standards Association
                    445 Hoes Lane
                    Piscataway, NJ 08854
                    USA
          E-mail:   stds-802-1-chairs@ieee.org"
    DESCRIPTION
        "The ECP MIB module for managing devices that support
        the Edge Control Protocol.

        Unless otherwise indicated, the references in this MIB
        module are to IEEE Std 802.1Q-2022.

        Copyright (C) IEEE (2022).

        This version of this MIB module is part of IEEE Std 802.1Q;
        see that standard for full legal notices."

    REVISION "202209130824Z" -- September 13, 2022
    DESCRIPTION
        "Initial version not yet published."

    ::= { ieee802dot1mibs 999 }  -- MUST BE SET TO CORRECT VALUE

-- ==============================================================
-- subtrees in the ECP MIB
-- ==============================================================

ieee8021BridgeEcpObjects
    OBJECT IDENTIFIER ::= { ieee8021BridgeEcpMib 1 }
```

```
ieee8021BridgeEcpConformance
    OBJECT IDENTIFIER ::= { ieee8021BridgeEcpMib 2 }

-- ==========================================
-- Edge Control Protocol Table
-- ==========================================

ieee8021BridgeEcpTable OBJECT-TYPE
    SYNTAX SEQUENCE OF Ieee8021BridgeEcpEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
    "A table that contains configuration information for
    the Edge Control Protocol (ECP)."
    REFERENCE "12.26.4.2"
    ::= { ieee8021BridgeEcpObjects 1 }

ieee8021BridgeEcpEntry OBJECT-TYPE
    SYNTAX Ieee8021BridgeEcpEntry
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION
    "A list of objects containing information for the Edge Control
    Protocol (ECP)."
    INDEX { ieee8021BridgeEcpIfIndex }
    ::= { ieee8021BridgeEcpTable 1 }

    Ieee8021BridgeEcpEntry ::=
        SEQUENCE {
        ieee8021BridgeEcpIfIndex    InterfaceIndexOrZero,
        ieee8021BridgeEcpAdminAckTimerInitExp Unsigned32,
        ieee8021BridgeEcpOperAckTimerInitExp  Unsigned32,
        ieee8021BridgeEcpAdminMaxRetries      Unsigned32,
        ieee8021BridgeEcpOperMaxRetries       Unsigned32,
        ieee8021BridgeEcpTxFrameCount         Counter32,
        ieee8021BridgeEcpTxRetryCount         Counter32,
        ieee8021BridgeEcpTxFailures           Counter32,
        ieee8021BridgeEcpRxFrameCount         Counter32,
        ieee8021BridgeEcpDestMacAddress       MacAddress
        }

ieee8021BridgeEcpIfIndex OBJECT-TYPE
    SYNTAX InterfaceIndexOrZero
    MAX-ACCESS not-accessible
    STATUS current
    DESCRIPTION "Identifies the ifIndex of the interface to
                which this table entry applies. A value of
                0 can be used if the interface table is not
                supported and the system has only one
                interface."
    ::= { ieee8021BridgeEcpEntry 1 }

ieee8021BridgeEcpAdminAckTimerInitExp OBJECT-TYPE
    SYNTAX Unsigned32 (0..31)
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION "The operator-desired value used to initialize ackTimer
        (43.3.6.1).
        A valuie x for ieee8021BridgeEcpAdminAckTimerInitExp indicates a
        an ackTimer of (10 microseconds) * (2**x).  Thus, a value of 0
        indicates 10 microseconds, and a value 0f 3 indicates 80."
    DEFVAL { 14 }
    ::= { ieee8021BridgeEcpEntry 2 }

ieee8021BridgeEcpOperAckTimerInitExp OBJECT-TYPE
```

```
    SYNTAX Unsigned32 (0..31)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "The initial value used to initialize ackTimer
        (43.3.6.1).  This is the same value reported in
        ieee8021BridgeEvbEcpOperAckTimerInitExp in IEEE8021-EVB-MIB.
        It is the maximum of the EVB TLV's RTE values (D.2.12.6) and
        ieee8021BridgeEcpAdminAckTimerInitExp.
        A valuie x for ieee8021BridgeEcpOperAckTimerInitExp indicates a
        an ackTimer of (10 microseconds) * (2**x).  Thus, a value of 0
        indicates 10 microseconds, and a value 0f 3 indicates 80."
    ::= { ieee8021BridgeEcpEntry 3 }

ieee8021BridgeEcpAdminMaxRetries OBJECT-TYPE
    SYNTAX Unsigned32 (0..7)
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION "This operator-desired value for the maximum number
                of times that the ECP transmit state machine will
                retry a transmission if no ACK is received."
    DEFVAL { 3 }
    ::= { ieee8021BridgeEcpEntry 4 }

ieee8021BridgeEcpOperMaxRetries OBJECT-TYPE
    SYNTAX Unsigned32 (0..7)
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "This integer variable reports the maximum number
                of times that the ECP transmit state machine will
                retry a transmission if no ACK is received.  It is
                the maximum of the EVB TLV's R values (D.2.12.5)
                and ieee8021BridgeEcpAdminMaxRetries."
    ::= { ieee8021BridgeEcpEntry 5 }

ieee8021BridgeEcpTxFrameCount OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "The EcpTxFrameCount is the number of ECP frames
                transmitted since ECP was instantiated."
    ::= { ieee8021BridgeEcpEntry 6 }

ieee8021BridgeEcpTxRetryCount OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "The EcpTxRetryCount is the number of times
                ECP re-tried transmission since ECP was
                instantiated."
    ::= { ieee8021BridgeEcpEntry 7 }

ieee8021BridgeEcpTxFailures OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "The EcpTxFailures is the number of times ECP
                failed to successfully deliver a frame since ECP
                was instantiated."
    ::= { ieee8021BridgeEcpEntry 8 }

ieee8021BridgeEcpRxFrameCount OBJECT-TYPE
    SYNTAX Counter32
    MAX-ACCESS read-only
    STATUS current
    DESCRIPTION "The EcpRxFrameCount is the number
```

```
                of frames received since ECP was instantiated."
    ::= { ieee8021BridgeEcpEntry 9 }

ieee8021BridgeEcpDestMacAddress OBJECT-TYPE
    SYNTAX MacAddress
    MAX-ACCESS read-write
    STATUS current
    DESCRIPTION "The destination MAC address to be used for
                ECP frames. Default value is the Nearest Customer
                Bridge address from Table 8-1."
    DEFVAL { "0180c20000" }
    ::= { ieee8021BridgeEcpEntry 10 }

-- ==============================================================
-- Conformance Information
-- ==============================================================

ieee8021BridgeEcpGroups
    OBJECT IDENTIFIER ::= { ieee8021BridgeEcpConformance 1 }
ieee8021BridgeEcpCompliances
    OBJECT IDENTIFIER ::= { ieee8021BridgeEcpConformance 2 }

-- ==============================================================
-- Units of conformance
-- ==============================================================

ieee8021BridgeEcpGroup OBJECT-GROUP

    OBJECTS {
        ieee8021BridgeEcpAdminAckTimerInitExp,
        ieee8021BridgeEcpOperAckTimerInitExp,
        ieee8021BridgeEcpAdminMaxRetries,
        ieee8021BridgeEcpOperMaxRetries,
        ieee8021BridgeEcpTxFrameCount,
        ieee8021BridgeEcpTxRetryCount,
        ieee8021BridgeEcpTxFailures,
        ieee8021BridgeEcpRxFrameCount,
        ieee8021BridgeEcpDestMacAddress
    }

    STATUS current
    DESCRIPTION
        "The collection of objects used to represent the ECP
        management objects."
    ::= { ieee8021BridgeEcpGroups 1 }

ieee8021BridgeEcpCompliance MODULE-COMPLIANCE
    STATUS current
    DESCRIPTION
    "The compliance statement for devices supporting
    the Edge Control Protocol."

MODULE IF-MIB -- The interfaces MIB, RFC 2863
    MANDATORY-GROUPS {
        ifGeneralInformationGroup
    }

MODULE
    MANDATORY-GROUPS {
        ieee8021BridgeEcpGroup
    }

::= { ieee8021BridgeEcpCompliances 1 }

END
```

43. Edge Control Protocol (ECP)

<< Editor's note: All the changes in Clause 43 are made exclusively for maintenance request #0248, as agreed in the comment resolution #60 for P802.1Qdd/D0.6. >>

Change the text in 43.3.6.1 as follows:

43.3.6.1 ackTimer

The ackTimer is used to determine how long the transmit state machine will wait for an acknowledgment PDU to be received before it either retries a transmission or aborts a transmission due to too many retries. This timer is initialized using the value of ackTimerInit~~ determined as stated in D.2.12.6~~.

Insert the following subclause after 43.3.6.1:

43.3.6.2 ackTimerInit

This parameter is used to initialize the ackTimer variable. It is set either by a three way negotiation between the values of the ecpProposedRTE object of the ECP management database and the local and remote EVB LLDP TLV's RTE value (D.2.12.6), or the ecpProposedRTE object of the ECP management database if EVB is not present.

Change the text in 43.3.7.4 as follows:

43.3.7.4 maxRetries

This integer variable defines the maximum number of times that the ECP transmit state machine will retry a transmission if no ACK is received. The default value of maxRetries is 3~~; this variable can be changed by management as documented in 12.26.2. The value is derived from ecpOperMaxRetries (Table 12-31)~~. The ECP variable maxRetries is set by a three way negotiation between the values of the ecpProposedR object of the ECP management database and the local and remote EVB LLDP TLV's R value (D.2.12.6), or the ecpProposedR object of the ECP management database if EVB is not present.

Insert the following clause after Clause 50:

51. Resource Allocation Protocol (RAP)

This clause specifies the Resource Allocation Protocol (RAP).

51.1 RAP overview

<< Editor's note: This subclause is intended to provide a brief introduction to RAP. >>

51.2 Conventions

This subclause defines the conventions used in the specification of RAP.

51.2.1 State machine diagrams

The state machine diagrams defined in 51.7.3 for RAP Participant and in 51.8.2 for RAP Propagator use the conventions defined in Annex E with the following extensions:

a) All transitions and operations are atomic and finish without any progress of time, as soon as the associated conditional expression(s) of transition(s) is/are satisfied. The sole reason for steady states is that none of the conditional expressions of any outgoing transition of a particular state is satisfied.
b) Service primitives can occur in the conditional expressions of transitions. Invocation of such service primitives immediately leads to activation of a transition, provided that:
   1) All other sub-expressions are satisfied, if such expressions exist, and
   2) The state machine resides in the associated state prior to invocation.
c) Service primitives that cannot be processed immediately are queued in their order of invocation for subsequent processing (i.e., no service primitive is lost).
d) In case the conditional expressions of more than one transition are satisfied simultaneously, the taken transition is arbitrarily chosen.

51.2.2 Pseudo-code

A C++ like pseudo-code is used where applicable in defining actions executed on entry to a state in state machine diagrams or when a procedure is invoked. The emphasis is on simplicity, clarity of specification and unambiguous description of the externally visible behavior. Efficiency (speed, memory usage, etc.) is left to the implementation (in software/firmware, hardware, combinations of the aforesaid, etc.).

51.2.3 Naming of parameters and variables

Table 51-1 shows the conventions used in naming RAP variables and parameters.

51.2.4 Array variables

RAP operates on a set of variables that have different scopes, such as per-Port, per-Port per-RA class, or per-Port per-Stream. Array variables are used to group the set of RAP variables that share the same scope together under a single container for convenience of reference in the pseudo-code.

For example, the following array notation

XYZ[<A, B>]

Table 51-1—Naming conventions for RAP parameters and variables

Format	Description	Applied to	Example
XxxXxx	upper camel case	parameters in Value fields of RAP TLVs/sub-TLVs	StreamId (51.5.3.1)
xxXxx	lower camel case with no prefix	variables local to a RAP entity	helloTime (51.7.4.4)
tXxx	lower camel case with prefix ‘t’	variables local to a procedure or a state	tAttrReg (51.7.5.11)
rXxx	lower camel case with prefix ‘r’	parameters of request primitives	DECLARE_ATTRIBUTE .request(rPortRef, rAttr) (51.7.2.1)
iXxx	lower camel case with prefix ‘i’	parameters of indication primitives	ATTRIBUTE_REGISTRATION .indication(iPortRef, iAttr) (51.7.2.3)
pXxx	lower camel case with prefix ‘p’	input parameters of procedures	checkHello(pHelloData) (51.7.5.3)

addresses an element in an array variable XYZ that is associated with a key that is expressed as <A, B>. It returns a reference to the addressed element if that element exists in the array, and a NULL value if that element does not exist. The asterisk character (‘*’) can be used as a wildcard for one or more elements of a key to address the subset of entries that match the remaining non-wildcarded key elements.

NOTE—The arrays used in this clause are analogous to associative arrays, a data type commonly used to store a collection of (key, value) pairs. The use of this data type only represents a way of organizing data inside a system. Thus, there is no explicit requirement that this be used in implementations.

Two functions, create and delete, are defined for use in pseudo-code, as follows:

a) Invocation of **`create`** `XYZ[<key>]` adds an element with the given key to the array XYZ and returns a reference to the added element.
b) Invocation of **`delete`** `XYZ[<key>]` removes the element (elements) that matches the given key from the array XYZ.

51.3 Model of operation

51.3.1 RAP architecture

RAP supports all three classes of systems specified in IEEE Std 802.1CS Link-local Registration Protocol (LRP): Native systems, Controlled systems and Proxy systems. For the purposes of this standard, these systems can be further classified as shown in Table 51-2.

A RAP instance is an LRP application that implements the RAP functionality. There are two types of RAP instances: RAP End instances for end stations and RAP Relay instances for Bridges.

There is a single RAP End instance or RAP Relay Instance residing in each RAP Native end station or RAP Native Bridge, respectively, executing the operation of RAP locally. For a RAP Controlled station, the operation of RAP is remotely executed at a given RAP Proxy. Conceptually, there is a single RAP End

Table 51-2—RAP System classes

		RAP system classes	
LRP system classes	Native system	RAP Native station	RAP Native end station
			RAP Native Bridge
	Controlled system	RAP Controlled station	RAP Controlled end station
			RAP Controlled Bridge
	Proxy system	RAP Proxy	RAP End Station Proxy
			RAP Bridge Proxy

instance or RAP Relay instance offloaded by each RAP Controlled end station or RAP Controlled Bridge, respectively, onto a corresponding RAP Proxy.

In a RAP End instance or RAP Relay Instance, a RAP Participant (51.7) exists for each physical Port that is connected a point-to-point medium, and communicates with the underlying LRP via a LRP-DS Service Interface (LSI, as specified in IEEE Std 802.1CS).

Each RAP End instance has a single RAP End Point (51.6), which communicates with the RAP Participant via a RAP Participant Service Interface (RPSI, 51.7.2) and provides a RAP Endpoint Service Interface (RESI, 51.6.2) to the higher layer entities.

Each RAP Relay instance has a single RAP Propagator (51.8), which communicates with the per-Port RAP Participants via the RPSI.

Figure 51-1 shows the architecture of RAP in a single Port end station and a two-Port Bridge, in the case of using the LRP-DT ECP mechanism.

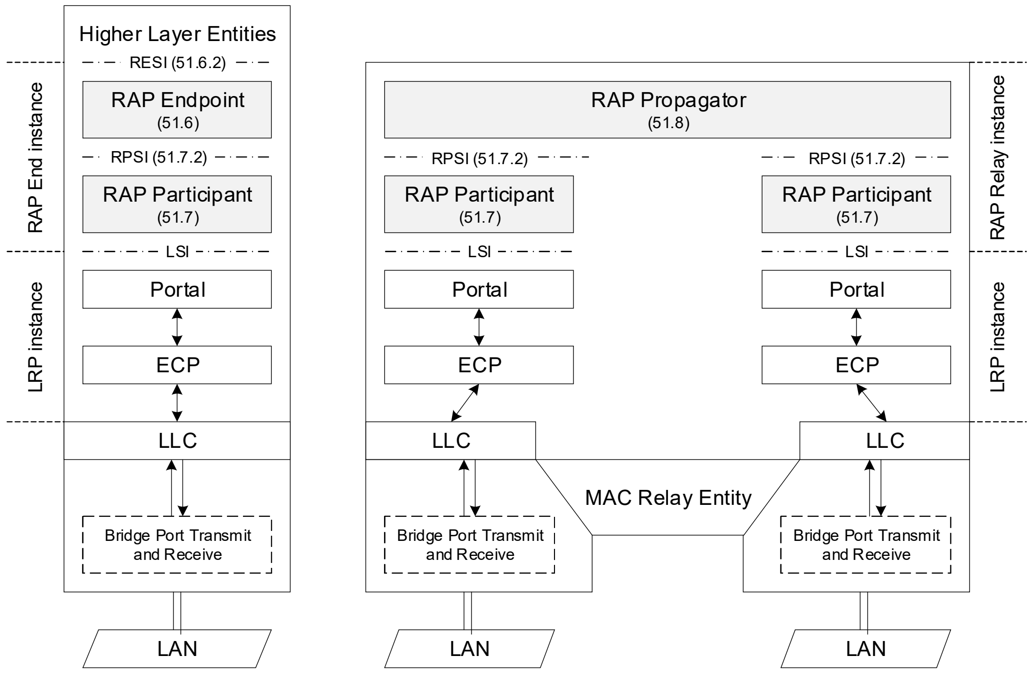


Figure 51-1—RAP architecture using ECP

Figure 51-2shows the architecture of RAP in a single Port end station and a two-Port Bridge, in the case of using the LRP-DT TCP mechanism.

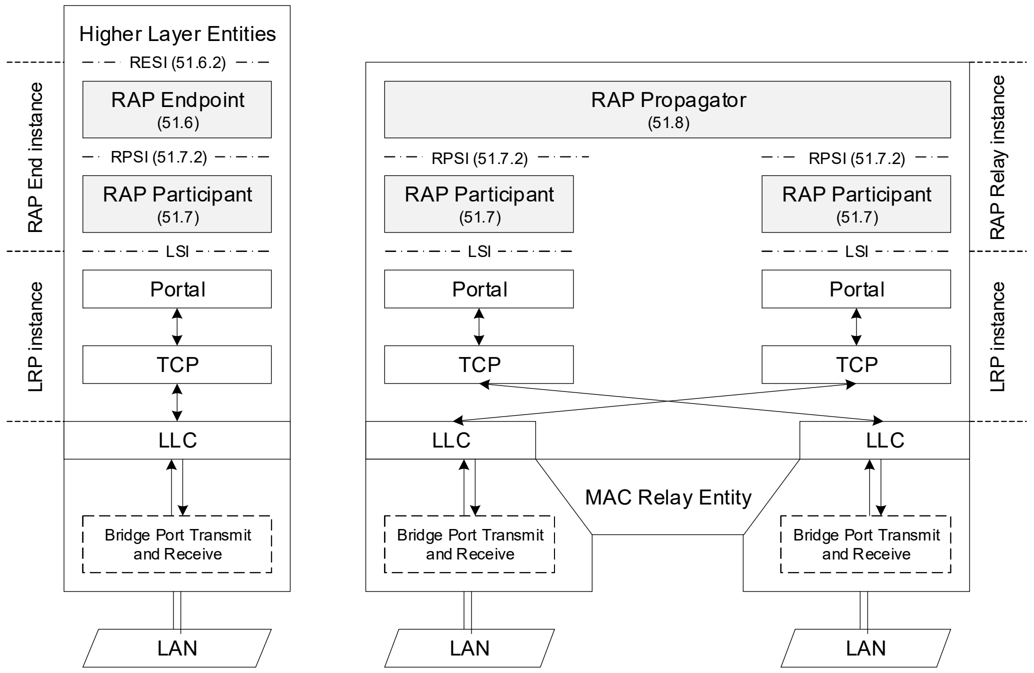


Figure 51-2—RAP architecture using TCP

51.3.2 RA class

An RA class represents a traffic class or a set of traffic classes in a station that uses a given transmission selection algorithm, in conjunction with other mechanisms (e.g., PSFP defined in 8.6.5.1, the traffic scheduling mechanisms defined in 8.6.8.4) if needed, and a resource reservation method, to provide a bounded latency and zero congestion loss for time-sensitive streams.

NOTE—An example of using more than one traffic class for a single RA class is an implementation of the cyclic queuing and forwarding shaper (CQF) using the approach as described in Annex T.2. In that case, two transmission queues operate in an coordinated manner and offer the same bounded latency to frames transmitted from either of the two queues.

51.3.2.1 RA class ID

The RA class ID is an integer in the range of 0 through 255 that uniquely identifies an RA class supported by an end station or Bridge.

NOTE—RA class ID is a numeric representation of the RA classes supported by a station. Among a set of RA classes on a given Bridge Port, one RA class that is assigned a numerically higher RA class ID than another RA class, is not necessarily mapped to a numerically higher traffic class than that RA class.

51.3.2.2 RA class priority

Each RA class is associated with a distinct priority value in the range 0 through 7, termed RA class priority. The RA class priority indicates the received priority of the frames which are to be mapped to the traffic class(es) with which that RA class is associated.

NOTE—Mapping of RA class priorities to traffic classes is a matter of port local configuration and can be managed using the Traffic Class Table (12.6.3). This standard does not specify default priority to traffic class mapping for a system that supports RA classes.

Since there are eight values available for the RA class priority, a station can support up to seven RA classes, in order to ensure that there is at least one traffic class that can support non-stream traffic that is not subject to resource allocation, such as “best effort” traffic.

51.3.2.3 RA class template and RTID

An RA class template describes a specific method for making up an RA class, including the following:

— The shaping or scheduling mechanisms employed.
— The algorithms for computing latency bounds and resources.
— The type of TSpec used to describe stream traffic characteristics.
— The encoding of the data as the value of the RaClassTemplateDefinedData field (51.5.2.1.5).

An RA class template is identified by an RA Class Template Identifier (RTID), which encodes a 3-octet OUI or CID value identifying the organization that defines that template, followed by a 1-octet index allocated by that organization. The RA class templates currently defined by IEEE 802.1 are listed in Table 51-3.

51.3.2.4 RA class domains and domain boundary ports

An RA class domain is a connected subset of stations (end stations and/or Bridges) that support a common RA class (i.e. the same RA class ID, 51.3.2.1) and associate the same priority value with that RA class (i.e. the same RA class priority, 51.3.2.2). Within an RA class domain, any traffic associated with the RA class priority of that domain is treated as stream traffic and subject to resource allocation by RAP.

Table 51-3—IEEE 802.1 RA class templates

RTID	Transmission Selection Algorithm	TSpec type
00-80-C2-00	Strict Priority (8.6.8.1)	Token Bucket TSpec (51.5.3.6)
00-80-C2-01	ATS Transmission Selection (8.6.8.5)	Token Bucket TSpec (51.5.3.6)

An RA class domain boundary port is a Port of a Bridge that is part of a given RA class domain and is connected via a LAN to an adjacent station that is not part of that RA class domain. On such a Port, any data frame received with a priority identical to the RA class priority of that RA class domain is regarded non-stream traffic.

To prevent non-stream traffic from disrupting stream traffic within an RA class domain, each RA class domain boundary port is subject to priority regeneration (6.9.4) according to an *RAP Priority Regeneration Override Table (12.35.7)*, such that non-stream traffic entering from outside the RA class domain is forwarded within that RA class domain using another priority that is not associated with any RA class.

The detection of RA class domains including the determination of the location of RA class domain boundary ports is controlled by the operation of RA Advertisement (51.3.3). Figure 51-3 gives an example of multiple RA Class domains established in a single network, where three domains for RA class ID 1 and three for RA class ID 2 are separately depicted on the left-hand and the right-hand side of the figure, respectively.

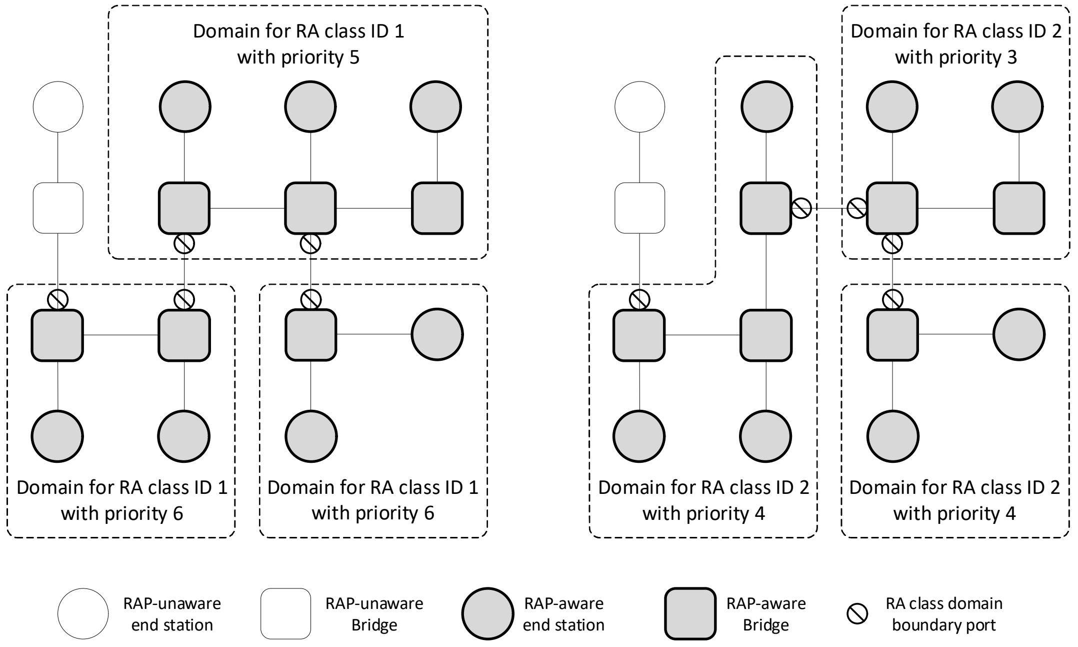

Domain for RA class ID 1 with priority 5
Domain for RA class ID 2 with priority 3
Domain for RA class ID 1 with priority 6
Domain for RA class ID 1 with priority 6
Domain for RA class ID 2 with priority 4
Domain for RA class ID 2 with priority 4
RAP-unaware end station
RAP-unaware Bridge
RAP-aware end station
RAP-aware Bridge
RA class domain boundary port

Figure 51-3—Examples of RA class domains and domain boundary ports

51.3.3 RA Advertisement

RA Advertisement refers to declarations of the RA attribute (51.5.2) by a station to each neighboring station.

The purposes of RA Advertisement are as follows:

- Advertisement of RA class domain configuration for detection of RA class domains and determination of the location of RA class domain boundary ports, as described in 51.3.2.4.
- End station leaning of the network settings e.g. RA classes, redundant trees. To allow adaptive participation in resource allocation, an end station can support the ability to adjust its local parameters according to the information received from its adjacent Bridge and then declare its own RA attribute with the adjusted parameters.
- Advertisement of RA class template configuration. In the process of reserving a stream in a given RA class, the RA class template used by that RA class and corresponding parameter configuration in an upstream station is necessary information used by each downstream station in performing actions such as calculation of the worst-case latency of streams transmitted in that RA class from that upstream station.

The following terminology relating to RA Advertisement is used:

a) **RA declaration**: A declaration of the RA attribute on an end-station or Bridge Port.
b) **RA registration**: A registration of the RA attribute on an end-station or Bridge Port.

An RA registration received from a neighboring station on a Bridge Port is not propagated by the Bridge to declare that registered RA attribute to another neighboring on any of other Ports.

51.3.4 Talker Announcement

Talker Announcement refers to the process of signaling the Talker Announce attribute (51.5.3) initially declared by the Talker of a stream across a Bridged network to potential Listeners. Each Talker Announcement is identified by a StreamId (51.5.3.1) as contained in the Talker Announce attribute.

The purposes of Talker Announcement are as follows:

- Indication of a Talker’s intention of supplying a stream.
- Advertisement of the stream’s characteristics. The Talker Announce attribute contains the information needed by the network to identify the stream and determine the required resources on each Bridge along the stream path(s).
- Gathering of QoS information from the Bridges along each path such as the accumulated latency and reporting of gathered information to Listeners.
- Signaling of status information along each potential path about whether a reservation can be made, and if not the information about the cause and the location of the failure.

The following terminology relating to Talker Announcement is used:

a) **Talker Announce declaration:** A declaration of the Talker Announce attribute on a Port.
b) **Talker Announce registration:** A registration of the Talker Announce attribute on a Port.
c) **Talker Announce deregistration:** A deregistration of the Talker Announce attribute on a Port.
d) **Talker Announce propagation:** Propagation of a Talker Announce registration from a Bridge Port to one or more other Bridge Ports.
e) **Talker Announce merge:** Merge of multiple Talker Announce registrations being propagated to a Bridge Port to result in a joint Talker Announce declaration on that Port.
f) **Originating Talker Announce registration:** A Talker Announce registration from which a given Talker Announce declaration results from is termed an originating Talker Announce registration of that Talker Announce declaration. A Talker Announce declaration can have more than one originating Talker Announce registration in the case of Talker Announce merge.

51.3.4.1 Talker Announce propagation

A set of rules applied to Talker Announce propagation are described in the following subclauses.

51.3.4.1.1 VLAN Context

Talker Announce propagation operates within a VLAN Context, or a set of VLAN Contexts, each identified by a VID, termed VLAN Context ID. A VLAN Context in a Bridge for a given VID corresponds to a VLAN topology and is formed by the set of Ports that includes each Bridge Port for which the following are true:

1) The Port is in the Forwarding state and part of the active topology that supports that VID.
2) The Port is a member of the member set (8.8.10) for that VID.

Depending on the number of VLAN Contexts used, a Talker Announcement can be either of following:

a) **Single-Context Talker Announcement:** A Talker Announcement propagated in a single VLAN Context and used in resource allocation for transmission of a stream along a single path from the Talker to each Listener.
b) **Multi-Context Talker Announcement:** A Talker Announcement propagated in more than VLAN Context and used in resource allocation for transmission of a stream with seamless redundancy along multiple paths from the Talker to each Listener.

The VLAN Context rule is used in the Talker Announce propagation to determine a set of potential Ports to which a Talker Announce registration is to be propagated according to the VLAN Context(s) indicated in the Talker Announce registration. The encoding of the Talker Announce attribute for the VLAN Context information is specified in item a) in 51.5.3.5.3 for Single-Context Talker Announcement and in 51.5.3.9 for Multi-Context Talker Announcement.

51.3.4.1.2 Stream DA Pruning

In addition to VLAN Context, Stream DA Pruning applies an additional constraint to Talker Announce propagation by reference to the status of MAC Address Registrations (8.8.4). Stream DA Pruning can be enabled or disabled on a per-Bridge Port basis.

When Stream DA Pruning is enabled on a Port, a Talker Announce registration is not be propagated to that Port, if the DestinationMacAddress (51.5.3.5.1) value contained in the Talker Announce attribute is not found in the MAC Address Registration Entries for that Port.

51.3.4.1.3 Ingress Blocking

Ingress Blocking prohibits propagation of a Talker Announce registration on a Bridge Port, if all of the following conditions are met:

a) Ingress Filtering (8.6.2) is enabled on the Port.
b) The Port is not in the member set (8.8.10) associated with the VID (51.5.3.5.3) contained in the registered Talker Announce attribute.

NOTE—The reason for not propagating Talker Announce registrations in the above mentioned cases is to ensure that a stream is not reserved if any data frame of that stream would be discarded due to the operation of Ingress Filtering.

51.3.4.2 Talker Announce status

The Talker Announce status of a Talker Announce declaration made on a given Port is associated with an ordinary stream in the case of a Single-Context Talker Announcement, or a Compound or a Member Stream in the case of a Multi-Context Talker Announcement, to be transmitted on that Port, indicating whether the stream could be reserved from the Talker to that Port along a single path or one or more paths, respectively, in the VLAN Context(s) indicted in the Talker Announce declaration, as follows:

a) **Announce Success:** The stream could be reserved on the Port, as it has not encounter any problems on a single path from the Talker to that Port, if the stream is a ordinary stream, or on at least one path from the Talker to that Port, if the stream is a Compound Stream or Member Stream. The Announce Success status is represented by the absence of a Failure Information sub-TLV (51.5.3.10) in the Value field of the Talker Announce attribute TLV (51.5.3).
b) **Announce Fail:** The stream could not be reserved on the Port, as it has encounter problems on each path from the Talker to that Port. The Announce Fail status is represented by the presence of a Failure Information sub-TLV (51.5.3.10) in the Value field Talker Announce attribute TLV (51.5.3).

51.3.4.3 Talker Announce merge

Talker Announce merge only occurs in Multi-Context Talker Announcements in resource allocation for seamless redundancy (51.3.8) and is associated with the FRER operation stream merging in FRER-capable Bridges, as described in item a) of 51.3.8.5.

Talker Announce merge is performed in a FRER-capable Bridge among a set of Multi-Context Talker Announce registrations that are propagated to the same Port and contain the same StreamId (51.5.3.1) value, to result in a joint Multi-Context Talker Announce declaration on that Port for that StreamId and containing a merged set of VLAN Contexts comprising each one used in the propagation of those Talker Announce registrations.

The operation of Talker Announce merge is specified in the processTaDecMerge() procedure (51.8.5.8).

51.3.5 Listener Attachment

Listener Attachment refers to the process of signaling the Listener Attach attribute (51.5.4) initially declared by one or more Listeners desiring to receive a stream across a Bridge network to the stream’s Talker. Each Listener Attachment is identified by a StreamId (51.5.4.1) as contained in the Listener Attach attribute.

The purposes of Listener Attachment are as follows:

- Indication of the intention of one or more Listeners of receiving a stream.
- Triggering of resource allocations (or deallocation) on each Bridge along potential stream path(s).
- Signaling of reservation status hop-by-hop in an upstream direction from each participating Listener back to the Talker.

The following terminology relating to Listener Attachment is used:

a) **Listener Attach declaration**: A declaration of the Listener Attach attribute on a Port.
b) **Listener Attach registration**: A registration of the Listener Attach attribute on a Port.
c) **Listener Attach deregistration**: A deregistration of the Listener Attach attribute on a Port.
d) **Listener Attach propagation**: Propagation of a Listener Attach registration from a Bridge Port to one or more other Bridge Ports.
e) **Listener Attach merge**: Merge of more multiple Listener Attach registrations propagated to a Bridge Port to result in a joint Listener Attach declaration on that Port.
f) **Associated Talker Announce declaration**: A Talker Announce declaration with which a given Listener Attach registration is associated is termed the associated Talker Announce declaration of that Listener Attach registration.
g) **Originating Listener Attach registration**: A Listener Attach registration from which a given Listener Attach declaration results is termed an originating Listener Attach registration of that

Listener Attach declaration. A Listener Attach declaration can have more than one originating Listener Attach registration in the case of Listener Attach merge.

h) **Associated Talker Announce registration**: A Talker Announce registration with which a given Listener Attach declaration is associated is termed the associated Talker Announce registration of that Listener Attach declaration.

51.3.5.1 Association between Talker Announcement and Listener Attachment

A Listener Attachment is associated with a Talker Announcement that carries the same StreamId as that of the Listener Attachment. Depending on the type of the associated Taker Announcement, a Listener Attachment can be either of the following:

a) **Single-Context Listener Attachment:** A Listener Attachment that is associated with a Single-Context Talker Announcement [item a) in 51.3.4.1.1].
b) **Multi-Context Listener Attachment:** A Listener Attachment that is associated with a Multi-Context Talker Announcement [item b) in 51.3.4.1.1].

A Single-Context Listener Attach registration is said to be associated with a Single-Context Talker Announce declaration, if all of the following conditions are met:

c) The Listener Attach registration is on the same Port as the Talker Announce declaration.
d) The StreamId (51.5.4.1) of the Listener Attach registration is identical to the StreamId (51.5.3.1) of the Talker Announce declaration.
e) The VID (51.5.4.2) of the Listener Attach registration is identical to the VID (51.5.3.5.3) of the Talker Announce declaration.

A Multi-Context Listener Attach registration is said to be associated with a Multi-Context Talker Announce declaration, if all of the following conditions are met:

f) The Listener Attach registration is on the same Port as the Talker Announce declaration.
g) The StreamId (51.5.4.1) of the Listener Attach registration is identical to the StreamId (51.5.3.1) of the Talker Announce declaration.
h) The set of VLAN Context(s) (51.5.4.4) indicated by the Listener Attach registration is identical to the set of VLAN Context(s) indicated by the Talker Announce declaration (51.5.3.9).

A Listener Attach declaration is said to be associated with a Talker Announce registration, if all of the following conditions are met:

i) The Listener Attach declaration is on the same Port as the Talker Announce registration.
j) The StreamId (51.5.4.1) of the Listener Attach declaration is identical to the StreamId (51.5.3.1) of the Talker Announce registration.
k) The Listener Attach declaration wholly or partially results from a given Listener Attach registration whose associated Talker Announce declaration wholly or partially results from the Taker Announce registration.

A Listener Attach registration with an associated Talker Announce declaration on a Bridge Port indicates a reservation request for transmission of a stream or a Compound or Member Stream through that Port and, upon successful resource allocation, is associated with a dedicated reservation in the Bridge Port. A reservation maintained in a Bridge Port is identified by <StreamId, VID> with the corresponding values contained in the associated Listener Attach registration.

51.3.5.2 Listener Attach propagation

A Listener Attachment follows the path(s) traversed by its associated Talker Announcement in reversed direction.

A Listener Attach registration on a Bridge Port with an associated Talker Announce declaration is propagated to each other Port that contains an originating Talker Announce registration of the associated Talker Announce declaration. A Listener Attach registration on a Bridge Port without an associated Talker Announce declaration is not propagated by the Bridge.

51.3.5.3 Listener Attach merge

Listener Attach merge is performed, when more than one Listener Attach registration whose associated Talker Announce declaration has a common originating Talker Announce registration on a given Bridge Port is propagated to that Port, to result in a joint Listener Attach declaration associated with that Talker Announce registration.

In a Single-Context or Multi-Context Listener Attachment for a stream, Listener Attach merge can occur when multiple Listener Attachment processes originating from different Listeners of that stream converge on a Bridge Port, indicating a potential place for multicast forwarding the stream. Additionally, in a Multi-Context Listener Attachment for a Compound Stream, Listener Attach merge can also occur when multiple Listener Attachment processes running in different VLAN Contexts converge on a Bridge Port, indicating a potential place for applying stream splitting (51.3.8.4) to the stream.

51.3.5.4 Listener Attach status

The Listener Attach status of a Listener Attach declaration on a Port is associated with one or more paths, depending on the VLAN Context(s) in which the Listener Attach declaration is made, from the Port to each Listener whose participation in the Listener Attachment has been perceived on that Port, and indicates one of the following:

a) **Attach Ready:** In the case of Single-Context Listener Attachment, indicating a reservation has been successfully made on each path to each Listener. In the case of Multi-Context Listener Attachment, indicating a reservation has been successfully made on at least one path to one Listener, while the reservation status of each path is indicated by the Path status (51.3.5.5).
b) **Attach Fail:** A reservation has failed on each path to each Listener;
c) **Attach Partial Fail:** A reservation has been successfully made on the path to at least one of these Listeners, but has failed on the path to other Listeners. This status is only applicable to Single-Context Listener Attachments.

51.3.5.5 Path status

A Multi-Context Listener Attachment also signals the Path status on a per VLAN Context basis. The Path status of a Listener Attach declaration on a Port for a given VLAN Context is associated with a single path within that VLAN Context from the Port to each Listener whose participation in the Listener Attachment has been perceived on that Port, and indicates one of the following:

a) **Faultless Path**: Resource allocation has succeeded without encountering any problems on the path.
b) **Faulty Path**: Resource allocation has encountered one or more problems on the path and failed wholly or partially on the path.
c) **Unresponsive Path**: No Listener Attachment with respect to the path has been received.

NOTE—In Multi-Context Listener Attachments, resource allocation can succeed partially on a path as a result of the fact that the path goes through one or more places of stream merging in the network, as described in 51.3.8.5.

51.3.6 Resource allocation constraints

Resource allocation is constrained by limited capacity of Bridges to ensure bounded latency and zero congestion loss for reserved streams. The capacity of a Bridge to accommodate streams is described in terms of the following three constraints:

a) **Latency constraint:** The latency constraint of a Bridge is defined by the per-RA class, per reception/transmission Port pair maxHopLatency parameter [item d) in 51.8.4.9], which is treated as a latency upper bound provided by the Bridge to each reserved stream that is received on a Bridge Port and transmitted on another Port in an RA class. A stream requesting a reservation is said to meet the latency constraint of a Bridge, if allowing transmission of this stream would not cause any stream being reserved in this Bridge to suffer from a latency higher than the corresponding maxHopLatency value.

NOTE 1—The latency bound check as required by the latency constraint requires the ability to determine the worst-case maximum latency of streams for each RA class in a Bridge. The method used for this purpose is specific to the RA class template used by each RA class.

b) **Bandwidth constraint:** The bandwidth constraint of a Bridge is defined by the per-RA class, per-Port maxBandwidth parameter, as specified in item f) in 51.8.4.8, which places an upper bound on the bandwidth that can be reserved for use by an RA class on a transmission Port. A stream requesting a reservation is said to meet the bandwidth constraint of a Bridge, if the total bandwidth needed to transmit that stream and all the streams being reserved in the same RA class and on the same Port does would not exceed the amount indicated in the corresponding maxBandwidth value.
c) **Resource constraint:** The resource constraint of a Bridge is associated with the availability of the Bridge resources required for handling of streams in operations such as filtering in FDB, buffering, shaping and gating. A stream requesting a reservation is said to meet the resource constraint of a Bridge, if the Bridge has sufficient resources available for supporting all the operations required for handling that stream.

NOTE 2—The determination of resource availability is heavily dependent on how resources are managed in a Bridge for use by streams and other traffic, and thus is an implementation choice.

Meeting all of these resource allocation constraints imposed by a Bridge is one of the necessary conditions for creating a reservation in the Bridge. The RAP Failure Codes associated with the resource allocation constraints are defined in Table 51-13.

51.3.7 Stream Rank and reservation importance

RAP supports the concept of the Rank (46.2.3.21) and gives streams with Rank zero higher precedence than streams with Rank one in a manner that allows Bridges to make a reservation for a stream with Rank zero in case of lacking resources by removing one or more existing reservations associated with streams with Rank one. Each reservation removed under such circumstances is termed “the reservation preempted by Rank zero” and uses a specific RAP Failure Code as defined in Table 51-13.

NOTE 3—RAP allows reservations of Rank one streams to be preempted only by reservations of Rank zero streams. Reservations of streams with the same Rank, regardless Rank zero or one, are handled on a first-come first-served basis.

The order used by a Bridge in the selection of a reservation, in preference to others, to be preempted by Rank zero is determined based on the reservation importance, i.e., a least important reservation is to be removed first. Among a set of reservations on a Bridge Port, each identified by <StreamId, VID> and with a reservationAge value [item e) in 51.8.4.3], one reservation is considered more important than another, according to the following rules in the shown order:

a) A Reservation with a numerically larger reservationAge value is more important.

b) If reservationAge is the same, the one with a numerically smaller StreamId is more important.
c) If StreamId is the same, the one with a numerically smaller VID is more important.

51.3.8 Resource allocation for seamless redundancy

RAP supports resource allocation for transmission of a stream along multiple paths within a VLAN Bridged network by using the seamless redundancy techniques described by IEEE Std 802.1CB. The subsequent subclauses describe additional capabilities and operations required for resource allocation for seamless redundancy, while the definitive specification is contained in 51.5, 51.6 and 51.8. For illustrative purposes, some resource allocation examples for seamless redundancy are provided in Annex Y.

51.3.8.1 FRER-capable stations

Resource allocation for seamless redundancy operates in a network where a set of FRER-capable stations, either or both FRER-capable end stations and FRER-capable Bridges, are placed at specific locations to perform FRER operations on Compound and Member Streams with corresponding FRER functions, as shown in Table 51-4.

Table 51-4—FRER functions and operations in FRER-capable stations

		FRER-capable stations		
		FRER-capable Talker	**FRER-capable Bridge**	**FRER-capable Listener**
FRER operations	**redundancy tagging (51.3.8.3)**	sequence generation function, Redundancy tag Stream encode/decode function		(none)
	stream splitting (51.3.8.4)	Stream splitting function	(Multicast)[a]	
		active Stream identification function[b]		
	stream merging (51.3.8.5)	(none)	sequence recovery function, active Stream identification function[b]	
	redundancy untagging (51.3.8.3)		Redundancy tag Stream encode/decode function	

[a]The stream splitting operation in a FRER-capable Bridge relies on the multicast ability of the Forwarding Process to replicate streams, without the need to use Stream splitting function defined in IEEE Std 802.1CB.
[b]This function refers to the Active Destination MAC and VLAN Stream identification function defined in IEEE Std 802.1CB. As in resource allocation each Member Stream associated with a Compound Stream identified by a <DestinationMacAddress, VID> pair is identified by the same DestinationMacAddress and a different VID, this function is deployed in the stream splitting and stream merging operations to change only the VID value in stream frames.

A Multi-Context Talker Announcement used in resource allocation for a Compound Stream carries additional information needed to determine the locations and the FRER operations required for the Compound Stream and its associated Member Streams. Once a reservation decision has been made in the associated Multi-Context Listener Attachment, each required FRER function is configured in the previously determined FRER-capable stations.

51.3.8.2 Redundancy Context

Resource allocation for seamless redundancy operates in a network in which one or more instances of redundant trees (3.207) are established and maintained by other protocols such as ISIS-PCR or management. Each Multi-Context Talker Announcement for a Compound Stream operates within a particular instance of redundant trees. The group of VLAN topologies associated with an instance of redundant trees comprising two or more trees forms the set of correlated VLAN Contexts, termed “the Redundancy Context”. A Redundancy Context is identified by the numerically smallest VLAN Context ID in the set, termed “the Redundancy Context ID”. Multiple Redundancy Contexts, each with distinct set of VLAN Context IDs, can exist in a network due to the existence of multiple instances of redundant trees.

NOTE—For example, there can be one instance of redundant trees for each Bridge located at the edge of the network and rooted at that Bridge, with the intention that each Compound Stream using the redundant trees rooted at a Bridge where it enters the network be established for transmission along the multiple paths that are disjoint from each other.

The per-Bridge parameter redundancyContext (51.8.4.10), configurable as a managed object defined in Table 12-43 and containing the configuration of each existing Redundancy Context in the network, provides the information needed by the operation of resource allocation such as Talker Announce merge as described in 51.3.4.3. A network-wide consistent configuration of this parameter in each Bridge is required for the successful operation of resource allocation for seamless redundancy. The value of this parameter is also carried in the RA Advertisement made by Bridges, which allows each end station to learn from its neighboring Bridge the Redundancy Context configuration in the network.

51.3.8.3 Redundancy tagging and redundancy untagging

In resource allocation, redundancy tagging refers to a FRER operation in which a FRER-capable Talker or a FRER-capable Bridge uses the particular FRER functions, as shown in Table 51-4, to transform an ordinary stream into a Compound Stream by inserting a Redundancy tag (R-TAG) into the data frames of the stream. Redundancy untagging refers to a FRER operation in which a FRER-capable Bridge or a FRER-capable Listener uses the particular FRER function, as shown in Table 51-4, to transform a Compound Stream back into an ordinary stream by removing the R-TAG carried in the data frames of the Compound Stream.

The locations in a network (including end stations) to apply redundancy tagging and redundancy untagging to a Compound Stream are dependent on the placement of the FRER-aware stations in the network, and are determined in resource allocation as follows:

a) The location of redundancy tagging can be either at the Talker, if the Talker is FRER-capable, or at a FRER-capable Bridge through which the stream enters the network, if the Talker is not FRER-capable. If neither of these two locations exists, resource allocation for transmission of the Compound Stream to all Listeners fails.
b) The location of redundancy untagging can be either on a FRER-Bridge Port that is a stream merging Port for all the VLAN Contexts in the Redundancy Context used by this Compound Stream, as described in 51.3.8.5, or in a Listener, if no such FRER-capable Bridge exists and the Listener is FRER-capable. If neither of these two locations exists, resource allocation for transmission of the Compound Stream to that Listener fails.

The Boolean flag RTagStatus (51.5.3.8.1) encoded in the Talker Announce attribute for use by Multi-Context Talker Announcements provides the information needed for determination of redundancy tagging and untagging locations.

51.3.8.4 Stream splitting

In resource allocation, stream splitting refers to a FRER operation in which a FRER-capable Talker or a FRER-capable Bridge uses the particular FRER functions, as shown in Table 51-4, to split a Compound Stream or a Member Stream into multiple Member Streams.

The locations in a network (including end stations) to apply stream splitting to a Compound Stream that uses a particular Redundancy Context are dependent on the placement of the FRER-capable stations and also the VLAN topologies of the Redundancy Context, and are determined in resource allocation as follows:

a) If the Talker is FRER-capable, it initiates a Multi-Context Talker Announcement by making multiple Talker Announce declarations, each using a different VLAN Context in the Redundancy Context, indicating that the Compound Stream is to be split into Member Streams by the Talker.
b) If the Talker is not FRER-capable, it initiates a Multi-Context Talker Announcement by making a single Talker Announce declaration that include all the VLAN Contexts in the Redundancy Context, indicating that the stream is expected to be transformed into a Compound Stream (i.e., redundancy tagging) and then split into Member Streams by a FRER-capable Bridge proxying for this Talker.
c) A FRER-capable Bridge involved in propagating a Talker Announce registration for the Compound Stream is considered to be the stream splitting location, if all of the following conditions are met:
    1) The Talker Announce registration contains more than one VLAN Contexts.
    2) This Talker Announce registration is to be propagated to at least two other Bridge Ports. And at least one of these Ports is not in all of the VLAN Contexts as contained in the Talker Announce registration. This also indicates the fact that the VLAN topologies in use are disjoint somehow on these Ports.
    3) All of the Ports to which the Talker Announce registration is to be propagated are in the member set of at least one VLAN Context used in the propagation of the Talker Announce registration. This is considered a necessary condition for the Bridge to multicast the stream to each of those Ports prior to using the active Stream identification to translate the stream into member streams with different VIDs on different Ports.

To enforce proper placement of FRER-capable stations and VLAN configuration required by stream splitting, a Bridge fails a Multi-Context Talker Announce registration with the RAP Failure Code 0x09, as specified in Table 51-13, in one of the following two circumstances:

d) A FRER-capable Bridge receives a Multi-Context Talker Announce registration that meets the conditions c).1) and c).2) but does not meet the condition c).3).
e) A Bridge that is not FRER-capable receives a Multi-Context Talker Announce registration that meets the conditions c).1) and c).2).

NOTE—Stream splitting is distinguished from the normal multicast, as the former transforms replicated streams into separate Member Streams with different VIDs. If a FRER-Bridge propagates a Multi-Context Talker Announcement for a given Compound Stream to several Ports within only one VLAN Context or multiple VLAN Contexts that overlap completely in the Bridge, it will apply just multicast instead of stream splitting to that Compound Stream.

51.3.8.5 Stream merging

In resource allocation, stream merging refers to a FRER operation in which a FRER-capable Bridge or a FRER-capable Listener performs the particular FRER function, as shown in Table 51-4, on a set of Member Streams that are originally split from a Compound Stream.

The locations in a network (including end stations) to apply stream merging to the Member Streams belonging to a Compound Stream within a particular Redundancy Context are dependent on the placement of the FRER-capable stations and also the VLAN Contexts of the Redundancy Context, and are determined in resource allocation as follows:

a) A FRER-capable Bridge Port is considered to be the stream merging Port, if more than one Talker Announce registration associated with the Compound Stream is expected to be propagated by the Bridge from a different Port to that Port. Such a stream merging Port is in fact a converging point of all or some of the VLAN topologies in the Redundancy Context used by the Compound Stream. There can be more than one stream merging Port in a FRER-Bridge for the Compound Stream.
b) A FRER-capable Listener of the Compound Stream is responsible for stream merging and redundancy untagging as well, if it receives more than one Talker Announce registration associated with the Compound Stream, each containing one or more VLAN Contexts in the Redundancy Context used by the Compound Stream.

51.3.9 Relationship of RAP to MSRP

RAP has conceptual and functional similarities to MSRP. Table 51-5 summarizes the similarities and differences between RAP and MSRP.

Table 51-5—RAP/MSRP differences

	MSRP (Clause 35)	**RAP (Clause 51)**
Attribute registration framework	MRP (Clause 10)	LRP (IEEE Std 802.1CS)
Operational modes	peer-to-peer (MSRPv0) MRP External Control (MSRPv1)	Native, Controlled and Proxy system
Attribute types	Table 35-1	Table 51-8
Traffic classes for streams	SR class	RA class
Traffic specification	35.2.2.8.4 (MSRPv0) 35.2.2.10.6 (MSRPv1)	51.5.3.6 51.5.3.7
Queuing and transmission functions for streams	FQTSS in Clause 34	RA class template based
Support for multiple-path streams	only in MSRPv1 with CNC	51.3.8

51.4 Definition of LRP protocol elements

This subclause specifies the LRP protocol elements that are specific to the operation of RAP.

51.4.1 Value of AppId

The general format of AppIds is defined in 9.2 of IEEE Std 802.1CS-2020. The AppId that identifies the RAP specified in this standard shall take the values shown in Table 51-6.

Table 51-6—The AppId value for RAP

Field	Value	Length (octets)	Offset (octets)
OUI or CID	00-80-C2	3	0
Application Sub-ID	1	1	3

51.4.2 Use of LLDP

When the IEEE Std 802.1AB Link Layer Discovery Protocol (LLDP) is deployed to advertise and discover target ports, the destination MAC address of the LLDP agent that is selected to carry the AppId (51.4.1) in an LRP ECP Discovery TLV (C.2.1 of IEEE Std 802.1CS-2020) and/or an LRP TCP Discovery TLV (C.2.2 of IEEE Std 802.1CS-2020), shall be the Nearest Bridge group address (01-80-C2-00-00-0E) as specified in Table 8-1, Table 8-2, and Table 8-3.

51.4.3 LRP-DT data transport mechanisms

A RAP Native system shall support either the LRP-DT ECP mechanism or the LRP-DT TCP mechanism, and may support both. A RAP Proxy system shall support the LRP-DT TCP mechanism.

NOTE—In a RAP Native system that supports both ECP and TCP, the choice of which one is used on a given target port is made by the LRP operating on that port.

51.4.3.1 Use of LRP ECP mechanism

In a RAP Native system that supports the LRP-DT ECP mechanism, the destination MAC address of the ECP instance operating on each target port of that system shall be the Nearest Bridge group address (01-80-C2-00-00-0E) as specified in Table 8-1, Table 8-2, and Table 8-3.

51.4.3.2 Use of LRP TCP mechanism

<< Editor's note: it is unclear to the editor what needs to be specified for use of TCP (see Annex Z.2.2. >>

51.4.4 Application Information TLV

An Application Information TLV, in the format specified in 9.4.2.10 of IEEE Std 802.1CS-2020 and exchanged via LRP Hello LRPDUs, is used by each RAP Participant to advertise the information about its capabilities and other operational parameters to its neighbor(s). Figure 51-4 shows the encoding of the Value field of the Application Information TLV:

	Octet	Length
ProtocolVersion	1	1

Figure 51-4—Value of Application Information TLV

a) **ProtocolVersion**: The ProtocolVersion for the version of RAP defined in this standard takes the hexadecimal value 0x00.

51.4.5 Use of LRP-DS service interface

For the purposes of the operation of a RAP Participant (51.7), a set of primitives and associated parameters chosen from the LRP-DS service interface defined in Clause 10 of IEEE Std 802.1CS-2020 are summarized in Table 51-7.

Table 51-7—LRP-DS service primitives used by RAP

Primitive	Subclause in IEEE Std 802.1CS-2020	Parameter	Item in IEEE Std 802.1CS-2020
LOCAL_TARGET_PORT.request	10.2.2	(see Table 51-11)	
NEIGHBOR_TARGET_PORT.request	10.2.3		
ASSOCIATE_PORTAL.request	10.2.4	rPortalId	10.2.4.1 a)
		rIsApproved	10.2.4.1 b)
FIRST_HELLO.indication	10.2.5	iPortalId	10.2.5.1 a)
		iHelloData	10.2.5.1 b)
PORTAL_STATUS.indication	10.2.6	iPortalId	10.2.6.1 a)
		iAssociationStatus	10.2.6.1 b)
WRITE_RECORD.request	10.3.1.1	rPortalId	10.3.1.1.1 a)
		rRecordNumber	10.3.1.1.1 b)
		rRecordData	10.3.1.1.1 c)
DELETE_RECORD.request	10.3.2.1	rPortalId	10.3.2.1.1 a)
		rRecordNumber	10.3.2.1.1 b)
RECORD_WRITTEN.indication	10.3.2.3	iPortalId	10.3.2.3.1 a)
		iRecordNumber	10.3.2.3.1 b)
		iRecordData	10.3.2.3.1 c)

NOTE—It is assumed that a FIRST_HELLO.indication primitive invoked due to receiving a Hello message on a given Port by LRP be passed to a RAP Participant on that Port.

51.5 RAP attribute and TLV encoding definitions

This subclause defines the RAP attributes and associated TLV encoding.

51.5.1 General TLV definition

The information of RAP attributes is encoded in a Type-Length-Value (TLV) format, which consists of a 1-octet Type field that indicates the type of that TLV, a 2-octet Length field that indicates the number of octets in the Value field, and then a Value field. The Value field of a TLV encodes zero or more fixed-size parameters, followed by zero or more TLVs of specific types. Figure 51-5 illustrates the general TLV format.

	Octet	Length
Type	1	1
Length	2	2
Value	4	Fixed or variable

Figure 51-5—General TLV format

There are two categories of TLVs defined for RAP, attribute TLVs and sub-TLVs. An attribute TLV encodes a RAP attribute and can include one or more sub-TLVs in the Value field. An sub-TLV can further include one or more sub-TLVs. Attributes TLVs, as top-level TLVs, are never included in a sub-TLV or another attribute TLV. Table 51-8 lists the set of RAP attribute TLVs and sub-TLVs, and the values of the Type and Length field for each of them.

Table 51-8—RAP TLV and sub-TLV Type field and Length field values

TLV name	Type field (1 octet)	Length field (2 octets)	Reference
RA attribute TLV	0x00	variable	51.5.2
Talker Announce attribute TLV	0x01	variable	51.5.3
Listener Attach attribute TLV	0x02	10	51.5.4
RA Class Descriptor sub-TLV	0x20	variable	51.5.2.1
Redundancy Context sub-TLV	0x21	variable	51.5.2.2
Data Frame Parameters sub-TLV	0x22	8	51.5.3.5
Token Bucket TSpec sub-TLV	0x23	16	51.5.3.6
MSRP TSpec sub-TLV	0x24	8	51.5.3.7
Redundancy Control sub-TLV	0x25	variable	51.5.3.8
VLAN Context Information sub-TLV	0x26	variable	51.5.3.9
Failure Information sub-TLV	0x27	9	51.5.3.10
Organizationally Defined sub-TLV	0x28	variable	51.5.3.11
VLAN Context Status sub-TLV	0x29	variable	51.5.4.4
Reserved for future standardization	0x03-0x1F, 0x2A-0xFF	-	-

The subsequent subclauses specify the encoding of the Value field for each attribute TLV and sub-TLV, by using the “big-endian” convention, in which higher significance bytes and bits within each byte appear to the left of and above bytes and bits of lower significance. Any fields that are labeled as “Reserved” are transmitted as zero and ignored on receipt. In addition, the term “the stream” used in the specification of Talker Announce attribute TLV and the sub-TLVs thereof refers to a stream for which the Talker Announce attribute is declared.

51.5.2 RA attribute and TLV encoding

The RA attribute is used in RA Advertisements (51.3.3). The Value field of an RA attribute TLV contains one or more RA class Descriptor sub-TLVs, followed by zero or more Redundancy Context sub-TLVs, as illustrated in Figure 51-6.

	Octet	Length
1 or more RA Class Desriptor sub-TLVs	1	variable
zero or more Redundancy Context sub-TLVs	variable	variable

Figure 51-6—Value of RA attribute TLV

51.5.2.1 RA Class Descriptor sub-TLV

Each RA Class Descriptor sub-TLV present in the RA attribute represents a distinct RA class (51.3.2). Figure 51-7 shows the encoding of the Value field of an RA Class Descriptor sub-TLV.

	Octet	Length
RaClassId	1	1
RaClassPriority	2	1
RTID	3	4
ProposedMaxHopLatency	7	4
RaClassTemplateDefinedData	11	variable

Figure 51-7—Value of RA Class Descriptor sub-TLV

51.5.2.1.1 RaClassId

A 1-octet unsigned integer containing the RA class ID (51.3.2.1).

51.5.2.1.2 RaClassPriority

A 1-octet unsigned integer containing the RA class priority (51.3.2.2).

51.5.2.1.3 RTID

A 4-octet RTID (51.3.2.3) identifying the RA class template used by the RA class.

51.5.2.1.4 ProposedMaxHopLatency

A 4-octet integer containing the proposedMaxHopLatency [item h) in 51.8.4.8] value associated with the RA class and the Port on which an RA declaration is made.

51.5.2.1.5 RaClassTemplateDefinedData

The encoding of this field and the semantics associated with its values if any, is specific to the RA class template identified by the value contained in 51.5.2.1.3.

51.5.2.2 Redundancy Context sub-TLV

Each Redundancy Context sub-TLV present in the RA attribute represents a distinct Redundancy Context (51.3.8.2). The Value field of a Redundancy Context sub-TLV contains one or more VLAN Context tuples, each encoding in the VlanContextId field a 12-bit VLAN Context ID, followed by a 4-bit Reserved field, as illustrated in Figure 51-8.

		Octet	Length
VLAN Contex tuple 1	VlanContextId	1	12 bits
	Reserved	2	4 bits
	...		
VLAN Context tuple n	VlanContextId	2n-1	12 bits
	Reserved	2n	4 bits

Figure 51-8—Value of Redundancy Context sub-TLV

51.5.3 Talker Announce attribute and TLV encoding

The Talker Announce attribute is used in Talker Announcements (51.3.4). Figure 51-9 shows the encoding of the Value Field of the Talker Announce attribute TLV.

	Octet	Length
StreamId	1	8
StreamRank	9	1
AccumulatedMaximumLatency	10	4
AccumulatedMinimumLatency	14	4
Data Frame Parameters sub-TLV	18	11
Token Bucket TSpec sub-TLV or MSRP TSpec sub-TLV	29	19 or 11
Redundancy Control sub-TLV *(only for Multi-Context Talker Announcement)*	variable	variable
0 or 1 Failure Information sub-TLV	variable	variable
0 or more Organizationally Defined sub-TLVs	variable	variable

Figure 51-9—Value of Talker Announce attribute TLV

For support of two types of Talker Announcements, the encoding of the Talker Announce attribute follows the following rules:

a) The Talker Announce attribute used in a Single-Context Talker Announcement [item a) in 51.3.4.1.1] contain no Redundancy Control sub-TLV (51.5.3.8).
b) The Talker Announce attribute used in a Multi-Context Talker Announcement [item b) in 51.3.4.1.1] always contains a Redundancy Control sub-TLV (51.5.3.8).

51.5.3.1 StreamId

An 8-octet field encoding the StreamID element as specified in 46.2.3.1.

51.5.3.2 StreamRank

A 1-octet field encoding a Rank value as specified in 46.2.3.2.1.

51.5.3.3 AccumulatedMaximumLatency

A 4-octet field encoding the AccumulatedLatency element as specified in 46.2.5.2.

51.5.3.4 AccumulatedMinimumLatency

A 4-octet unsigned integer, indicating the minimum latency, in nanoseconds, that a single frame of the stream can encounter when transmitted from the Talker along a given path to the declaring Port of this attribute.

51.5.3.5 Data Frame Parameters sub-TLV

This sub-TLV contains a set of parameters whose values are carried in the data frames of the stream when transmitted through the declaring Port of this attribute. Figure 51-10 shows the encoding of the Value field of the Data Frame Parameters sub-TLV.

	Octet	**Length**
DestinationMacAddress	1	6
Priority	7	3 bits
Reserved	7	1 bit
VID	7	12 bits

Figure 51-10—Value of Data Frame Parameters sub-TLV

51.5.3.5.1 DestinationMacAddress

A 6-octet destination MAC address of the data frames of the stream.

51.5.3.5.2 Priority

A 3-bit unsigned integer, indicating the priority to be encoded in the PCP field of the VLAN tag, with which the data frames of the stream are tagged. This priority value is used by each receiving Bridge to associate the stream to a local RA class of the same priority.

51.5.3.5.3 VID

A 12-bit VID. The semantics of this field is dependent on the type of a Talker Announcement in which the Talker Announce attribute is used, as follows:

a) In a Single-Context Talker Announcement, this field indicates a VID to be encoded in the VLAN tag with which the data frames of the stream are tagged, and also a single VLAN Context used by the Talker Announce attribute.
b) In the case of Multi-Context Talker Announcement, this field is set to one of the VLAN Context ID in the Redundancy Context (51.3.8.2) used by the Multi-Context Talker Announcement.

51.5.3.6 Token Bucket TSpec sub-TLV

This sub-TLV contains a TSpec that makes use of a token bucket algorithm to describe the traffic characteristics of a stream. Figure 51-11 shows the encoding of the Value field of the Token Bucket TSpec sub-TLV.

	Octet	Length
MaxTransmittedFrameLength	1	2
MinTransmittedFrameLength	3	2
CommittedInformationRate	5	8
CommittedBurstSize	13	4

Figure 51-11—Value of Token Bucket TSpec sub-TLV

51.5.3.6.1 MaxTransmittedFrameLength

A 2-octet unsigned integer, indicating the maximum frame length of the stream, in octets, including all media-dependent overhead (12.4.2.2).

51.5.3.6.2 MinTransmittedFrameLength

A 2-octet unsigned integer, indicating the minimum frame length of the stream, in octets, including all media-dependent overhead (12.4.2.2).

51.5.3.6.3 CommittedInformationRate

A 8-octet unsigned integer, indicating the committed information rate, in bits per second, of the token bucket used to describe the traffic characteristics of the stream.

51.5.3.6.4 CommittedBurstSize

A 4-octet unsigned integer, indicating the committed burst size, in bits, of the token bucket used to describe the traffic characteristics of the stream.

51.5.3.7 MSRP TSpec sub-TLV

The MSRP TSpec sub-TLV contains a TSpec that is measured in terms of the same set of parameters, except the TransmissionSelection parameter, as specified in Table 46-8. Figure 51-12 shows the encoding of the Value field of the MSRP TSpec sub-TLV.

	Octet	Length
Interval	1	4
MaximumFramesPerInterval	5	2
MaximumFrameSize	7	2

Figure 51-12—Value of MSRP TSpec sub-TLV

51.5.3.7.1 Interval

A 4-octet unsigned integer, indicating the interval of time, in nanoseconds, over which the MaximumFramesPerInterval (51.5.3.7.2) value contained in the same MSRP TSpec sub-TLV is measured.

NOTE—Although the Interval element is specified as a per-stream parameter, the capability of a given RA class to allow its streams to use distinct Interval values can be constrained by the shaping or scheduling method used by that RA class. For example, an RA class that supports the credit-based shaper would require each stream to set its Interval parameter to a class measurement interval value that is common to all the streams of that RA class, in the same manner as SR class A or B does as described in 34.4. The constraints, if any, relating to the use of the Interval parameter, are to be described as part of an RA class template.

51.5.3.7.2 MaximumFramesPerInterval

A 2-octet unsigned integer, indicating the maximum number of maximum sized frames that the stream's Talker can generate in one Interval (51.5.3.7.1).

51.5.3.7.3 MaximumFrameSize

A 2-octet unsigned integer, indicating the maximum fame size, in octets, that the Talker can generate for the stream. This parameter takes no account of any medium-specific framing overheads (e.g., preamble, 802.3 header, VLAN tag, CRC, interframe gap) associated with transmitting the stream on any Port.

51.5.3.8 Redundancy Control sub-TLV

The Redundancy Control sub-TLV contains the information exclusively used by Multi-Context Talker Announcements. Figure 51-13 shows the encoding of the Value filed of the Redundancy Control sub-TLV.

	Octet	Length
RTagStatus	1	1 bit
Reserved	1	7 bits
1 or more VLAN Context Informtion sub-TLVs	2	vairable

Figure 51-13—Value of Redundancy Control sub-TLV

51.5.3.8.1 RTagStatus

A Boolean value indicating whether the data frames of the stream are (TRUE) or are not (FALSE) to contain a Redundancy tag (R-TAG, see IEEE Std 802.1CB), when they are transmitted through the Port that declares this sub-TLV.

51.5.3.9 VLAN Context Information sub-TLV

Each VLAN Context Information sub-TLV present in the Redundancy Control sub-TLV represents a distinct VLAN Context (51.3.4.1.1). The Value field encodes in the VlanContextId field a 12-bit VLAN Context ID, followed by a 4-bit Reserved field and then zero or one Failure Information sub-TLV (51.5.3.10), as illustrated in Figure 51-14.

	Octet	Length
VlanContextId	1	12 bits
Reserved	2	4 bits
0 or 1 Failure Information sub-TLV	3	vairable

Figure 51-14—Value of VLAN Context Information sub-TLV

51.5.3.10 Failure Information sub-TLV

The Failure Information sub-TLV contains the information about the location and the type of a failure encountered in the Talker Announcement. Figure 51-15 shows the encoding of the Value field of the Failure Information sub-TLV.

	Octet	Length
SystemId	1	8
FailureCode	9	1

Figure 51-15—Value of Failure Information sub-TLV

51.5.3.10.1 SystemId

An 8-octet unsigned integer, indicating the identifier of a Bridge or end station at which the failure occurs. The SystemId value for an end station shall be the 6-octet MAC Address of the end station's port extended to 8 octets by prepending 2 octets of zero. The SystemId value for a Bridge shall be the 6-octet Bridge Identifier (13.26.2) of the Bridge extended to 8 octets by prepending 2 octets of zero.

51.5.3.10.2 Failure Code

A 1-octet unsigned integer, encoding a RAP Failure Code specified in Table 51-13.

51.5.3.11 Organizationally Defined sub-TLV

The Organizationally Defined sub-TLV contains information defined by an organization that owns an OUI or a CID obtained from the IEEE registration authority. Figure 51-16 shows the encoding of the Value field of the Organizationally Defined sub-TLV.

	Octet	Length
OUI/CID	1	3
OrganizationallyDefinedData	4	varaible

Figure 51-16—Value of Organizationally Defined sub-TLV

51.5.3.11.1 OUI/CID

A 3-octet OUI or an CID.

51.5.3.11.2 OrganizationallyDefinedData

The encoding of this field and the semantics associated with its values if any, is specific to and defined by the organization that owns the OUI/CID contained in 51.5.3.11.1.

51.5.4 Listener Attach attribute and TLV encoding

The Listener Attach attribute is used in Listener Attachments (51.3.5). Figure 51-17 shows the encoding of the Value field of the Listener Attach attribute TLV.

	Octet	**Length**
StreamId	1	8
VID	9	12 bits
ListenerAttachStatus	10	4 bits
VLAN Context Status sub-TLV *(only for Multi-Context Listener Attachment)*	11	vairable

Figure 51-17—Value of Listener Attach attribute TLV

For support of two types of Listener Attachments, the encoding of the Listener Attach attribute follows the following rules:

a) The Listener Attach attribute used in a Single-Context Listener Attachment [item a) in 51.3.5.1] contains no VLAN Context Status sub-TLV (51.5.4.4).
b) The Listener Attach attribute used in a Multi-Context Listener Attachment [item b) in 51.3.5.1] contains one VLAN Context Status sub-TLV (51.5.4.4).

51.5.4.1 StreamId

An 8-octet field encoding the StreamID element as specified in 46.2.3.1.

51.5.4.2 VID

A 12-bit integer, indicating a VID to be encoded in the VLAN tag with which the data frames of the stream are tagged.

51.5.4.3 ListenerAttachStatus

A 4-bit field, taking one of the following enumerated values to indicate the Listener Attach status (51.3.5.4):

a) **0: Attach Ready** [item a) in 51.3.5.4];
b) **1: Attach Fail** [item b) in 51.3.5.4];
c) **2: Attach Partial Fail** [item c) in 51.3.5.4].

51.5.4.4 VLAN Context Status sub-TLV

The VLAN Context Status sub-TLV contains the information exclusively used by Multi-Context Listener Attachments. The Value field contains one or more VLAN Context tuples, each encoding in VlanContextId field a 12-bit VLAN Context ID, followed by a 4-bit PathStatus field, as illustrated in Figure 51-18.

		Octet	Length
VLAN Contex tuple 1	VlanContextId	1	12 bits
	PathStatus	2	4 bits
	...		
VLAN Context tuple n	VlanContextId	2n-1	12 bits
	PathStatus	2n	4 bits

Figure 51-18—Value of VLAN Context Status sub-TLV

51.5.4.4.1 PathStatus

A 4-bit field, taking one of the following enumerated values to indicate the Path status (51.3.5.5):

a) **0: Faultless Path** [item a) in 51.3.5.5];
b) **1: Faulty Path** [item b) in 51.3.5.5];
c) **2: Unresponsive Path** [item c) in 51.3.5.5].

51.6 RAP Endpoint

51.6.1 RAP Endpoint overview

The RAP Endpoint of a RAP End Instance is responsible for the following:

— Handling of resource allocation for an end station.
— Generation of attributes for declarations and processing of received attribute registrations.
— Provision of service interface to higher layer entities for controlling of the signaling by end stations.

The operation of a RAP Endpoint is specified by the following:

— RAP Endpoint Service Interface (RESI) in 51.6.2.

51.6.2 RAP Endpoint Service Interface (RESI)

The RESI provided to higher layer entities comprises a set of primitives and associated parameters, which are divided into three groups, as summarized in Table 51-9.

Table 51-9—RAP Endpoint Service Interface primitives

	primitive name	reference
RA primitives (51.6.2.1)	DECLARE_RA.request	51.6.2.1.1
	WITHDRAW_RA.request	51.6.2.1.2
	REGISTER_RA.indication	51.6.2.1.3
	DEREGISTER_RA.indication	51.6.2.1.4
Talker primitives (51.6.2.2)	ANNOUNCE_STREAM.request	51.6.2.2.1
	TERMINATE_STREAM.request	51.6.2.2.2
	ATTACH_STREAM.indication	51.6.2.2.3
	DETACH_STREAM.indication	51.6.2.2.4
Listener primitives (51.6.2.3)	ATTACH_STREAM.request	51.6.2.3.1
	DETACH_STREAM.request	51.6.2.3.2
	ANNOUNCE_STREAM.indication	51.6.2.3.3
	TERMINATE_STREAM.indication	51.6.2.3.4

51.6.2.1 RA primitives

The RA primitives defined in the following subclauses are used by a RAP Endpoint user to control RA Advertisement (51.3.3) in an end station.

51.6.2.1.1 DECLARE_RA.request()

A DECLARE_RA.request primitive is invoked by the RAP Endpoint user to request the RAP Endpoint to make an RA declaration with the supplied values of the following parameters:

a) One or more sets of the following parameters, each set corresponding to an RA class:
   1) **RAClassId**: as specified in 51.5.2.1.1.
   2) **RaClassPriority**: as specified in 51.5.2.1.2.
   3) **RTID**: as specified in 51.5.2.1.3.
   4) **RaClassTemplateDefinedData**: as specified in 51.5.2.1.5.

51.6.2.1.2 WITHDRAW_RA.request()

A WITHDRAW_RA.request primitive is used by the RAP Endpoint user to request the RAP Endpoint to withdraw the existing RA declaration.

51.6.2.1.3 REGISTER_RA.indication()

A REGISTER_RA.indication primitive is used by the RAP Endpoint to notify the RAP Endpoint user of an RA registration with a set of parameter values abstracted from the registered RA attribute.

The primitive has the same set of parameters as in 51.6.2.1.1.

51.6.2.1.4 DEREGISTER_RA.indication()

A DEREGISTER_RA.indication primitive is used by the RAP Endpoint to notify the RAP Endpoint user that the RA registration previously received has been removed.

51.6.2.2 Talker primitives

The Talker primitives defined in the following subclauses are used in interaction between a RAP Endpoint user that represents Talker applications and the underlying RAP Endpoint.

51.6.2.2.1 ANNOUNCE_STREAM.request()

An ANNOUNCE_STREAM.request primitive is used by the RAP Endpoint user to request the RAP Endpoint to initiate a Talker Announcement with the supplied values of the following parameters:

a) **StreamId**: as specified in 51.5.3.1;
b) **StreamRank**: as specified in 51.5.3.2;
c) **AccumulatedMaximumLatency**: as specified in 51.5.3.3;
d) **AccumulatedMinimumLatency**: as specified in 51.5.3.4;
e) **DestinationMacAddress**: as specified in 51.5.3.5.1;
f) **Priority**: as specified in 51.5.3.5.2;
g) **VID**: as specified in 51.5.3.5.3;
h) Either a set of Token Bucket TSpec parameters as follows:
   1) **MaxTransmittedFrameLength**: as specified in 51.5.3.6.1;
   2) **MinTransmittedFrameLength**: as specified in 51.5.3.6.2;
   3) **CommittedInformationRate**: as specified in 51.5.3.6.3;
   4) **CommittedBurstSize**: as specified in 51.5.3.6.4;

   or a set of MSRP TSpec parameters as follows:

   5) **Interval:** as specified in 51.5.3.7.1;

    6) **MaximumFramesPerInterval**: as specified in 51.5.3.7.2;
    7) **MaximumFrameSize**: as specified in 51.5.3.7.3;

i) **OUI/CID:** as specified in 51.5.3.11.1;
j) **OrganizationallyDefinedData:** as specified in 51.5.3.11.2.

51.6.2.2.2 TERMINATE_STREAM.request()

A TERMINATE_STREAM.request primitive is used by the RAP Endpoint user to request the RAP Endpoint to terminate a Talker Announcement identified by the supplied StreamId (51.5.3.1) value.

51.6.2.2.3 ATTACH_STREAM.indication()

An ATTACH_STREAM.indication primitive is used by the RAP Endpoint to notify the RAP Endpoint user that a Listener Attachment has been received with the values of the following parameters abstracted from the received Listener Attachment:

a) **StreamId**: as specified in 51.5.4.1;
b) **VID**: as specified in 51.5.4.2;
c) **ListenerAttachStatus**: as specified in 51.5.4.3.

51.6.2.2.4 DETACH_STREAM.indication()

A DETACH_STREAM.indication primitive is used by the RAP Endpoint to notify the RAP Endpoint user that a Listener Attachment identified by the supplied StreamId (51.5.4.1) value has been removed.

51.6.2.3 Listener primitives

The Listener primitives defined in the following subclauses are used in interaction between a RAP Endpoint user that represents Listener applications and the underlying RAP Endpoint.

51.6.2.3.1 ATTACH_STREAM.request()

An ATTACH_STREAM.request primitive is used by the RAP Endpoint user to request the RAP Endpoint to initiate a Listener Attachment with the supplied parameter values.

The primitive has the same set of parameters as in 51.6.2.2.3.

51.6.2.3.2 DETACH_STREAM.request()

A DETACH_STREAM.request primitive is used by the RAP Endpoint user to request the RAP Endpoint to terminate a Listener Attachment identified by the supplied StreamId (51.5.4.1) value.

51.6.2.3.3 ANNOUNCE_STREAM.indication()

An ANNOUNCE_STREAM.indication primitive is used by the RAP Endpoint to notify the RAP Endpoint user that a Talker Announcement has been received with the supplied parameter values abstracted from the received Talker Announcement.

The primitive has the same set of parameters as in 51.6.2.2.1.

51.6.2.3.4 TERMINATE_STREAM.indication()

A TERMINATE_STREAM primitive is used by the RAP Endpoint to notify the RAP Endpoint user that an existing Talker Announcement identified by the supplied StreamId (51.5.3.1) value has been terminated.

51.7 RAP Participant

51.7.1 RAP Participant overview

A RAP Participant associated with a Port of an end station or Bridge is responsible for the following:

- Cooperation with the underlying LRP in managing Portal creation and association on that Port.
- Provision of attribute declaration and registration services to a RAP Endpoint or RAP Propagator (termed “RAP Participant user”).
- Maintenance of attribute declaration and registration databases.

The operation of a RAP Participant, described in terms of a RAP Participant state machine, is specified by the following:

- RAP Participant Service Interface (RPSI) in 51.7.2.
- RAP Participant state machine diagrams in 51.7.3.
- RAP Participant state machine variables in 51.7.4.
- RAP Participant state machine procedures in 51.7.5.

51.7.2 RAP Participant Service Interface (RPSI)

The RPSI provides a set of primitives (Table 51-10) for interaction between each RAP Participant and its RAP Participant user. The rPortRef (or iPortRef) parameter in a request (or indication) primitive indicates a RAP Participant to (or from) which an RPSI primitive is issued.

Table 51-10—RAP Participant Service Interface primitives

primitive name	reference
DECLARE_ATTRIBUTE.request(rPortRef, rAttr)	51.7.2.1
WITHDRAW_ATTRIBUTE.request(rPortRef, rAttr)	51.7.2.2
ATTRIBUTE_REGISTRATION.indication(iPortRef, iAttr)	51.7.2.3
ATTRIBUTE_DEREGISTRATION.indication(iPortRef, iAttr)	51.7.2.4
DEC_OPER_STATUS.indication(iPortRef, iOperStatus)	51.7.2.5

51.7.2.1 DECLARE_ATTRIBUTE.request(rPortRef, rAttr)

A DECLARE_ATTRIBUTE.request primitive is invoked by the RAP Participant user to request the RAP Participant on a Port (rPortRef) to make a declaration of the indicated attribute (rAttr) .

51.7.2.2 WITHDRAW_ATTRIBUTE.request(rPortRef, rAttr)

A WITHDRAW_ATTRIBUTE.request primitive is invoked by the RAP Participant user to request the RAP Participant on a Port (rPortRef) to withdraw an existing declaration being made for the indicated attribute (rAttr).

51.7.2.3 ATTRIBUTE_REG.indication(iPortRef, iAttr)

An ATTRIBUTE_REG.indication primitive is invoked by a RAP Participant on a Port (iPortRef) to notify the RAP Participant user that a registration of the indicated attribute (iAttr) has been made.

51.7.2.4 ATTRIBUTE_DEREG.indication(iPortRef, iAttr)

An ATTRIBUTE_DEREGISTRATION.indication primitive is used by a RAP Participant on a Port (iPortRef) to notify the RAP Participant user that a registration of the indicated attribute (iAttr) has been removed.

51.7.2.5 DEC_OPER_STATUS.indication(iPortRef, iStatus)

A DEC_OPER_STATUS.indication primitive is inovked by a RAP Participant on a Port (iPortRef) to notify the RAP Participant user of whether its attribute declaration function is operational (iStatus == TRUE) or not (iStatus == FALSE).

51.7.3 RAP Participant state machine diagrams

A RAP Participant associated with a local target Port of an end station or Bridge maintains a single RAP Participant state machine. The RAP Participant state machine diagram is shown in Figure 51-19.

51.7.4 RAP Participant state machine variables

There is one instance of the RAP Participant state machine variables per RAP Participant state machine.

51.7.4.1 portRef

A reference to the local target Port with which the RAP Participant state machine is associated.

NOTE—The portRef variable is only used inside a RAP Instance for interaction between each RAP Participant and the RAP Participant user. There is no specified relationship between portRef and localTargetPort.portId [item b) in 51.7.4.7.2].

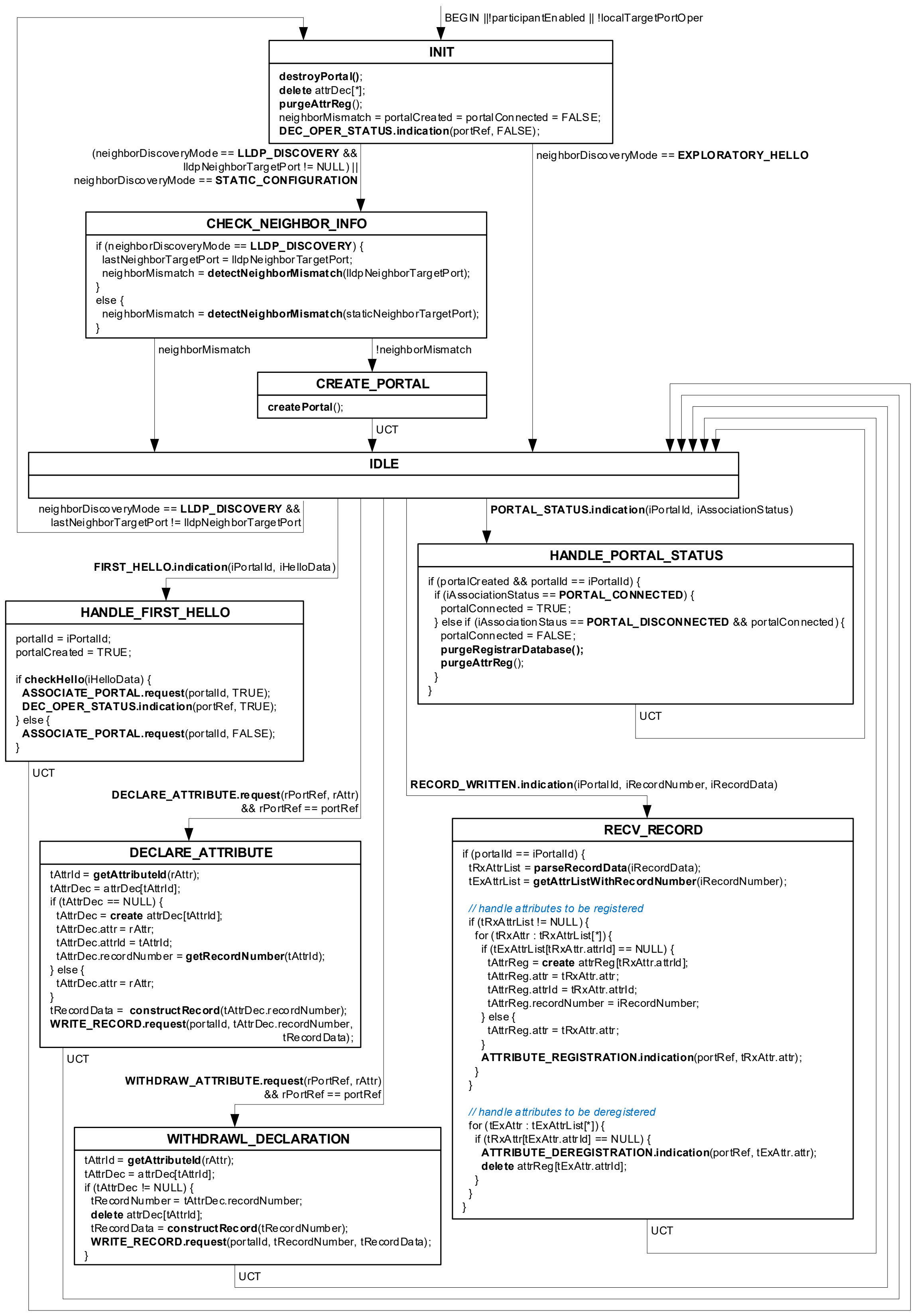
51.7.4.2 participantEnabled

A Boolean variable indicating whether the operation of the RAP Participant state machine is administratively enabled (TRUE) or not (FALSE).

51.7.4.3 neighborDiscoveryMode

An administratively assigned value, indicating the operation mode in which neighbor discovery is performed on the local target Port, and taking one of the following enumerated values:

1) **LLDP_DISCOVERY**: The information about a neighbor target port to be passed to the underlying LRP is obtained through the exchange of LRP Discovery TLVs (Annex C of IEEE Std 802.1CS-2020) by use of LLDP, and contained in lldpNeighborTargetPort (51.7.4.7.4).
2) **STATIC_CONFIGURATION**: The information about a neighbor target port to be passed to the underlying LRP is statically configured by the management and contained in staticNeighborTargetPort (51.7.4.7.3).
3) **EXPLORATORY_HELLO**: No neighbor target port information needs to be passed to the underlying LRP.


BEGIN ||!participantEnabled || !localTargetPortOper
INIT
(neighborDiscoveryMode == LLDP_DISCOVERY &&
lldpNeighborTargetPort != NULL) ||
neighborDiscoveryMode == STATIC_CONFIGURATION
neighborDiscoveryMode == EXPLORATORY_HELLO
CHECK_NEIGHBOR_INFO
neighborMismatch
!neighborMismatch
CREATE_PORTAL
UCT
IDLE
neighborDiscoveryMode == LLDP_DISCOVERY &&
lastNeighborTargetPort != lldpNeighborTargetPort
PORTAL_STATUS.indication(iPortalId, iAssociationStatus)
FIRST_HELLO.indication(iPortalId, iHelloData)
HANDLE_PORTAL_STATUS
UCT
HANDLE_FIRST_HELLO
UCT
RECORD_WRITTEN.indication(iPortalId, iRecordNumber, iRecordData)
DECLARE_ATTRIBUTE.request(rPortRef, rAttr)
&& rPortRef == portRef
RECV_RECORD
DECLARE_ATTRIBUTE
UCT
WITHDRAW_ATTRIBUTE.request(rPortRef, rAttr)
&& rPortRef == portRef
UCT
WITHDRAWL_DECLARATION
UCT

INIT
```
destroyPortal();
delete attrDec[*];
purgeAttrReg();
neighborMismatch = portalCreated = portalConnected = FALSE;
DEC_OPER_STATUS.indication(portRef, FALSE);
```

CHECK_NEIGHBOR_INFO
```
if (neighborDiscoveryMode == LLDP_DISCOVERY) {
  lastNeighborTargetPort = lldpNeighborTargetPort;
  neighborMismatch = detectNeighborMismatch(lldpNeighborTargetPort);
}
else {
  neighborMismatch = detectNeighborMismatch(staticNeighborTargetPort);
}
```

CREATE_PORTAL
```
createPortal();
```

HANDLE_PORTAL_STATUS
```
if (portalCreated && portalId == iPortalId) {
  if (iAssociationStatus == PORTAL_CONNECTED) {
    portalConnected = TRUE;
  } else if (iAssociationStaus == PORTAL_DISCONNECTED && portalConnected) {
    portalConnected = FALSE;
    purgeRegistrarDatabase();
    purgeAttrReg();
  }
}
```

HANDLE_FIRST_HELLO
```
portalId = iPortalId;
portalCreated = TRUE;

if checkHello(iHelloData) {
  ASSOCIATE_PORTAL.request(portalId, TRUE);
  DEC_OPER_STATUS.indication(portRef, TRUE);
} else {
  ASSOCIATE_PORTAL.request(portalId, FALSE);
}
```

DECLARE_ATTRIBUTE
```
tAttrId = getAttributeId(rAttr);
tAttrDec = attrDec[tAttrId];
if (tAttrDec == NULL) {
  tAttrDec = create attrDec[tAttrId];
  tAttrDec.attr = rAttr;
  tAttrDec.attrId = tAttrId;
  tAttrDec.recordNumber = getRecordNumber(tAttrId);
} else {
  tAttrDec.attr = rAttr;
}
tRecordData = constructRecord(tAttrDec.recordNumber);
WRITE_RECORD.request(portalId, tAttrDec.recordNumber,
                                          tRecordData);
```

RECV_RECORD
```
if (portalId == iPortalId) {
  tRxAttrList = parseRecordData(iRecordData);
  tExAttrList = getAttrListWithRecordNumber(iRecordNumber);

  // handle attributes to be registered
  if (tRxAttrList != NULL) {
    for (tRxAttr : tRxAttrList[*]) {
      if (tExAttrList[tRxAttr.attrId] == NULL) {
        tAttrReg = create attrReg[tRxAttr.attrId];
        tAttrReg.attr = tRxAttr.attr;
        tAttrReg.attrId = tRxAttr.attrId;
        tAttrReg.recordNumber = iRecordNumber;
      } else {
        tAttrReg.attr = tRxAttr.attr;
      }
      ATTRIBUTE_REGISTRATION.indication(portRef, tRxAttr.attr);
    }
  }

  // handle attributes to be deregistered
  for (tExAttr : tExAttrList[*]) {
    if (tRxAttr[tExAttr.attrId] == NULL) {
      ATTRIBUTE_DEREGISTRATION.indication(portRef, tExAttr.attr);
      delete attrReg[tExAttr.attrId];
    }
  }
}
```

WITHDRAWL_DECLARATION
```
tAttrId = getAttributeId(rAttr);
tAttrDec = attrDec[tAttrId];
if (tAttrDec != NULL) {
  tRecordNumber = tAttrDec.recordNumber;
  delete attrDec[tAttrId];
  tRecordData = constructRecord(tRecordNumber);
  WRITE_RECORD.request(portalId, tRecordNumber, tRecordData);
}
```

Figure 51-19—RAP Participant state machine diagram

51.7.4.4 helloTime

An administratively assigned integer value, in the range 30 through 65535, for the Hello Time parameter in a Local Target Port request [item c):1) in 10.2.2.1 of IEEE Std 802.1CS-2020] issued by the RAP Participant state machine to the underlying LRP. The default value is 30.

51.7.4.5 completeListTimerReset

An administratively assigned integer value, in the range x through y, for the cplCompleteListTimerReset parameter in a Local Target Port request [item c):3) in 10.2.2.1 of IEEE Std 802.1CS-2020] issued by the RAP Participant state machine to the underlying LRP. The default value is z.

<< Editor's note: The editor is unsure what values for x, y, z to take. Contribution is required. >>

51.7.4.6 exploreHelloRecvEnabled

An administratively assigned Boolean value for the imPplExploreRecv parameter in a Neighbor Target Port request [item b):3):iii) in 10.2.3.1 of IEEE Std 802.1CS-2020] issued by the RAP Participant state machine to the underlying LRP.

51.7.4.7 Variables of TargetPort data type

51.7.4.7.1 TargetPort data type

The TargetPort data type is a structure that consists of a collection of configuration parameters for a target port, as follows:

a) **chassisId**: The Chassis ID (8.5.2 of IEEE Std 802.1AB-2016).
b) **portId**: The Port ID (8.5.3 of IEEE Std 802.1AB-2016).
c) **ecpCapable**: A Boolean value indicating whether the target port supports the LRP-DT ECP mechanism (TRUE) or not (FALSE).
d) **tcpCapable**: A Boolean value indicating whether that the target port supports the LRP-DT TCP mechanism (TRUE) or not (FALSE).
e) **tcpPort:** A 2-byte TCP port number for the target port (C.2.2.6.1 of IEEE Std 802.1CS-2020).
f) **addrIPv4**: A 4-byte IPv4 address for the target port [item 1) in C.2.2.6.2 of IEEE Std 802.1CS-2020] or NULL.
g) **addrIPv6:** A 16-byte IPv6 address for the target port [item 2) in C.2.2.6.2 of IEEE Std 802.1CS-2020] or NULL.

51.7.4.7.2 localTargetPort

The localTargetPort variable is a structure of the TargetPort data type (51.7.4.7.1). It contains the administratively configured parameters of the local target port.

51.7.4.7.3 staticNeighorTargetPort

The staticNeighorTargetPort variable is a structure of the TargetPort data type (51.7.4.7.1). It contains the administratively configured parameters of a neighbor target port to which the local target port is to be connected.

51.7.4.7.4 lldpNeighborTargetPort

The lldpNeighborTargetPort variable is a structure of the TargetPort data type (51.7.4.7.1). It contains the configuration parameters of a neighbor target port discovered by LLDP.

The values contained in this variable are derived from a matching LRP ECP Discovery TLV and/or a matching LRP TCP Discovery TLV currently stored in the LLDP remote systems MIB on the local target port and meeting the following two conditions:

— received by an LLDP agent using a MAC address as specified in 51.4.2.
— containing an Application descriptor (C.2.1.6 and C.2.2.6 of IEEE Std 802.1CS-2020) with an AppId value as specified in Table 51-6.

In the presence of the matching TLV(s), each member variable of lldpNeighborTargetPort is set as follows:

a) **chassisId**: Set to the Chassis ID value (8.5.2 of IEEE Std 802.1AB-2016) of the received LLDPDUs that carry the matching LRP Discovery TLV(s).
b) **portId**: Set to the Port ID value (8.5.3 of IEEE Std 802.1AB-2016) of the received LLDPDUs that carry the LRP Discovery TLV(s).
c) **ecpCapable**: Set TRUE if a matching LRP ECP Discovery TLV is present, indicating that the neighbor Port supports the LRP-DT ECP mechanism, and set FALSE otherwise.
d) **tcpCapable**: Set TRUE if a matching LRP TCP Discovery TLV is present, indicating that the neighbor Port supports the LRP-DT TCP mechanism, and set FALSE otherwise.
e) **tcpPort**: If tcpCapable in item d) is TRUE, set to the value contained in the TCP port number field (C.2.2.6.1 of IEEE Std 802.1CS-2020) of the matching Application descriptor in the received LRP TCP Discovery TLV.
f) **addrIPv4**: If tcpCapable in item d) is TRUE and the matching Application descriptor in the received LRP TCP Discovery TLV indicates an IPv4 address, set to the corresponding value encoded in the Address info field (C.2.2.6.2 of IEEE Std 802.1CS-2020); otherwise, set to NULL.
g) **addrIPv6:** If tcpCapable in item d) is TRUE and the matching Application descriptor in the received LRP TCP Discovery TLV indicates an IPv6 address, set to the corresponding value encoded in the Address info field (C.2.2.6.2 of IEEE Std 802.1CS-2020); otherwise, set to NULL.

If no matching LRP Discovery TLV(s) is present, the lldpNeighborTargetPort variable set to NULL, indicating that there is currently no neighbor discovered by LLDP.

NOTE—This standard does not specify the means for keeping lldpNeighborTargetPort up to date with the information maintained by LLDP. A Native system could run a local procedure to access the LLDP MIBs residing within the same system. A Proxy system with a RAP Participant that has LRP discovery TLVs exchanged on a Controlled system's target port could update this variable by observing the contents of the lrpLldpEcpTlvRecvInfo and lrpLldpTcpTlvRecvInfo managed objects (11.6.2.3 and 11.6.2.6 of IEEE Std 802.1CS-2020, respectively) of that Controlled system via management interfaces.

51.7.4.7.5 lastNeighborTargetPort

The lastNeighborTargetPort variable is a structure of the TargetPort data type (51.7.4.7.1). It is used to contain the values copied from lldpNeighborTargetPort (51.7.4.7.4) since the latter was last updated.

51.7.4.8 neighborMismatch

A Boolean variable, set TRUE when detecting a mismatch between the local target port and the neighbor target port to be connected.

NOTE—The purpose of targetPortMismatch is to supply diagnostic information, through the associated managed object, for the management to take appropriate actions, if necessary, when its value is TRUE.

51.7.4.9 localTargetPortOper

A Boolean variable indicating the status of the Internal Sublayer Service MAC_Operational parameter (11.2 of IEEE Std 802.1AC-2016) associated with the local target port.

NOTE—If the Continuity Check protocol (20.1) is deployed to detect physical link connectivity on the local target port, the MEP Continuity Check Receiver (19.2.8) will cause that port's MAC_Operational parameter to be false upon detecting a connectivity fault.

51.7.4.10 portalCreated

A Boolean value indicating whether a Portal for the operation of the RAP Participant on the local Port has been created by the underlying LRP (TRUE) or not (FALSE).

51.7.4.11 portalId

The identifier of a Portal created by the underlying LRP for the local target port, set to the corresponding parameter value contained in a FIRST_HELLO.indication (Table 51-7) received from the underlying LRP.

51.7.4.12 portalConnected

A Boolean value indicating whether a Portal association for the Portal as indicated in portalId (51.7.4.11) has been established by the underlying LRP (TRUE) or not (FALSE).

51.7.4.13 attrDec

The attrDec variable is an array in which each element represents an attribute declaration maintained by the RAP Participant and consists of the following member variables:

a) **attr**: An attribute instance as specified in 51.5.
b) **attrId**: An integer identifier generated by the getAttributeId procedure (51.7.5.5) for the attribute instance contained in item a).
c) **recordNumber**: The record number of a record in which the attribute instance contained in item a) is carried.

The attrDec variable is associated with a key <attrId>.

51.7.4.14 attrReg

The attrReg variable is an array in which each element represents an attribute registration maintained by the RAP Participant and consists of the following member variables:

a) **attr**: An attribute instance as specified in 51.5.
b) **attrId**: An integer identifier generated by the getAttributeId procedure (51.7.5.5) for the attribute instance contained in item a).
c) **recordNumber**: The record number of a record in which the attribute instance contained in item a) is carried.

The attrReg variable is associated with a key <attrId>.

51.7.5 RAP Participant state machine procedures

51.7.5.1 detectNeighborMismatch()

This procedure determines whether there is a mismatch between the local target port and the neighbor target port to be connected, as follows:

```
detectNeighborMismatch(pNeighborTargetPort) {
  if ((pNeighborTargetPort.ecpCapable && localTargetPort.ecpCapable) ||
      (pNeighborTargetPort.tcpCapable && localTargetPort.tcpCapable &&
       ((pNeighborTargetPort.addrIPv4 != NULL &&
         localTargetPort.addrIPv4!= NULL) ||
        (pNeighborTargetPort.addrIPv6 != NULL &&
         localTargetPort.addrIPv6 != NULL)
       )
      )
     ){
    return FALSE;
  } else {
    return TRUE;
  }
```

51.7.5.2 createPortal()

This procedure requests the underlying LRP to create a Portal for the RAP Participant, by issuing a **LOCAL_TARGET_PORT.request** primitive and a **NEIGHBOR_TARGET_PORT.request** primitive, with the parameters values shown in Table 51-11.

Table 51-11—Input parameter values for Portal Creation

		neighborDiscoveryMode		
		LLDP_DISCOVERY	STATIC_CONFIGURATION	EXPLORATORY_ HELLO
Local Target Port request input parameters (item in 10.2.2.1 of IEEE Std 802.1CS-2020)	a).1)	the AppId value specified in Table 51-6		
	b).1)	localTargetPort.chassisId		
	b).2)	localTargetPort.portId		
	b).3)	localTargetPort.ecpCapable		
	b).4).i)	localTargetPort.addrIPv4		
	b).4).ii)	localTargetPort.addrIPv6		
	b).4).iii)	localTargetPort.tcpPort		
	c).1)	helloTime		
	c).2)	the value specified in 51.4.4		
	c).3)	completeListTimerReset		
Neighbor Target Port request input parameters (item in 10.2.3.1 of IEEE Std 802.1CS-2020)	a).1)	a 3-tuple of {AppId, localTargetPort.chassisId, localTargetPort.portId}		
	b).1)	lldpNeighborTargetPort.chassisId	staticNeighborTargetPort.chassisId	none
	b).2)	lldpNeighborTargetPort.portId	staticNeighborTargetPort.portId	none
	b).3).i)	if lldpNeighborTargetPort.ecpCapable == TRUE, set to the ECP MAC address as specified in 51.4.3.1, otherwise NULL	if staticNeighborTargetPort.ecpCapable == TRUE, set to the ECP MAC address as specified in 51.4.3.1, otherwise NULL	the ECP MAC address as specified in 51.4.3.1
	b).3).ii)	FALSE		TRUE
	b).3).iii)	exploreHelloRecvEnabled		TRUE
	b).4).i)	lldpNeighborTargetPort.addrIPv4	staticNeighborTargetPort.addrIPv4	none
	b).4).ii)	lldpNeighborTargetPort.addrIPv6	staticNeighborTargetPort.addrIPv6	none
	b).4).iii)	lldpNeighborTargetPort.tcpPort	staticNeighborTargetPort.tcpPort	none

51.7.5.3 checkHello(pHelloData)

This procedure checks the contents of a Hello LRPDU (pHelloData) passed from the underlying LRP to the RAP Participant state machine, to determine whether to approve a Portal association, as follows:

a) If pHelloData contains an Application Information TLV whose value field is encoded as specified in 51.4.4, approve the association by returning a value of TRUE.
b) Otherwise, deny the association by returning a value of FALSE.

51.7.5.4 destroyPortal()

This procedure is used by the RAP Participant state machine to request the underlying LRP to destroy the Portal previously created (if any), by issuing a **LOCAL_TARGET_PORT.request** (Table 51-7) with the input parameters (items defined in 10.2.2.1 of IEEE Std 802.1CS-2022) set as follows:

a) item a).1) set to the AppId value defined in Table 51-6.
b) item b).1) set to localTargetPort.chassisId.
c) item b).2) set to localTargetPort.portId.
d) item b).3) set to FALSE.
e) other items left empty.

51.7.5.5 getAttributeId(pAttr)

This procedure generates and returns an integer identifier for an attribute instance (pAttr) to be declared or registered on the local target port. Each attribute declaration or registration maintained by the RAP Participant state machine is identified by a subset of the parameters encoded in the attribute TLV, as specified in Table 51-12.

Table 51-12—Attribute identification parameters in a RAP Participant

TLV name (value in Type field from Table 51-8)	Parameters in Value field of the attribute TLV	
RA attribute TLV (0x00)	N/A	
Talker Announce attribute TLV (0x01)	StreamID 51.5.3.1	VID 51.5.3.5.3
Listener Attach attribute TLV (0x02)	StreamID 51.5.4.1	VID 51.5.4.2

NOTE—An attribute identifier generated by this procedure is only of local significance to and used within a particular RAP Participant state machine.

51.7.5.6 getRecordNumber(pAttrId)

This procedure allocates and returns a 4-octet record number (as specified in Table 9-7 of IEEE Std 802.1CS-2020), for a new attribute declaration identified by pAttrId.

The assigned record number could be a new one that is not used by any of the other entries in attrDec, or an existing one that is being used by one or more of the other entries in attrDec. Assignment of a record number to multiple declarations, which causes the attribute instances of these declarations to be serialized into a single record, shall not cause the record to exceed the maximum record size determined by the underlying LRP.

The strategies deployed by this procedure for assignment of record numbers are not specified by this standard.

NOTE—According to the operation of LRP, record number assignment is performed by an LRP application on the Applicant size of a Portal connection. Within a Portal, there is no relationship between a record in an applicant database and a record with the same record number in the registrar database. Thus, there is no requirement for an LRP application to use the same record number assignment strategies on both sides of a Portal connection.

51.7.5.7 constructRecord(pRecordNumber)

This procedure searches in attrDec for all the elements whose recordNumber value matches the pRecordNumber value, and returns an octet string in one of the following cases:

a) No matching element found: a zero length octet string.
b) Only one matching element: an octet string holding the attr value of the matching attrDec element.
c) More than one matching element: an octet string holding the concatenation of the attr values of all the matching attrDec elements in any order.

51.7.5.8 parseRecordData(pRecordData)

This procedure parses the data (pRecordData) received in a record from the underlying LRP into RAP attributes in accordance with the attribute TLV formats defined in 51.5.

If one or more attribute instances are parsed from the record, the procedures returns an array in which each element corresponds to an attribute instance and consists of the following member variables:

a) **attr**: An attribute instance parsed from the given record.
b) **attrId**: The return value of invoking the getAttributeId procedure (51.7.5.5) with the attribute contained in a).

Otherwise, the procedure returns a NULL value.

51.7.5.9 getAttrListWithRecordNumber(pRecordNumber)

This procedure searches in the attrReg variable for all the elements whose recordNumber value matches the pRecordNumber value. It returns an array in which each element corresponds to a matching element in attrReg and consists of the following member variables:

a) **attr**: copied from the attr value of a matching element in attrReg.
b) **attrId**: copied from the attrId value of a matching element in attrReg.

51.7.5.10 purgeRegistrarDatabase()

This procedure is used by the RAP Participant state machine to request the underlying to purges all records in the Portal’s registrar database, as follows:

```
purgeRegistrarDatabase() {
  for (tAttrReg : attrReg[*]) {
    DELETE_RECORD.request(portalId, tAttrReg.recordNumber);
  }
}
```

51.7.5.11 purgeAttrReg()

This procedure purges all attribute registrations contained in attrReg, as follows:

```
purgeAttrReg() {
  for (tAttrReg : attrReg[*]) {
    ATTRIBUTE_DEREGISTRATION.indication(portRef, tAttrReg.attr);
    delete attrReg[tAttrReg.attrId];
  }
}
```

51.8 RAP Propagator

51.8.1 RAP Propagator overview

The operation of a RAP Propagator, described in terms of a per-Bridge RAP Propagator state machine, is specified by the following:

— RAP Propagator state machine diagrams in 51.8.2.
— RAP Propagator state machine events in 51.8.3.
— RAP Propagator state machine variables in 51.8.4.
— RAP Propagator state machine procedures in 51.8.5.

51.8.2 RAP Propagator state machine diagrams

The operation of the RAP Propagator state machine is specified by the state machine diagram in Figure 51-20. The actions executed on entry to each state of the state machine (except for the IDLE state in which no actions are defined) are shown in Figure 51-21, Figure 51-22, Figure 51-23, Figure 51-24, Figure 51-25 and Figure 51-26.

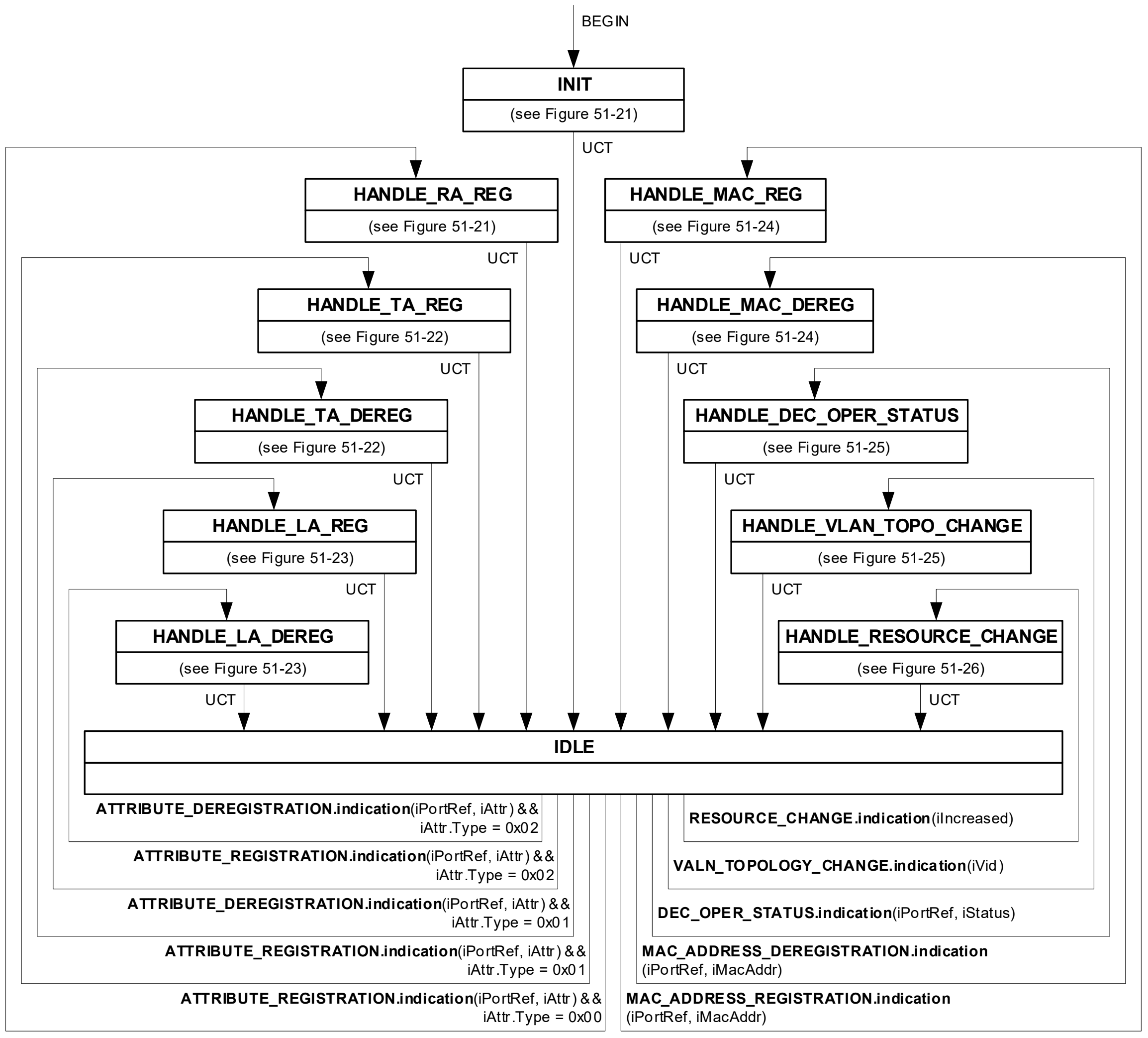

BEGIN
INIT
(see Figure 51-21)
UCT
HANDLE_RA_REG
(see Figure 51-21)
HANDLE_MAC_REG
(see Figure 51-24)
HANDLE_TA_REG
(see Figure 51-22)
HANDLE_MAC_DEREG
(see Figure 51-24)
HANDLE_TA_DEREG
(see Figure 51-22)
HANDLE_DEC_OPER_STATUS
(see Figure 51-25)
HANDLE_LA_REG
(see Figure 51-23)
HANDLE_VLAN_TOPO_CHANGE
(see Figure 51-25)
HANDLE_LA_DEREG
(see Figure 51-23)
HANDLE_RESOURCE_CHANGE
(see Figure 51-26)
IDLE
ATTRIBUTE_DEREGISTRATION.indication(iPortRef, iAttr) && iAttr.Type = 0x02
ATTRIBUTE_REGISTRATION.indication(iPortRef, iAttr) && iAttr.Type = 0x02
ATTRIBUTE_DEREGISTRATION.indication(iPortRef, iAttr) && iAttr.Type = 0x01
ATTRIBUTE_REGISTRATION.indication(iPortRef, iAttr) && iAttr.Type = 0x01
ATTRIBUTE_REGISTRATION.indication(iPortRef, iAttr) && iAttr.Type = 0x00
RESOURCE_CHANGE.indication(iIncreased)
VALN_TOPOLOGY_CHANGE.indication(iVid)
DEC_OPER_STATUS.indication(iPortRef, iStatus)
MAC_ADDRESS_DEREGISTRATION.indication (iPortRef, iMacAddr)
MAC_ADDRESS_REGISTRATION.indication (iPortRef, iMacAddr)

Figure 51-20—RAP Propagator state machine diagram

Figure 51-21 defines the actions in the INIT state for initialization of the RAP Propagator state machine, and the actions in the HANDLE_RA_REG state for handling of an ATTRIBUTE_REGISTRATION.indication (51.7.2.3) that indicates an RA registration.

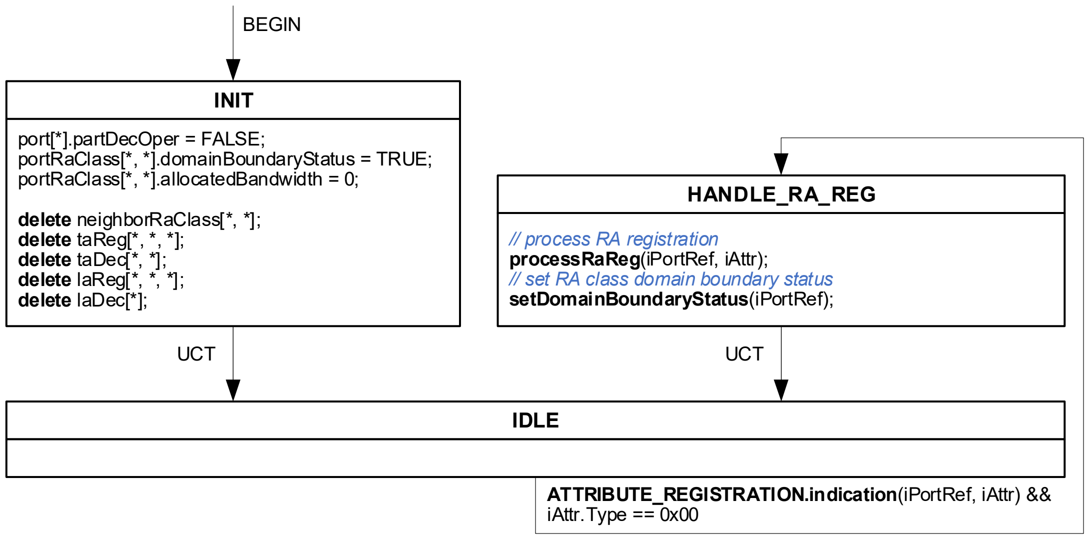
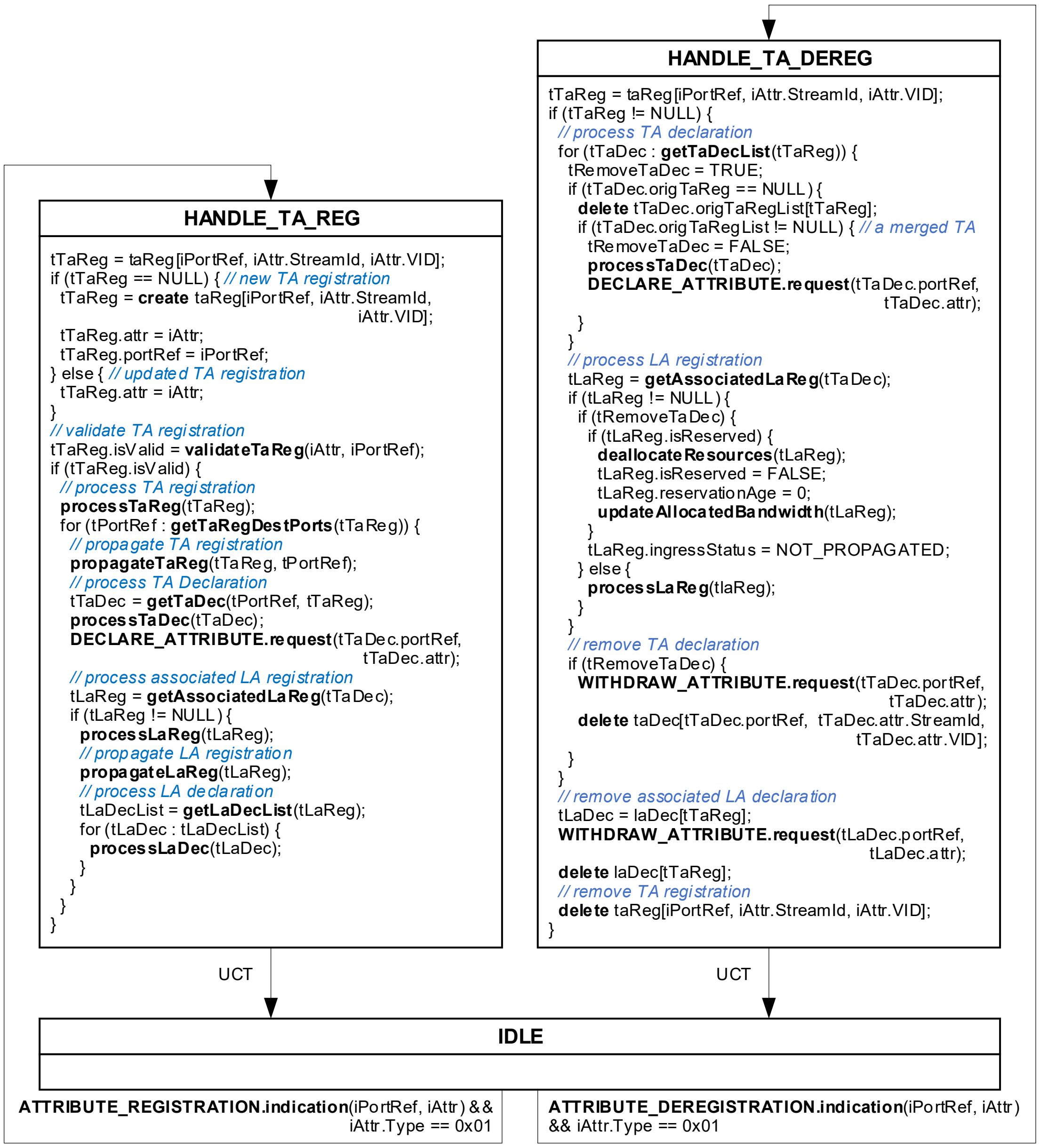

BEGIN
INIT
port[*].partDecOper = FALSE;
portRaClass[*, *].domainBoundaryStatus = TRUE;
portRaClass[*, *].allocatedBandwidth = 0;
delete neighborRaClass[*, *];
delete taReg[*, *, *];
delete taDec[*, *];
delete laReg[*, *, *];
delete laDec[*];
HANDLE_RA_REG
// process RA registration
processRaReg(iPortRef, iAttr);
// set RA class domain boundary status
setDomainBoundaryStatus(iPortRef);
UCT
UCT
IDLE
ATTRIBUTE_REGISTRATION.indication(iPortRef, iAttr) &&
iAttr.Type == 0x00

Figure 51-21—Initialization and handling of RA registrations in a RAP Propagator

Figure 51-22 defines the actions in the HANDLE_TA_REG state for handling of an ATTRIBUTE_REGISTRATION.indication (51.7.2.3) that indicates a Talker Announce registration, and the actions in the HANDLE_TA_DEREG state for handling of an ATTRIBUTE_DEREGISTRATION.indication (51.7.2.4) that indicates a Taker Announce deregistration.



HANDLE_TA_REG:

```
tTaReg = taReg[iPortRef, iAttr.StreamId, iAttr.VID];
if (tTaReg == NULL) { // new TA registration
  tTaReg = create taReg[iPortRef, iAttr.StreamId,
                                   iAttr.VID];
  tTaReg.attr = iAttr;
  tTaReg.portRef = iPortRef;
} else { // updated TA registration
  tTaReg.attr = iAttr;
}
// validate TA registration
tTaReg.isValid = validateTaReg(iAttr, iPortRef);
if (tTaReg.isValid) {
  // process TA registration
  processTaReg(tTaReg);
  for (tPortRef : getTaRegDestPorts(tTaReg)) {
    // propagate TA registration
    propagateTaReg(tTaReg, tPortRef);
    // process TA Declaration
    tTaDec = getTaDec(tPortRef, tTaReg);
    processTaDec(tTaDec);
    DECLARE_ATTRIBUTE.request(tTaDec.portRef,
                              tTaDec.attr);
    // process associated LA registration
    tLaReg = getAssociatedLaReg(tTaDec);
    if (tLaReg != NULL) {
      processLaReg(tLaReg);
      // propagate LA registration
      propagateLaReg(tLaReg);
      // process LA declaration
      tLaDecList = getLaDecList(tLaReg);
      for (tLaDec : tLaDecList) {
        processLaDec(tLaDec);
      }
    }
  }
}
```

HANDLE_TA_DEREG:

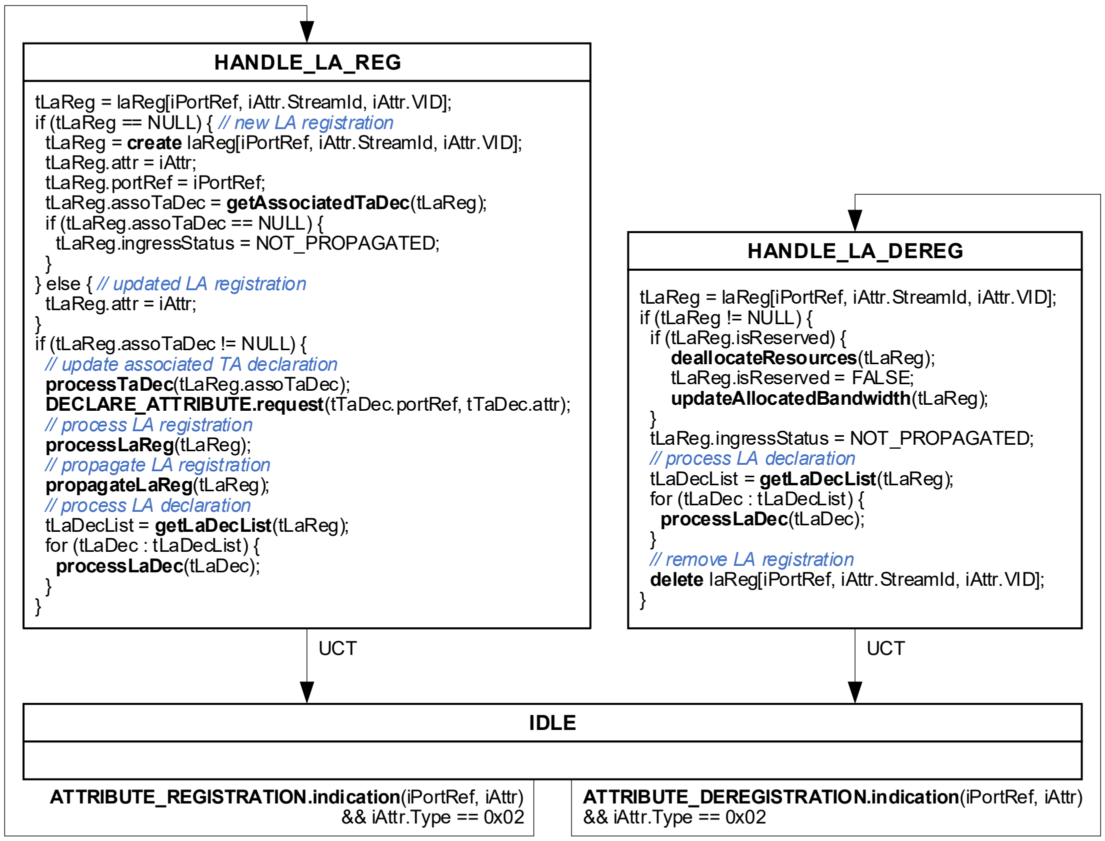
```
tTaReg = taReg[iPortRef, iAttr.StreamId, iAttr.VID];
if (tTaReg != NULL) {
 // process TA declaration
 for (tTaDec : getTaDecList(tTaReg)) {
   tRemoveTaDec = TRUE;
   if (tTaDec.origTaReg == NULL) {
    delete tTaDec.origTaRegList[tTaReg];
    if (tTaDec.origTaRegList != NULL) { // a merged TA
      tRemoveTaDec = FALSE;
      processTaDec(tTaDec);
      DECLARE_ATTRIBUTE.request(tTaDec.portRef,
                                tTaDec.attr);
    }
   }
   // process LA registration
   tLaReg = getAssociatedLaReg(tTaDec);
   if (tLaReg != NULL) {
    if (tRemoveTaDec) {
      if (tLaReg.isReserved) {
        deallocateResources(tLaReg);
        tLaReg.isReserved = FALSE;
        tLaReg.reservationAge = 0;
        updateAllocatedBandwidth(tLaReg);
      }
      tLaReg.ingressStatus = NOT_PROPAGATED;
    } else {
      processLaReg(tlaReg);
    }
   }
   // remove TA declaration
   if (tRemoveTaDec) {
    WITHDRAW_ATTRIBUTE.request(tTaDec.portRef,
                               tTaDec.attr);
    delete taDec[tTaDec.portRef, tTaDec.attr.StreamId,
                                 tTaDec.attr.VID];
   }
 }
 // remove associated LA declaration
 tLaDec = laDec[tTaReg];
 WITHDRAW_ATTRIBUTE.request(tLaDec.portRef,
                            tLaDec.attr);
 delete laDec[tTaReg];
 // remove TA registration
 delete taReg[iPortRef, iAttr.StreamId, iAttr.VID];
}
```

```
ATTRIBUTE_REGISTRATION.indication(iPortRef, iAttr) &&
                                   iAttr.Type == 0x01
```

```
ATTRIBUTE_DEREGISTRATION.indication(iPortRef, iAttr)
&& iAttr.Type == 0x01
```

Figure 51-22—Handling of Talker Announce registrations and deregistrations in a RAP Propagator

Figure 51-23 defines the actions in the HANDLE_LA_REG state for handling of an ATTRIBUTE_REGISTRATION.indication (51.7.2.3) that indicates a Listener Attach registration, and the actions in the HANDLE_LA_DEREG state for handling of an ATTRIBUTE_DEREGISTRATION.indication (51.7.2.4) that indicates a Listener Attach deregistration.



```
HANDLE_LA_REG

tLaReg = laReg[iPortRef, iAttr.StreamId, iAttr.VID];
if (tLaReg == NULL) { // new LA registration
  tLaReg = create laReg[iPortRef, iAttr.StreamId, iAttr.VID];
  tLaReg.attr = iAttr;
  tLaReg.portRef = iPortRef;
  tLaReg.assoTaDec = getAssociatedTaDec(tLaReg);
  if (tLaReg.assoTaDec == NULL) {
    tLaReg.ingressStatus = NOT_PROPAGATED;
  }
} else { // updated LA registration
  tLaReg.attr = iAttr;
}
if (tLaReg.assoTaDec != NULL) {
  // update associated TA declaration
  processTaDec(tLaReg.assoTaDec);
  DECLARE_ATTRIBUTE.request(tTaDec.portRef, tTaDec.attr);
  // process LA registration
  processLaReg(tLaReg);
  // propagate LA registration
  propagateLaReg(tLaReg);
  // process LA declaration
  tLaDecList = getLaDecList(tLaReg);
  for (tLaDec : tLaDecList) {
    processLaDec(tLaDec);
  }
}
```

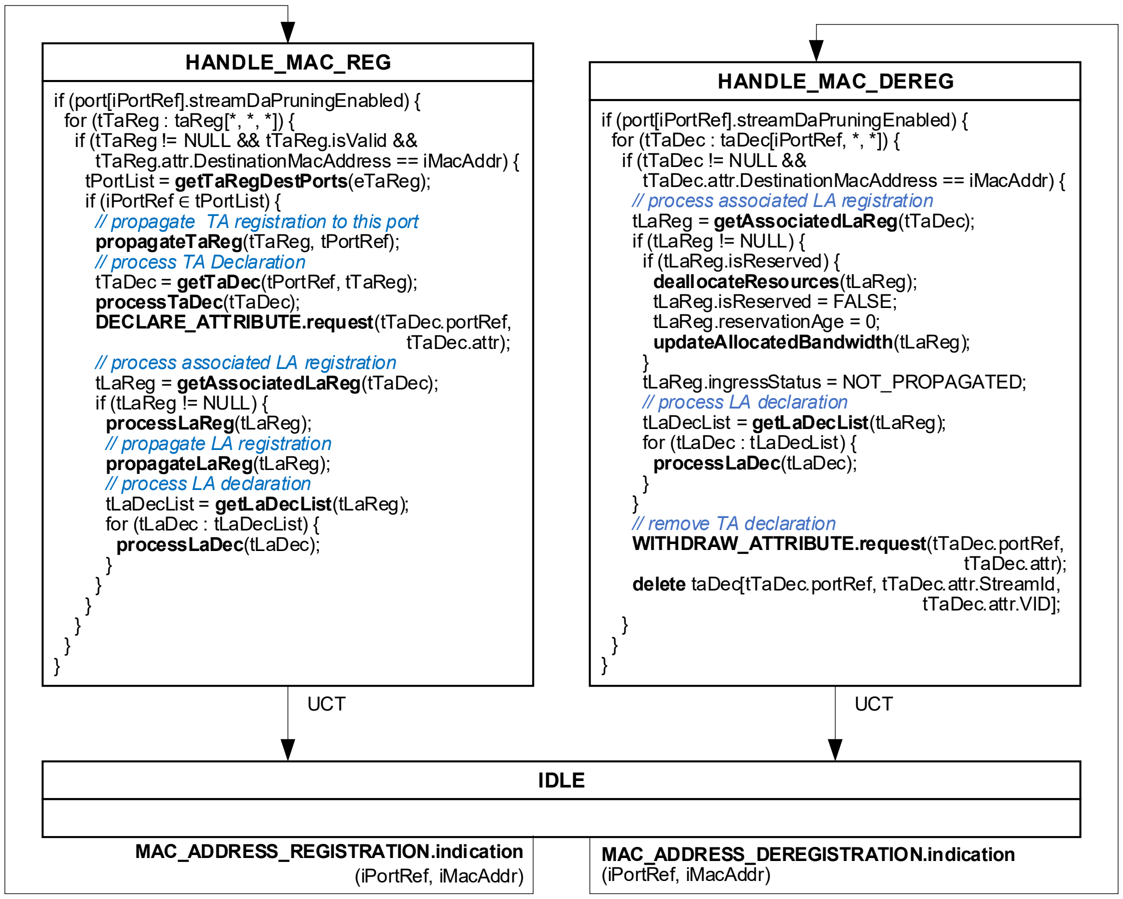
```
HANDLE_LA_DEREG

tLaReg = laReg[iPortRef, iAttr.StreamId, iAttr.VID];
if (tLaReg != NULL) {
  if (tLaReg.isReserved) {
      deallocateResources(tLaReg);
      tLaReg.isReserved = FALSE;
      updateAllocatedBandwidth(tLaReg);
  }
  tLaReg.ingressStatus = NOT_PROPAGATED;
  // process LA declaration
  tLaDecList = getLaDecList(tLaReg);
  for (tLaDec : tLaDecList) {
    processLaDec(tLaDec);
  }
  // remove LA registration
  delete laReg[iPortRef, iAttr.StreamId, iAttr.VID];
}
```

ATTRIBUTE_REGISTRATION.indication(iPortRef, iAttr) && iAttr.Type == 0x02

ATTRIBUTE_DEREGISTRATION.indication(iPortRef, iAttr) && iAttr.Type == 0x02

Figure 51-23—Handling of Listener Attach registrations and deregistrations in a RAP Propagator

Figure 51-24 defines the actions in the HANDLE_MAC_REG state for processing of a MAC_ADDRESS_REGISTRATION.indication (51.8.3.2), and the actions in the HANDLE_MAC_DEREG state for processing of a MAC_ADDRESS_DEREGISTRATION.indication (51.8.3.3).

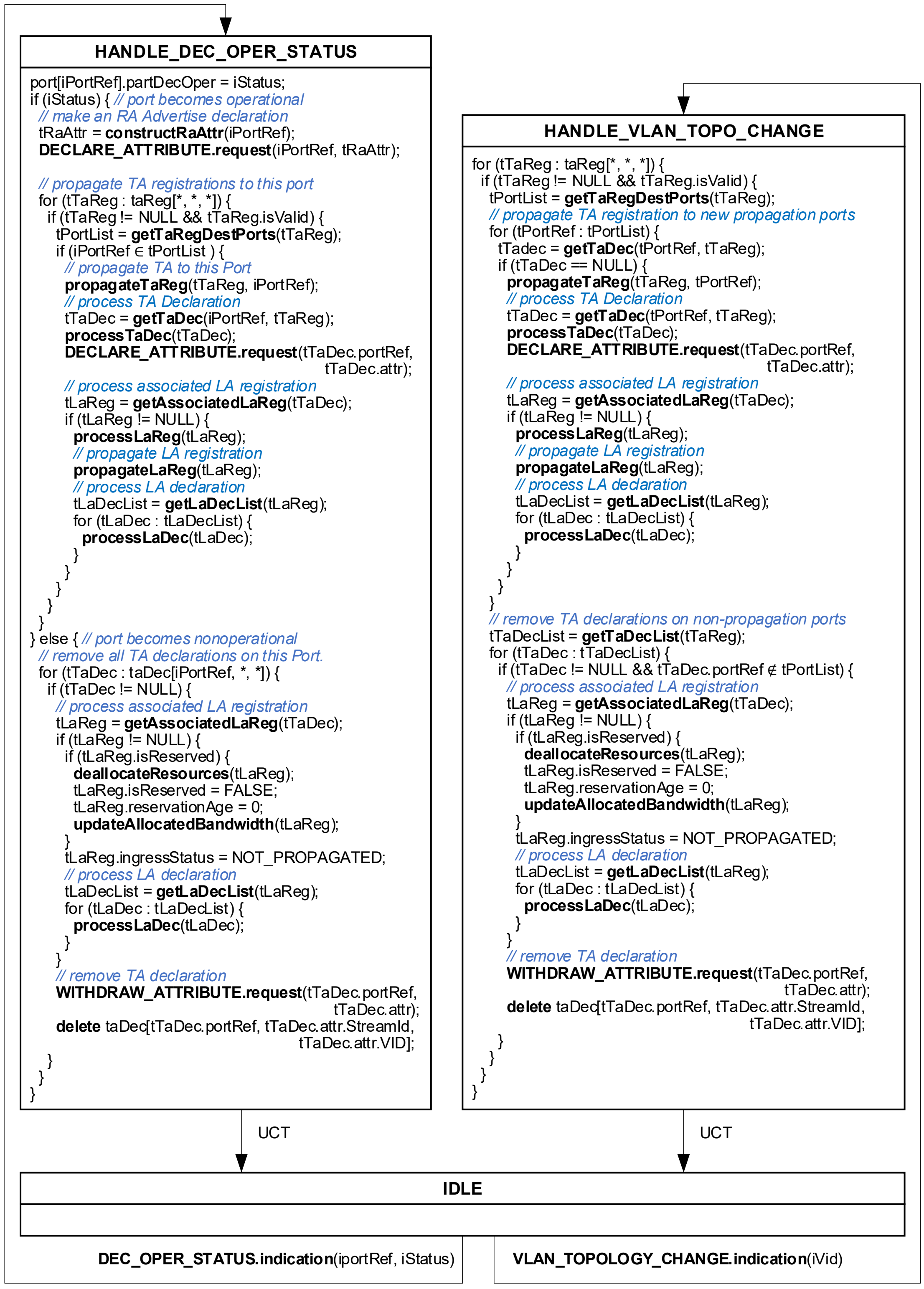

HANDLE_MAC_REG
HANDLE_MAC_DEREG
UCT
UCT
IDLE
MAC_ADDRESS_REGISTRATION.indication (iPortRef, iMacAddr)
MAC_ADDRESS_DEREGISTRATION.indication (iPortRef, iMacAddr)

```
if (port[iPortRef].streamDaPruningEnabled) {
  for (tTaReg : taReg[*, *, *]) {
    if (tTaReg != NULL && tTaReg.isValid &&
        tTaReg.attr.DestinationMacAddress == iMacAddr) {
      tPortList = getTaRegDestPorts(eTaReg);
      if (iPortRef ∈ tPortList) {
        // propagate  TA registration to this port
        propagateTaReg(tTaReg, tPortRef);
        // process TA Declaration
        tTaDec = getTaDec(tPortRef, tTaReg);
        processTaDec(tTaDec);
        DECLARE_ATTRIBUTE.request(tTaDec.portRef,
                                  tTaDec.attr);
        // process associated LA registration
        tLaReg = getAssociatedLaReg(tTaDec);
        if (tLaReg != NULL) {
          processLaReg(tLaReg);
          // propagate LA registration
          propagateLaReg(tLaReg);
          // process LA declaration
          tLaDecList = getLaDecList(tLaReg);
          for (tLaDec : tLaDecList) {
            processLaDec(tLaDec);
          }
        }
      }
    }
  }
}
```

```
if (port[iPortRef].streamDaPruningEnabled) {
  for (tTaDec : taDec[iPortRef, *, *]) {
    if (tTaDec != NULL &&
        tTaDec.attr.DestinationMacAddress == iMacAddr) {
      // process associated LA registration
      tLaReg = getAssociatedLaReg(tTaDec);
      if (tLaReg != NULL) {
        if (tLaReg.isReserved) {
          deallocateResources(tLaReg);
          tLaReg.isReserved = FALSE;
          tLaReg.reservationAge = 0;
          updateAllocatedBandwidth(tLaReg);
        }
        tLaReg.ingressStatus = NOT_PROPAGATED;
        // process LA declaration
        tLaDecList = getLaDecList(tLaReg);
        for (tLaDec : tLaDecList) {
          processLaDec(tLaDec);
        }
      }
      // remove TA declaration
      WITHDRAW_ATTRIBUTE.request(tTaDec.portRef,
                                 tTaDec.attr);
      delete taDec[tTaDec.portRef, tTaDec.attr.StreamId,
                   tTaDec.attr.VID];
    }
  }
}
```

Figure 51-24—Handling of MAC address registrations and deregistrations in a RAP Propagator

Figure 51-25 defines the actions in the HANDLE_DEC_OPER_STATUS state for handling of a DEC_OPER_STATUS.indication (51.7.2.5), and the actions in the HANDLE_VLAN_TOPO_CHANGE state for handling of a VLAN_TOPOLOGY_CHANGE.indication (51.8.3.1).


HANDLE_DEC_OPER_STATUS
HANDLE_VLAN_TOPO_CHANGE
UCT
UCT
IDLE
DEC_OPER_STATUS.indication(iportRef, iStatus)
VLAN_TOPOLOGY_CHANGE.indication(iVid)

HANDLE_DEC_OPER_STATUS

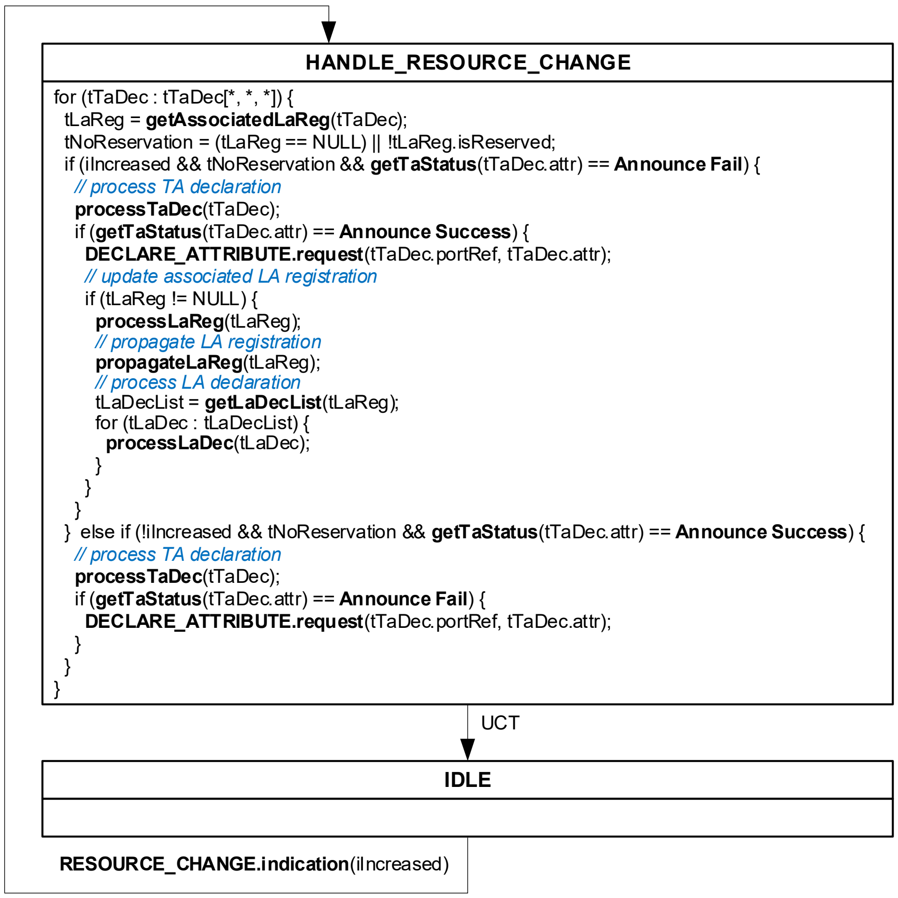
```
port[iPortRef].partDecOper = iStatus;
if (iStatus) { // port becomes operational
  // make an RA Advertise declaration
  tRaAttr = constructRaAttr(iPortRef);
  DECLARE_ATTRIBUTE.request(iPortRef, tRaAttr);

  // propagate TA registrations to this port
  for (tTaReg : taReg[*, *, *]) {
    if (tTaReg != NULL && tTaReg.isValid) {
      tPortList = getTaRegDestPorts(tTaReg);
      if (iPortRef ∈ tPortList ) {
        // propagate TA to this Port
        propagateTaReg(tTaReg, iPortRef);
        // process TA Declaration
        tTaDec = getTaDec(iPortRef, tTaReg);
        processTaDec(tTaDec);
        DECLARE_ATTRIBUTE.request(tTaDec.portRef,
                                   tTaDec.attr);
        // process associated LA registration
        tLaReg = getAssociatedLaReg(tTaDec);
        if (tLaReg != NULL) {
          processLaReg(tLaReg);
          // propagate LA registration
          propagateLaReg(tLaReg);
          // process LA declaration
          tLaDecList = getLaDecList(tLaReg);
          for (tLaDec : tLaDecList) {
            processLaDec(tLaDec);
          }
        }
      }
    }
  }
} else { // port becomes nonoperational
  // remove all TA declarations on this Port.
  for (tTaDec : taDec[iPortRef, *, *]) {
    if (tTaDec != NULL) {
      // process associated LA registration
      tLaReg = getAssociatedLaReg(tTaDec);
      if (tLaReg != NULL) {
        if (tLaReg.isReserved) {
          deallocateResources(tLaReg);
          tLaReg.isReserved = FALSE;
          tLaReg.reservationAge = 0;
          updateAllocatedBandwidth(tLaReg);
        }
        tLaReg.ingressStatus = NOT_PROPAGATED;
        // process LA declaration
        tLaDecList = getLaDecList(tLaReg);
        for (tLaDec : tLaDecList) {
          processLaDec(tLaDec);
        }
      }
      // remove TA declaration
      WITHDRAW_ATTRIBUTE.request(tTaDec.portRef,
                                  tTaDec.attr);
      delete taDec[tTaDec.portRef, tTaDec.attr.StreamId,
                                   tTaDec.attr.VID];
    }
  }
}
```

HANDLE_VLAN_TOPO_CHANGE

```
for (tTaReg : taReg[*, *, *]) {
  if (tTaReg != NULL && tTaReg.isValid) {
    tPortList = getTaRegDestPorts(tTaReg);
    // propagate TA registration to new propagation ports
    for (tPortRef : tPortList) {
      tTadec = getTaDec(tPortRef, tTaReg);
      if (tTaDec == NULL) {
        propagateTaReg(tTaReg, tPortRef);
        // process TA Declaration
        tTaDec = getTaDec(tPortRef, tTaReg);
        processTaDec(tTaDec);
        DECLARE_ATTRIBUTE.request(tTaDec.portRef,
                                   tTaDec.attr);
        // process associated LA registration
        tLaReg = getAssociatedLaReg(tTaDec);
        if (tLaReg != NULL) {
          processLaReg(tLaReg);
          // propagate LA registration
          propagateLaReg(tLaReg);
          // process LA declaration
          tLaDecList = getLaDecList(tLaReg);
          for (tLaDec : tLaDecList) {
            processLaDec(tLaDec);
          }
        }
      }
    }
    // remove TA declarations on non-propagation ports
    tTaDecList = getTaDecList(tTaReg);
    for (tTaDec : tTaDecList) {
      if (tTaDec != NULL && tTaDec.portRef ∉ tPortList) {
        // process associated LA registration
        tLaReg = getAssociatedLaReg(tTaDec);
        if (tLaReg != NULL) {
          if (tLaReg.isReserved) {
            deallocateResources(tLaReg);
            tLaReg.isReserved = FALSE;
            tLaReg.reservationAge = 0;
            updateAllocatedBandwidth(tLaReg);
          }
          tLaReg.ingressStatus = NOT_PROPAGATED;
          // process LA declaration
          tLaDecList = getLaDecList(tLaReg);
          for (tLaDec : tLaDecList) {
            processLaDec(tLaDec);
          }
        }
        // remove TA declaration
        WITHDRAW_ATTRIBUTE.request(tTaDec.portRef,
                                    tTaDec.attr);
        delete taDec[tTaDec.portRef, tTaDec.attr.StreamId,
                                     tTaDec.attr.VID];
      }
    }
  }
}
```

Figure 51-25—Handling of Port status changes and VLAN topology changes in a RAP Propagator

Figure 51-26 defines the actions in the HANDLE_RESOURCE_CHANGE state for handling of a RESOURCE_CHANGE.indication (51.8.3.4).



```
for (tTaDec : tTaDec[*, *, *]) {
  tLaReg = getAssociatedLaReg(tTaDec);
  tNoReservation = (tLaReg == NULL) || !tLaReg.isReserved;
  if (iIncreased && tNoReservation && getTaStatus(tTaDec.attr) == Announce Fail) {
    // process TA declaration
    processTaDec(tTaDec);
    if (getTaStatus(tTaDec.attr) == Announce Success) {
      DECLARE_ATTRIBUTE.request(tTaDec.portRef, tTaDec.attr);
      // update associated LA registration
      if (tLaReg != NULL) {
        processLaReg(tLaReg);
        // propagate LA registration
        propagateLaReg(tLaReg);
        // process LA declaration
        tLaDecList = getLaDecList(tLaReg);
        for (tLaDec : tLaDecList) {
          processLaDec(tLaDec);
        }
      }
    }
  } else if (!iIncreased && tNoReservation && getTaStatus(tTaDec.attr) == Announce Success) {
    // process TA declaration
    processTaDec(tTaDec);
    if (getTaStatus(tTaDec.attr) == Announce Fail) {
      DECLARE_ATTRIBUTE.request(tTaDec.portRef, tTaDec.attr);
    }
  }
}
```

Figure 51-26—Handling of resource availability changes in a RAP Propagator

51.8.3 RAP Propagator state machine events

This subclause defines a set of events for triggering transitions from the IDLE state in a RAP Propagator state machine. These events are defined in terms of primitives, which can be divided into two groups.

The first group consists of all of the indication primitives defined in 51.7.2, which represent events generated at the interface between the per-Port RAP Participants and the RAP Propagator.

The second group consists of primitives defined in 51.8.3.1, 51.8.3.2 and 51.8.3.3, which represent events generated by the underlying mechanisms in the Bridge, such as the FDB.

51.8.3.1 VLAN_TOPOLOGY_CHANGE.indication(iVid)

A VLAN_TOPOLOGY_CHANGE.indication primitive indicates to the RAP Propagator state machine that there is a change in the VLAN topology identified by a given VID (iVid). Such a change can result from a spanning tree (7.3) or VLAN membership (7.4) reconfiguration.

51.8.3.2 MAC_ADDRESS_REGISTRATION.indication(iPortRef, iMacAddr)

A MAC_ADDRESS_REGISTRATION.indication primitive indicates to the RAP Propagator state machine that a MAC address (iMacAddr) is registered on a Port (iPortRef).

51.8.3.3 MAC_ADDRESS_DEREGISTRATION.indication(iPortRef, iMacAddr)

A MAC_ADDRESS_DEREGISTRATION.indication primitive indicates to the RAP Propagator state machine that a MAC address (iMacAddr) is deregistered on a Port (iPortRef).

51.8.3.4 RESOURCE_CHANGE.indication(iIncreased)

A RESOURCE_CHANGE.indication primitive indicates to the RAP Propagator state machine that there are increased (iIncreased == TRUE) or decreased (iIncreased == FALSE) remaining resources of this Bridge available for allocation to streams, as a consequence of one or more reservations being recently removed or created, respectively.

NOTE—As specified in Figure 51-26, this primitive is used to trigger reprocessing of the Talker Announce declarations that could be impacted by the indicated event to keep their status up-to-date. Proper usage of these primitives can help reduce repetitive computing tasks and avoid excessive processing burden. For example, issuing only a single primitive in the case of a series of reservations being removed could be more efficient than issuing one for each of them. The method used to decide when and how to issue this primitive is an implementation choice.

51.8.4 RAP Propagator state machine variables

51.8.4.1 taReg

The taReg variable is an array in which each element represents a Talker Announce registration maintained by the RAP Propagator state machine and consists of the following member variables:

a) **portRef**: The portRef (51.7.4.1) value of a Bridge Port holding the Talker Announce registration.
b) **attr**: The Talker Announce attribute (51.5.3) of the Talker Announce registration.
c) **isValid**: A Boolean indicating whether the Talker Announce registration is valid (TRUE) or not (FALSE).
d) **ingressStatus**: The ingress status of the Talker Announce registration, taking one of the following enumerated values:
    1) **TA_RECV_FAIL**: The Talker Announce registration contains a RAP Failure Code generated by an upstream station.
    2) **TA_INGRESS_SUCCESS**: The Talker Announce registration contains no RAP Failure Code and is not failed on ingress of this Bridge.
    3) **TA_INGRESS_FAIL**: This Talker Announce registration contains no RAP Failure Code but is failed on ingress of this Bridge with a RAP Failure Code contained in ingressFailureCode [item e), below].
e) **ingressFailureCode**: A RAP Failure Code specified in Table 51-13.
f) **rTagging**: A Boolean indicating whether the stream of this Talker Announce registration is subject to redundancy tagging (51.3.8.3) by this Bridge (TRUE) or not (FALSE). This variable is used only by a FRER-capable Bridge in processing a Multi-Context Talker Announcement.

The taRag variable is associated with a key <portRef, attr.StreamId, attr.VID>.

51.8.4.2 taDec

The taDec variable is an array in which each element represents a Talker Announce declaration being made by the RAP Propagator state machine and consists of the following member variables:

a) **portRef**: The portRef (51.7.4.1) value of a Bridge Port holding the Talker Announce declaration.
b) **attr**: The Talker Announce attribute (51.5.3) of the Talker Announce declaration.
c) **origTaReg**: A reference to an element in taReg (51.8.4.1) that is the single originating Talker Announce registration of the Talker Announce declaration, or NULL indicating the use of origTaRegList in item c) below.
d) **origTaRegList**: An array of references to one or more elements in taReg (51.8.4.1) that are originating Talker Announce registrations of the Talker Announce declaration. Each element in origTaRegList has a single variable origTaRegRef, which contains a reference to an originating Talker Announce registration. This variable is used only by a FRER-capable Bridge in handling a Multi-Context Talker Announce declaration that is subject to Talker Announce merge (51.3.4.3).
e) **rUntagging**: A Boolean indicating whether the stream of this Talker Announce declaration is subject to redundancy untagging (51.3.8.3) on the Port as indicated in portRef (TRUE) or not (FALSE). This variable is used only by a FRER-capable in handling a Multi-Context Talker Announce declaration that is subject to Talker Announce merge (51.3.4.3).

The taDec variable is associated with a key <portRef, attr.StreamId, attr.VID>.

51.8.4.3 laReg

The laReg variable is an array in which each element represents a Listener Attach registration maintained by the RAP Propagator state machine and consists of the following member variables:

a) **portRef**: The portRef (51.7.4.1) value of a Bridge Port holding the Listener Attach registration.
b) **attr**: The Listener Attach attribute (51.5.4) of the Listener Attach registration.
c) **assoTaDec**: A reference to an element in taDec (51.8.4.2) that is the associated Talker Announce declaration of the Listener Attach registration, or NULL indicating no associated Talker Announce declaration exists.
d) **isReserved**: A Boolean value indicating whether the Listener Attach registration is associated with a reservation in the Bridge (TRUE) or not (FALSE).
e) **reservationAge**: A 32-bit unsigned integer, indicating the time, in seconds, since a reservation associated with the Listener Attach registration was successfully made, and set to zero when the reservation is removed.
f) **ingressStatus**: The ingress status of the Listener Attach registration, taking one of the following enumerated values:
    1) **NOT_PROPAGATED**: The Listener Attach registration has no associated Talker Announce declaration and is not propagated.
    2) **ATTACH_READY**: The Listener Attach registration is propagated with **Attach Ready** [item a) in 51.5.4.3].
    3) **ATTACH_FAIL**: The Listener Attach registration is propagated with **Attach Fail** [item b) in 51.5.4.3].
    4) **ATTACH_PARTIAL_FAIL**: This Listener Attach registration is propagated with **Attach Partial Fail** [item c) in 51.5.4.3].

The laReg variable is associated with a key <portRef, attr.StreamId, attr.VID>.

51.8.4.4 laDec

The laDec variable is an array in which each element represents a Listener Attach declaration maintained by the RAP Propagator state machine and consists of the following member variables:

a) **assoTaReg**: A reference to an element in taReg (51.8.4.1) that is the associated Talker Announce registration of the Listener Attach declaration.
b) **attr**: The Listener Attach attribute (51.5.4) of the Listener Attach declaration.

c) **origLaRegList**: An array of references to one or more elements in laReg (51.8.4.3) that are originating Listener Attach registrations of the Listener Attach declaration and subject to Listener Attach merge (51.3.5.3). Each element in origLaRegList has a single variable origLaRegRef, which contains a reference to an originating Listener Attach registration.

The laDec variable is associated with a key <assoTaReg>.

51.8.4.5 localRaClass

The localRaClass variable is an array in which each element represents an RA class supported by the Bridge and consists of the following member variables:

a) **id**: An RA class ID (51.3.2.1).
b) **priority**: An RA class priority (51.3.2.2)
c) **rtid**: An RTID (51.3.2.3)
d) **templateDefinedData:** The value to be contained in the RaClassTemplateDefinedData field (51.5.2.1.5) of the RA Class Descriptor sub- TLV for this RA class and carried in the RA attribute declared by this Bridge.

The localRaClass variable is associated with a key <id>.

51.8.4.6 neighborRaClass

The neighborRaClass variable is an array in which each element represents an RA class supported by a neighboring station, corresponding to an RA Class Descriptor sub-TLV (51.5.2.1) contained in the RA attribute registered on a Port of this Bridge, and consists of the following member variables:

a) **portRef**: The portRef (51.7.4.1) value of the Bridge Port registering the RA attribute.
b) **id**: The value in the RaClassId field (51.5.2.1.1) of the RA Class Descriptor sub-TLV.
c) **priority**: The value in the RaClassPriority (51.5.2.1.2) field of the RA Class Descriptor sub-TLV.
d) **rtid**: The value in the RTID (51.5.2.1.3) field of the RA Class Descriptor sub-TLV.
e) **proposedMaxHopLatency:** The value in the ProposedMaxHopLatency (51.5.2.1.3) field of the RA Class Descriptor sub-TLV.
f) **templateDefinedData:** The value in the RaClassTemplateDefinedData (51.5.2.1.5) field of the RA Class Descriptor sub-TLV.

The neighborRaClass variable is associated with a key <portRef, id>.

51.8.4.7 port

The port variable is an array in which each element corresponds to a Bridge Port and consists of the following member variables:

a) **portRef**: The portRef (51.7.4.1) value of the associated Port.
b) **streamDaPruningEnabled:** A Boolean indicating whether Stream DA Pruning (51.3.4.1.2) is administratively enabled (TRUE) or disabled (FALSE) on the Port.
c) **partDecOper:** A Boolean indicating whether the attribute declaration function of the RAP Participant associated with the Port is operational (TRUE) or not (FALSE).
d) **portTransmitRate:** The transmission rate, in bits per second, of the underlying MAC service on the Port.
e) **maxInterferingFrameSize:** An unsigned integer, indicating the maximum frame size, in bytes, including media-dependent overhead (12.4.2.2), that is allowed to be transmitted through the Port. The value of this parameter is determined by the operation of the underlying MAC Service.

f) **maxPropagationDelay**: An unsigned integer, indicating the maximum latency, in nanoseconds, a frame can experience when transmitted from the underlying physical medium on the Port to a reception port connected via a LAN to the Port.
g) **minPropagationDelay**: An unsigned integer, indicating the minimum latency, in nanoseconds, a frame can experience when transmitted from the underlying physical medium on the Port to a reception port connected via a LAN to the Port.

The port variable is associated with a key <portRef>.

51.8.4.8 portRaClass

The portRaClass variable is an array in which each element is associated with an RA class supported by the Bridge and a Bridge Port, and consists of the following member elements:

a) **portRef**: The portRef (51.7.4.1) value of the associated Port.
b) **raClassId**: The RA class ID (51.3.2.1) of the associated local RA class.
c) **domainBoundaryStatus:** A Boolean indicating whether the Port is a domain boundary port for the RA class (TRUE) or not (FALSE).
d) **maxStreamFrameSize:** An unsigned integer, indicating the maximum frame size, in bytes, of the streams allowed to be transmitted in the RA class on the Port.
e) **minStreamFrameSize**: An unsigned integer, indicating the minimum frame size, in bytes, of the streams allowed to be transmitted in the RA class on the Port.
f) **maxBandwidth**: An unsigned integer, indicating the maximum amount of bandwidth that can be allocated to the streams reserved in the RA class on the Port. The bandwidth value is represented as a percentage of the portTransmitRate [item d) in 51.8.4.7] value on that Port and expressed as a fixed-point number scaled by a factor of 1,000,000; i.e., 100,000,000 (the maximum value) represents 100%.
g) **allocatedBandwidth**: An unsigned integer, indicating the amount of bandwidth that has been allocated to the streams reserved in the RA class on the Port. The bandwidth value is represented as a percentage of the portTransmitRate [item d) in 51.8.4.7] value on that Port and expressed as a fixed-point number scaled by a factor of 1,000,000; i.e., 100,000,000 (the maximum value) represents 100%.
h) **proposedMaxHopLatency:** An unsigned integer, indicating the administratively configured value, in nanoseconds, to be contained in the ProposedMaxHopLatency field of an RA Class Descriptor sub-TLV for the RA class in the RA attribute declared by this Bridge on the Port. The latency value is intended for use by a downstream station, e.g. an Listener, connected via a LAN to the Port to determine the corresponding maxHopLatency value [item d) in 51.8.4.9].

The portRaClass variable is associated with a key <portRef, raClassId>.

51.8.4.9 portPairRaClass

The portPairRaClass variable is an array in which each element is associated with an RA class supported by the Bridge and a reception Port transmission Port pair, and consists of the following member variables:

a) **receptionPortRef**: The portRef (51.7.4.1) value of the associated reception Port.
b) **transmissionPortRef**: The portRef (51.7.4.1) value of the associated transmission Port.
c) **raClassId**: The RA class ID (51.3.2.1) of the associated local RA class.
d) **maxHopLatency**: An unsigned integer, containing the administratively configured maximum latency value, in nanoseconds, that is used as an upper bound latency in the latency constraint [item a) in 51.3.6] imposed on each stream reserved in the RA class, received on the reception Port, and transmitted on the transmission Port. The latency is measured from a point located in an upstream station connected via a LAN to the reception Port, to a point located in the transmission Port, where

the exact measurement points are specific to and defined by the RA class template being used by the RA class.

The portPairRaClass variable is associated with a key <receptionPortRef, transmissionPortRef, raClassId>.

51.8.4.10 redundancyContext

The redundancyContext variable is an array in which each element represents a Redundancy Context supported by the Bridge and consists of the following member variables:

a) **id**: The Redundancy Context ID of the Redundancy Context, as defined in 51.3.8.2.
b) **vlanContextList**: The set of VLAN Context IDs associated with the Redundancy Context.

The redundancyContext variable is associated with a key <id>.

51.8.4.11 frerCapable

A Boolean value, indicating whether the Bridge is a FRER-capable Bridge (TRUE) or not (FALSE).

51.8.4.12 maxProcessingDelay

An unsigned integer, indicating the maximum delay, in nanoseconds, that a frame can experience during the forwarding process of the Bridge until it is placed into an outbound queue.

51.8.4.13 minProcessingDelay

An unsigned integer, indicating the minimum delay, in nanoseconds, that a frame can experience during the forwarding process of the Bridge until it is placed into an outbound queue.

51.8.5 RAP Propagator state machine procedures

51.8.5.1 processRaReg(pPortRef, pRaAttr)

This procedure processes the RA attribute (pRaAttr) registered on a Port (pPortRef), as follows:

a) Delete all the elements in neighborRaClass (51.8.4.6) whose portRef value matches pPortRef.
b) For each RA Class Descriptor sub-TLV (51.5.2.1), if any, contained in pRaAttr, create a new element in neighborRaClass and set each of its member variables to the corresponding value according to 51.8.4.6.

51.8.5.2 setDomainBoundaryStatus(pPortRef)

This procedure determines the RA class domain boundary status for each local RA class on a given Port (pPortRef), as follows:

```
setDomainBoundaryStatus(pPortRef) {
  for (tLocalRaClass : localRaClass[*]) {
    tNeighborRaClass = neighborRaClass[pPortRef, tLocalRaClass.id];
    if (tNeighborRaClass != NULL &&
        tNeighborRaClass.priority == tLocalRaClass.priority) {
      portRaClass[pPortRef, tLocalRaClass.id].domainBoundaryStatus = FALSE;
    } else {
      portRaClass[pPortRef, tLocalRaClass.id].domainBoundaryStatus= TRUE;
    }
  }
}
```

NOTE—The RA class domain boundary status determined by this procedure is used by the Bridge to adjust the operation of priority regeneration, in accordance with 6.9.4.

51.8.5.3 validateTaReg(pPortRef, pTaAttr)

<< Editor's note: A discussion of the operations needed in this procedure is found in Annex Z.7.1. >>

51.8.5.4 processTaReg(pTaReg)

This procedure processes a Talker Announce registration referenced by a taReg element (pTaReg), as follows:

```
processTaReg(pTaReg) {
  if (getTaStatus(pTaReg.attr) == Announce Fail) {
    // TA failed at an upstream station
    pTaReg.ingressStatus = TA_RECV_FAIL;
    return;
  }
  tLocalRaClass = getLocalRaClass(pTaReg.attr.Priority);
  tNeighborRaClass = getNeighborRaClass(pTaReg.portRef, pTaReg.attr.Priority);
  if (tLocalRaClass == NULL || tNeighborRaClass == NULL ||
                              tLocalRaClass.id != tNeighborRaClass.id){
    // TA registration is received across an RA class domain boundary
    pTaReg.ingressStatus = TA_INGRESS_FAIL;
    pTaReg.ingressFailureCode = 0x04;
    return;
  }
  // check of redundancy tagging conditions
  if (!isSingleContext(pTaReg.attr) && !pTaReg.attr.RedundancyControl.RTagStatus
                                                              && !frerCapable) {
    pTaReg.ingressStatus = TA_INGRESS_FAIL;
    pTaReg.ingressFailureCode = 0x05;
    return;
  }
  // check of stream spitting conditions
  if (!isSingleContext(pTaReg.attr) && !checkStreamSplitting(pTaReg)) {
    pTaReg.ingressStatus = TA_INGRESS_FAIL;
    pTaReg.ingressFailureCode = 0x09;
    return;
  }
  // set redundancy tagging
  if (isRedundancyTagging(pTaReg)) {
    pTaReg.rTagging = TRUE;
  } else {
    pTaReg.rTagging = FALSE;
  }
  pTaReg.ingressStatus = TA_INGRESS_SUCCESS;
}
```

51.8.5.5 propagateTaReg(pTaReg, pPortRef)

This procedure propagates a Talker Announce registration referenced by a taReg element (pTaReg) to a Port (pPortRef), by associating it with a new or existing Talker Announce declaration in taDec, as follows:

```
propagateTaReg(pTaReg, pPortRef) {
  tTaDec = getTaDec(pPortRef, pTaReg);
  if (tTaDec == NULL) { // a new propagated TA registration

    if (!isStreamMerging(pTaReg, pPortRef) { // not subject to TA merge
      tPropagatedVlanContextList = getPropagatedVlanContextList(pTaReg,
                                                                pPortRef);
      tVid = min(tPropagatedVlanContextList);
      tTaDec = create taDec[pPortRef, pTaReg.attr.StreamId, tVid];
      tTaDec.portRef = pPortRef;
      tTaDec.origTaReg = pTaReg;
    } else { // subject to TA merge
      // search for a TA declaration to merge
      tRedundancyContexId = getRedundancyContextId(pTaReg.attr.VID);
      tMergedVlanContexList = getMergedVlanContextList(tRedundancyContexId,
                                                        pPortRef);
      // use the numerically smallest VLAN Context ID as the merged VLAN Contexts
      tVid = min(tMergedVlanContexList);
      tTaDec = taDec[pPortRef, pTaReg.attr.StreamId, tVid];
      if (tTaDec == NULL) { // 1st TA to be merged
        tTaDec = create taDec[pPortRef, pTaReg.attr.StreamId, tVid];
        tTaDec.portRef = pPortRef;
        tTaDec.origTaReg = NULL;
      }
      // insert it to TA merge list
      tOrigTaReg = create tTaDec.origTaRegList[pTaReg];
      tOrigTaReg.origTaRegRef = pTaReg;
    }
  }
}
```

51.8.5.6 processTaDec(pTaDec)

This procedure processes a Talker Announce declaration referenced by a taDec element (pTaDec), as follows:

```
processTaDec(pTaDec) {
  if (pTaDec.origTaReg != NULL) { // not subject to TA merge
    processTaDecNonMerge(pTaDec);
  } else { // subject to TA merge
    processTaDecMerge(pTaDec);
  }
}
```

51.8.5.7 processTaDecNonMerge(pTaDec)

This procedure processes a Talker Announce declaration referenced by a taDec element (pTaDec) that is not subject to Talker Announce merge (51.3.4.3), as follows:

```
processTaDecNonMerge(pTaDec) {
  tTaReg = pTaDec.origTaReg;
  pTaDec.attr = tTaReg.attr;

  if(!isSingleContext(pTaDec.attr)) { // a Multi-Context TA
    tPropagatedVlanContextList = getPropagatedVlanContextList(tTaReg,
                                                        pTaDec.portRef);
    pTaDec.attr.VID = min(tPropagatedVlanContextList);
```

```
      // remove VLAN Context Information sub-TLV(s) not propagated
      for (tVlanContextInfoSubTlv : pTaDec.attr.VLANContextInformation[*]) {
        tVlanContextId = tVlanContextInfoSubTlv.VlanContextId;
        if (tVlanContextId NOT IN tPropagatedVlanContextList) {
          delete pTaDec.attr.VLANContextInformation[tVlanContextId];
        }
      }
    }

    if (tTaReg.ingressStatus == TA_RECV_FAIL) {
      // TA failed at an upstream Bridge, propagate it as is
      return;
    } else if (tTaReg.ingressStatus == TA_INGRESS_FAIL) {
      // TA failed at ingress of this Bridge
      failTaDec(pTaDec, tTaReg.ingressFailureCode);
    } else if (tTaReg.ingressStatus == TA_INGRESS_SUCCESS){ // TA succeeded so far
      // adjust accumulated latencies
      pTaDec.attr.AccumulatedMaximumLatency =
                              calAccuMaxLatency(pTaDec.origTaReg, pTaDec.portRef);
      pTaDec.attr.AccumulatedMinimumLatency =
                              calAccuMinLatency(pTaDec.origTaReg, pTaDec.portRef);
      tLaReg = getAssociatedLaReg(pTaDec);
      if (tLaReg == NULL || !tLaReg.isReserved) { // the stream not yet reserved
        // check reservation constraints
        tEgressFailureCode = checkReservationConstraints(pTaDec);
        if (tEgressFailureCode != 0) { // fail TA at egress
          failTaDec(pTaDec, tEgressFailureCode);
          return;
        }
      if (!isSingleContext(pTaDec.attr) && tTaReg.rTagging) {
        pTaDec.attr.RedundancyControl.RTagStatus = TRUE;
        adjustTSpecForRTag(pTaDec.attr, TRUE);
      }
    }
  }
```

51.8.5.8 processTaDecMerge(pTaDec)

This procedure performs Talker Announce merge (51.3.4.3) for a Talker Announce declaration element (pTaDec) in a taDec. It determines the Talker Announce attribute in pTaDec.attr for use by the Talker Announce declaration based on a list of one or more originating Talker Announce registrations contained in the pTaDec.origTaRegList, as follows:

a) Set pTaDec.attr.VID to the VID value used in the key of pTaDec, as initially determined at the creation of this Talker Announce declaration element by the propagateTaReg() procedure (51.8.5.5).
b) Construct a Redundancy Control sub-TLV (51.5.3.8) in pTaDec.attr.RedundancyControl as follows:
    1) tRedundancyContextId = **getRedundancyContextId**(pTaDec.attr.VID)
    2) tMergedVlanContextList = **getMergedVlanContextList**(tRedundancyContextId,
       pTaDec.portRef
    3) If **isRedundancyUntagging**(pTaDec) == TRUE, set pTaDec.RedundancyControl.RTagStatus to FALSE and set pTaDec.rUntagging to TRUE; otherwise, set pTaDec.rUntagging to FALSE.
    4) For each tTaReg in pTaDec.origTaRegList, copy each VLAN Context Information sub-TLV (51.5.3.9) in tTaReg.attr whose VlanContextId value is listed in tMergedVlanContextList to pTaDec.RedundancyControl. If a copied VLAN Context Information sub-TLV does not contain a Failure Information sub-TLV (51.5.3.10) and the tTaReg from which the sub-TLV is copied indicates in ingressStatus TA_INGRESS_FAIL, construct a Failure Information sub-

TLV using the RAP Failure Code contained in tTaReg.ingressFailureCode and the SystemId value of this Bridge, and then append it to the copied sub-TLV.

5) For each VLAN Context listed in tMergedVlanContextList but not indicated in any VLAN Context Information sub-TLV copied to pTaDec.RedundancyControl in b).4) above, construct a VLAN Context Information sub-TLV for that VLAN Context with a Failure Information sub-TLV that contains the RAP Failure Code 0x08 and the SystemId value of this Bridge, and then append the constructed VLAN Context Information sub-TLV to pTaDec.attr.RedundancyControl.

c) Call **mergeTSpec**(pTaDec) to assign values to either pTaDec.attr.TokenBucketTSpec or pTaDec.attr.MSRPTSpec.

d) Call **mergeAccumulatedLatencies**(pTaDec) to assign values to both pTaDec.attr.AccumulatedMaximumLatency and pTaDec.attr.AccumulatedMinimumLatency.

e) If there is at least one tTaReg listed in pTaDec.origTaRegList that indicates in ingressStatus TA_INGRESS_SUCCESS, perform the following actions:
   1) tEgressFailureCode = **checkReservationConstraints**(pTaDec);
   2) If tEgressFailureCode != 0, call **failTaDec**(pTaDec, tEgressFailureCode); otherwise, if pTaDec.rUntagging == TRUE, call **adjustTSpecForRTag**(pTaDec.attr, FALSE).

f) Otherwise, i.e., there is no tTaReg in pTaDec.origTaRegList that indicates in ingressStatus TA_INGRESS_SUCCESS, perform the following actions:
   1) set tEgressFailureCode to the numerically smallest value of one or more RAP Failure Codes contained in the VLAN Context Information sub-TLV(s) in pTaDec.attr.RedundancyControl.
   2) call **failTaDec**(pTaDec, tEgressFailureCode).

51.8.5.9 processLaReg(pLaReg)

This procedure processes a Listener Attach registration referenced by a laReg element (pLaReg), as follows:

```
processLaReg(pLaReg) {
  tTaDec = pLaReg.assoTaDec;
  if (tTaDec != NULL) { // has associated TA declaration
    if (pLaReg.attr.ListenerAttachStatus == Attach Fail ||
            getTaStatus(tTaDec.attr) == Announce Fail) {
      // reservation conditions not satisfied
      if (pLaReg.isReserved) { // remove the existing reservation
        deallocateResources(pLaReg);
        pLaReg.isReserved = FALSE;
        pLaReg.reservationAge = 0;
        updateAllocatedBandwidth(pLaReg);
      }
      pLaReg.ingressStatus = ATTACH_FAIL; // propagate LA as Attach Fail
    } else { // reservation conditions satisfied
      if (!plaReg.isReserved) {
        if (tTaDec.attr.StreamRank == 0) {
          premptReservationsForRank0(pLaReg);
        }
        tAllocated = allocateResources(pLaReg);
        if (tAllocated) {// resource allocation succeeded
          pLaReg.isReserved = TRUE;
          updateAllocatedBandwidth(pLaReg);
          pLaReg.ingressStatus = pLaReg.attr.ListenerAttachStatus;
        } else { // resource allocation failed
          pLaReg.isReserved = FALSE;
          pLaReg.ingressStatus = ATTACH_FAIL;
          // fail the TA declaration
          failTaDec(tTaDec, 0x06);
          DECLARE_ATTRIBUTE.request(tTaDec.portRef, tTaDec.attr);
        }
      }
    }
  } else { // associated TA not found
    pLaReg.ingressStatus = NOT_PROPAGATED;
  }
}
```

51.8.5.10 propagateLaReg(pLaReg)

This procedure propagates a given Listener Attach registration referenced by a laReg element (pLaReg), by associating it with one or more new or existing Listener Attach declaration elements in laDec, as follows:

```
propagateLaReg(pLaReg) {
  tTaDec = getAssociatedTaDec(pLaReg);
  if (tTaDec != NULL) {
    if (tTaDec.origTaReg != NULL) { // associated TA declaration is not merged
      tLaDec = laDec[tTaDec.origTaReg];
      if (tLaDec = NULL) { // propagated to a new LA declaration
        tLaDec = create laDec[tTaDec.origTaReg];
        tLaDec.assoTaReg = tTaDec.origTaReg;
        tOrigLaReg = create tLaDec.origLaRegList[pLaReg];
        tOrigLaReg.origLaRegRef = pLaReg;
      } else if (tLaDec.origLaRegList(pLaReg) == NULL) {
        // propagated to an existing LA declaration
        tOrigLaReg = create tLaDec.origLaRegList[pLaReg];
        tOrigLaReg.origLaRegRef = pLaReg;
      }
    } else { // associated TA declaration is merged
      for (tTaReg : tTaDec.origTaRegList[*]) {
        tLaDec = laDec[tTaReg];
        if (tLaDec = NULL) { // propagated to a new LA declaration
          tLaDec = create laDec[tTaReg];
          tLaDec.assoTaReg = tTaReg;
          tOrigLaReg = create tLaDec.origLaRegList[pLaReg];
          tOrigLaReg.origLaRegRef = pLaReg;
        } else if (tLaDec.origLaRegList(pLaReg) == NULL) {
          // propagated to an existing LA declaration
          tOrigLaReg = create tLaDec.origLaRegList[pLaReg];
          tOrigLaReg.origLaRegRef = pLaReg;
        }
      }
    }
  }
}
```

51.8.5.11 processLaDec(pLaDec)

This procedure processes a Listener Attach declaration referenced by a laDec element (pLaDec), as follows:

```
processLaDec(pLaDec) {
  tNumPropagated = tNumReady = tNumFailed = 0;
  for (tLaReg : pLaDec.origLaRegList[*] {
    if (tLaReg.ingressStatus == NOT_PROPAGATED) {
      delete pLaDec.origLaRegList[tLaReg];
      continue;
    } else if (tLaReg.ingressStatus == ATTACH_READY) {
      tNumReady++;
    } else if (tLaReg.ingressStatus == ATTACH_FAIL) {
      tNumFailed++;
    }
    tNumPropagated++;
  }
  if (tNumPropagated == 0) { // remove this LA declaration
    WITHDRAW_ATTRIBUTE.request(pLaDec.assoTaReg.portRef, pLaDec.attr);
    delete laDec[pLaDec.assoTaReg];
    return;
  }
  // construct the Listener Attach attribute
  pLaDec.attr.StreamId = pLaDec.assoTaReg.attr.StreamId;
  pLaDec.attr.VID = pLaDec.assoTaReg.attr.VID;
  if (isSingleContext(pLaDec.assoTaReg) { // a single context LA declaration
    if (tNumPropagated == tNumReady) { // all propagated as Attach Ready
      pLaDec.attr.ListenerAttachStatus = Attach Ready;
    } else if (tNumPropagated == tNumFailed) { // all propagated as Attach Fail
      pLaDec.attr.ListenerAttachStatus = Attach Fail;
    } else { // Otherwise: declare Attach Partial Fail
      pLaDec.attr.ListenerAttachStatus = Attach Partial Fail;
    }
  } else { // a Multi-Context LA declaration
    pLaDec.VLANContextStatus = constructVlanContextStatus(pLaDec);
    if (tNumReady != 0) { // at least one propagated as Attach Ready
      pLaDec.attr.ListenerAttachStatus = Attach Ready;
    } else {
      pLaDec.attr.ListenerAttachStatus = Attach Fail;
    }
    setMultiContextLaDecVid(pLaDec);
  }
  DECLARE_ATTRIBUTE.request(pLaDec.assoTaReg.portRef, pLaDec.attr);
}
```

51.8.5.12 getTaRegDestPorts(pTaReg)

This procedure determines the Port(s) to which a Talker Announce registration referenced by a taReg element (pTaReg) is to be propagated. It returns a list of portRef (51.7.4.1) values, possibly empty, as follows:

a) If the propagation of this Taker Announce registration is prohibited due to Ingress Blocking as specified in 51.3.4.1.3, return an empty list.
b) Otherwise, return a list that includes each possible portRef value for which all of the following conditions are met:
   1) portRef != pTaReg.portRef;
   2) port[portRef].partDecOper == TRUE;
   3) If **isSingleContext**(pTagReg.attr) == TRUE, portRef is in the VLAN Context as indicated by the pTaReg.attr.VID value according to 51.5.3.5.3;
   4) If **isSingleContext**(pTagReg.attr) == FALSE, portRef is in at least one of the VLAN Contexts as indicated in the VLAN Context Information sub-TLV(s) contained in pTaReg.attr, according to 51.5.3.9;
   5) If port[portRef].streamDaPruningEnabled == TRUE, pTaReg.attr.DestinationMacAddress is registered on the Port referenced by portRef, according to 51.3.4.1.2.

51.8.5.13 isRedundancyTagging(pTaReg)

This procedure determines whether a stream as indicated by a Talker Announce registration element (pTaReg) in taReg is subject to redundancy tagging (51.3.8.3). It returns TRUE, if all of the following conditions are met:

a) **isSingleContext**(pTaReg.attr) != TRUE;
b) frerCapable == TRUE;
c) pTaReg.attr.RedundancyControl.RTagStatus == FALSE;

Otherwise, it returns FALSE.

51.8.5.14 isRedundancyUntagging(pTaDec)

This procedure determines whether a stream as indicated by a Talker Announce declaration element (pTaDec) in taDec is subject to redundancy untagging (51.3.8.3). It returns TRUE, if all of the following conditions are met:

a) **isSingleContext**(pTaDec.attr) != TRUE;
b) frerCapable == TRUE;
c) Let tRedundancyContextId = **getRedundancyContextId**(pTaDec.attr.VID) and tMergedVlanContextList = **getMergedVlanContextList**(tRedundancyContextId, pTaDec.portRef). The tMergedVlanContextList contains all the VLAN Contexts in redundancyContext[tRedundancyContextId].

Otherwise, it returns FALSE.

51.8.5.15 checkStreamSplitting(pTaReg)

This procedure checks whether a stream as indicated by a Talker Announce registration element (pTaReg) in taReg meets the requirements for stream splitting as described in 51.3.8.4. It returns FALSE, if all of the following conditions are met:

a) Let tPortList = **getTaRegDestPorts**(tTaReg). The tPortList contains more than one Port.
b) For each Port (tPortRef) in tPortList, let tPropagatedVlanContextList = **getPropagatedVlanContextList**(pTaReg, tPortRef). There is one Port for which the value in tPropagatedVlanContextList is not the same as that of another Port.
c) Either of the following two conditions is met:
    1) frerCapable == FALSE.
    2) frerCapable == TRUE and there is no VLAN Context commonly present in each tPropagatedVlanContextList obtained in b).

Otherwise, it returns TRUE.

51.8.5.16 isStreamMerging(pTaReg, pPortRef)

This procedure determines whether a stream as indicated by a Talker Announce registration element (pTaReg) in taReg is subject to stream merging (51.3.8.5) on the Port (pPortRef). It returns TRUE, if all of the following conditions are met:

a) **isSingleContext**(pTaReg.attr) != TRUE;
b) frerCapable == TRUE;
c) Let tPropagatedVlanContextList = **getPropagatedVlanContextList**(pTaReg, pPortRef), tRedundancyContextId = **getRedundancyContextId**(pTaReg.attr.VID), and

tMergedVlanContextList = **getMergedVlanContextList**(tRedundancyContextId, pPortRef). The list of VID(s) contained in tPropagatedVlanContextList is a subset (less than) of the list of VID(s) contained in tMergedVlanContextList.

Otherwise, it returns FALSE.

51.8.5.17 getPropagatedVlanContextList(pTaReg, pPortRef)

This procedures returns a list containing each VLAN Context ID used in propagation of a given Talker Announce registration element (pTaReg) in taReg to a given Port (pPortRef), according to the VLAN Context(s) indicated in pTaReg.attr and the VLAN membership of the Port, as described in 51.3.4.1.1.

51.8.5.18 getMergedVlanContextList(pRedundancyContextId, pPortRef)

This procedures returns a list containing each VLAN Context ID associated with the Redundancy Context identified by pRedundancyContextId and including the Port pPortRef in its member set.

51.8.5.19 getRedundancyContextId(pVlanContextId)

This procedures returns the Redundancy Context ID of a Redundancy Context in redundancyContext (51.8.4.10) that includes a VLAN Context ID contained in pVlanContextId.

51.8.5.20 getTaDec(pPortRef, pTaReg)

This procedure searches in taDec for a Talker Announce declaration element that contains in either origTaReg or origTaRegList a reference to a given Talker Announce registration element (pTaReg) in taReg and contains in portRef a value matching pPortRef. If such an element is found in taDec, it returns a reference to the matching element in taDec. Otherwise, it returns NULL.

51.8.5.21 getTaDecList(pTaReg)

This procedure searches in taDec for one or more Talker Announce declaration elements, each containing in either origTaReg or origTaRegList a reference to a given Talker Announce registration element (pTaReg) in taReg. If such elements are found in taDec, it returns a list of references to the matching element(s) in taDec. Otherwise, it returns NULL.

51.8.5.22 getAssociatedTaDec(pLaReg)

This procedure returns a reference to a taDec element that contains a Talker Announce declaration with which a given Listener Attach registration referenced by a laReg element (pLaReg) is associated, in accordance with 51.3.5.1, or NULL if such a taDec element does not exist.

51.8.5.23 getAssociatedLaReg(pTaDec)

This procedure returns a reference to a laReg element that contains a Listener Attach registration associated with a given Talker Announce declaration referenced by a taDec element (pTaDec), in accordance with 51.3.5.1, or NULL if such a laReg element does not exist.

51.8.5.24 getLaDecList(pLaReg)

This procedure searches in laDec for one or more Listener Attach declaration elements, each containing in origLaRegList a reference to a given Listener Attach registration element (pLaReg) in taReg. If such elements are found in taDec, it returns a list of references to the matching element(s) in laDec. Otherwise, it returns NULL.

51.8.5.25 isSingleContextTa(pTaAttr)

This procedure returns TRUE if the Talker Announce attribute supplied in pTaAttr indicates a Single-Context Talker Announcement according to [item a) in 51.5.3], and returns FALSE if pTaAttr indicates a Multi-Context Talker Announcement according to the [item b) in 51.5.3].

51.8.5.26 getTaStatus(pTaAttr)

This procedure returns the Talker Announce status of the Talker Announce attribute supplied in pTaAttr, according to 51.3.4.2.

51.8.5.27 calAccuMaxLatency(pTaReg, pPortRef)

This procedure computes and returns the maximum accumulated latency value for a Talker Announce registration element (pTaReg) being propagated to a Port (pPortRef), as follows:

```
calAccuMaxLatency(pTaReg, pPortRef) {
  tRaClass = getLocalRaClass(pTaReg.attr.priority);
  tMaxHopLatency =
           portPairRaClass[pTaReg.portRef, pPortRef, tRaClass.id].maxHopLatency;
  return pTaReg.attr.AccumulatedMaximumLatency + tMaxHopLatency;
}
```

51.8.5.28 calAccuMinLatency(pTaReg, pPortRef)

This procedure computes and returns the minimum accumulated latency value for a Talker Announce registration element (pTaReg) being propagated to a Port (pPortRef), as follows:

```
calAccuMinLatency(pTaReg, pPortRef) {
  tRxPort = pTaReg.portRef;
  tMinHopLatency = minProcessingDelay + port[tRxPort].minPropagationDelay +
           pTaReg.attr.TokenBucketTSpec.MinTransmittedFrameLength*8*1e9 /
                                          port[tRxPort].portTransmitRate;
  return pTaReg.attr.AccumulatedMinimumLatency + tMinHopLatency;
}
```

51.8.5.29 mergeAccumulatedLatencies(pTaDec)

This procedure adjusts the accumulated latency values in a Talker Announce declaration element (pTaDec) in taDec that is subject to Talker Announce merge (51.3.4.3), as follows:

```
mergeAccumulatedLatencies(pTaDec) {
  tMaxAccuMaxLatency = tMinAccuMinLatency = 0;
  for (tTaReg : pTaDec.origTaRegList[*]) {
    if (tTaReg.ingressStatus = TA_INGRESS_SUCCESS) {
      tAccuMaxLatency = calAccuMaxLatency(tTaReg, pTaDec.portRef);
      tAccuMinLatency = calAccuMinLatency(tTaReg, pTaDec.portRef);
      if (tAccuMaxLatency > tMaxAccuMaxLatency) {
        tMaxAccuMaxLatency = tAccuMaxLatency;
      }
      if (tMinAccuMinLatency == 0 || tAccuMinLatency < tMinAccuMinLatency) {
        tMinAccuMinLatency = tAccuMinLatency;
      }
    }
  }
  pTaDec.attr.AccumulatedMaximumLatency = tMaxAccuMaxLatency;
  pTaDec.attr.AccumulatedMinimumLatency = tMinAccuMinLatency;
}
```

51.8.5.30 mergeTSpec(pTaDec)

This procedures determines the TSpec values in a Talker Announce declaration element (pTaDec) in taDec that is subject to Talker Announce merge (51.3.4.3), as follows:

51.8.5.31 adjustTSpecForRTag(pAttr, pRTagging)

This procedure adjusts the TSpec values in the Talker Announce attribute (pAttr) due to the operation of redundancy tagging (pRTagging == TRUE) or redundancy untagging (pRTagging ==FALSE) by increasing or decreasing, respectively, the stream frame length by the length of a R-TAG, which is 6 octets as specified in IEEE Std 802.1CB, as follows:

```
adjustTSpecForRTag(pTaDec, pRTagging) {
  if (pRTagging) { //
    if (pTaDec.attr.TokenBucketTSpec != NULL) {
      pTaDec.attr.TokenBucketTSpec.MaxTransmittedFrameLength += 6;
      pTaDec.attr.TokenBucketTSpec.MinTransmittedFrameLength += 6;
    } else {
      pTaDec.attr.MSRPTSpec.MaximumFrameSize += 6;
    }
  } else {
    if (pTaDec.attr.TokenBucketTSpec != NULL) {
      pTaDec.attr.TokenBucketTSpec.MaxTransmittedFrameLength -= 6;
      pTaDec.attr.TokenBucketTSpec.MinTransmittedFrameLength -= 6;
    } else {
      pTaDec.attr.MSRPTSpec.MaximumFrameSize -= 6;
    }
  }
}
```

51.8.5.32 failTaDec(pTaDec, pFailureCode)

This procedure fails a Talker Announce declaration referenced by a taDec element (pTaDec), as follows:

a) Construct a Failure Information sub-TLV (51.5.3.10) using the System ID of this Bridge and a RAP Failure Code contained in pFailureCode.
b) Append the Failure Information sub-TLV constructed in item a) to the Talker Announce attribute contained in pTaDec.attr, in accordance with 51.5.3.
c) If pTaDec.origTaReg == NULL, indicating a merged Multi-Context Talker Announce declaration, append the Failure Information sub-TLV constructed in item a) to each VLAN Context Information sub-TLV (51.5.3.9) in pTaDec.attr that currently contains no Failure Information sub-TLV.

51.8.5.33 constructVlanContextStatus(pLaDec)

This procedure constructs and returns in tVlanContextStatus a VLAN Context Status sub-TLV (51.5.4.4) for a Multi-Context Listener Attach declaration element (pLaDec) in laDec, as follows:

a) For each originating Listener Attach registration (tLaReg) contained in pLaDec.origLaRegList, perform the following actions:
   1) tPropagatedVlanContextStatus = tLaReg.attr.VLANContextStatus;

   2) Delete in tPropagatedVlanContextStatus each VLAN Context tuple for a VLAN Context that is not indicated in pLaDec.assoTaReg.attr.RedundancyControl.
   3) If tLaReg.ingressStatus ==ATTACH FAIL, set the PathStatus (51.5.4.4.1) of each VLAN Context tuple contained in tPropagatedVlanContextStatus to **Faulty Path**.

b) Construct a VLAN Context Status sub-TLV in tVlanContextStatus, as follows:
   1) Copy each VLAN Context tuple contained in each tPropagatedVlanContextStatus obtained in a) to tVlanContextStatus.
   2) For each VLAN Context indicated in pLaDec.assoTaReg.attr.RedundancyControl but not contained in tVlanContextStatus, create a VLAN Context tuple for that VLAN Context with its PathStatus set to **Unresponsive Path** and then append it to tVlanContextStatus.

c) If there is more than one VLAN Context tuple in tVlanContextStatus indicating the same VLAN Context, retain only one of them and delete the others, with the following rules:
   1) If there is one whose PathStatus is **Faulty Path**, retain this one;
   2) Else if there is one whose PathStatus is **Faultless Path**, retain this one;
   3) Otherwise, retain any one.

d) Return tVlanContextStatus.

51.8.5.34 setMultiContextLaDecVid(pLaDec)

This procedure determines the VID (51.5.4.2) value for a Multi-Context Listener Attach declaration element (pLaDec) in laDec, as follows:

a) If there is only one VLAN Context indicated in pLaDec.attr.VLANContextStatus, set pLaDec.attr.VID to the VLAN Context ID of that VLAN Context and return.
b) Let tPortList = **getTaRegDestPorts**(pLaDec.assoTaReg); For each Port (tPortRef) in tPortList, let tPropagatedVlanContextList = **getPropagatedVlanContextList**(pLaDec.assoTaReg, tPortRef);
c) If there is one or more VLAN Contexts commonly present in each tPropagatedVlanContextList obtained in b), set pLaDec.attr.VID to the numerically smallest VLAN Context ID of the common VLAN Context(s).
d) Otherwise, set pLaDec.attr.VID to the numerically smallest VLAN Context ID of the VLAN Contexts in pLaDec.attr.VLANContextStatus.

51.8.5.35 constructRaAttr(pPortRef)

This procedure constructs and returns an RA attribute (tRaAttr) to be declared on a Port (pPortRef), as follows:

a) For each localRaClass (51.8.4.5) element (tLocalRaClass), construct an RA Class Descriptor sub-TLV (raDescTlv) in accordance with 51.5.2.1 and fill its Value field, as follows:

```
raDescTlv.RaClassID             = tLocalRaClass.id;
raDescTlv.RaClassPriority       = tLocalRaClass.priority;
raDescTlv.RTID                  = tLocalRaClass.rtid;
raDescTlv.ProposedMaxHopLatency =
           portRaClass[pPortRef, tLocalRaClass.id].proposedMaxHopLatency;
raDescTlv.RaClassTemplateDefinedData = tLocalRaClass.templateDefinedData;
```

b) Construct an RA attribute TLV (tRaAttr) and fills its Value field with raDescTlv(s) constructed in item a) above, in accordance with 51.5.2.
c) Return tRaAttr.

51.8.5.36 getLocalRaClass(pPriority)

This procedure returns a reference to a localRaClass (51.8.4.5) element whose priority value matches the pPriority value.

51.8.5.37 getNeighborRaClass(pPortRef, pPriority)

This procedure returns a reference to a neighborRaClass (51.8.4.6) element whose portRef value matched the pPortRef value and whose priority value matches the pPriority value.

51.8.5.38 getTrafficClass(pPortRef, pPriority)

This procedure returns the traffic class to which a priority (pPriority) is assigned on a given Bridge Port (pPortRef).

51.8.5.39 checkReservationConstraints(pTaDec)

This procedure performs a series of checks to determine whether a Talker Announce declaration referenced by a taDec element (pTaDec) meets the reservation constraints (51.3.6). The procedure described in this subclause specifies only the input and output parameters, along with the rules of handling the stream Rank, but not the detailed algorithms that are either specific to a given RA class template or dependent on the particular Bridge implementation.

Depending on the stream Rank value, the procedure takes the existing reservations in this Bridge into account in the following two different ways:

a) If pTaDec.attr.StreamRank contains the value zero, only those existing reservations being made for the streams with Rank zero are treated as “reserved” and all other existing reservations for the streams with Rank one are treated as “not reserved”.
b) If pTaDec.attr.StreamRank contains the value one, all of the existing reservations regardless of the stream Rank are treated as “reserved”.

The procedure performs the needed calculations for checking each reservation constraint defined in 51.3.6, for which there is no requirement for the order in which one constraint is examined after another. When determining that one or more of these constraints are not met, the procedure returns a RAP Failure Code defined in Table 51-13 associate with one constraint not met. It returns the value zero, if determining that all of the reservation constraints are met.

The procedures described in 51.8.5.44 and 51.8.5.46 provide example algorithms for checking the latency constraint for streams using an RA class template defined in Table 51-3.

51.8.5.40 premptReservationsForRank0(pLaReg)

This procedure is invoked prior to making a reservation for a stream with Rank zero and indicated by a Listener Attach registration referenced by a laReg element (pLaReg) to perform the following actions:

a) Determine whether it is necessary to remove one or more reservations of the streams with Rank one, if necessary, determine which of those reservations are to be removed according to the rules of reservation importance as defined in 51.3.7.

NOTE—For making the decisions required by a), this procedure can use the same algorithms as used by the checkResevationConstraints() procedure (51.8.5.39) with different assumptions about which of the existing reservations are treated “reserved” or “not reserved”.

b) For each reservation to be moved as determined in a), if any, perform the following actions on a Listener Attach registration (tLaReg) with which the reservation is associated:

```
deallocateResources(tLaReg);
tLaReg.isReserved = FALSE;
tLaReg.reservationAge = 0;
```

```
      updateAllocatedBandwidth(tLaReg);
      tTaDec = getAssociatedTaDec(tLaReg);
      failTaDec(tTaDec, 0x07);
      DECLARE_ATTRIBUTE.request(tTaDec.portRef, tTaDec.attr);
      tLaReg.ingressStatus = ATTACH_FAIL;
      for (tLaDec : getLaDecList(tLaReg)) {
        processLaDec(tLaDec);
      }
```

51.8.5.41 allocateResources(pLaReg)

This procedure is invoked to create a reservation for a stream as indicated by a Listener Attach registration referenced by a laReg element (pLaReg), by performing actions such as configuration and resource assignment in the underlying Bridge mechanisms as required by the stream.

If all of the actions required for the reservation have been successfully carried out, the procedure starts time measurement for pLaReg.reservationAge and returns TRUE. If one or more of the required actions have failed, the procedure returns FALSE.

51.8.5.42 deallocateResources(pLaReg)

This procedure is invoked to remove a reservation associated with a Listener Attach registration referenced by a laReg element (pLaReg), by reversing all the changes to the underlying Bridge mechanisms previously made by the allocateResources() procedure (51.8.5.41) in creating this reservation.

51.8.5.43 updateAllocatedBandwidth(pLaReg)

This procedure is invoked, when creating or removing a reservation associated with a Listener Attach registration referenced by a laReg element (pLaReg), to update the corresponding allocatedBandwidth value [item g) in 51.8.4.8], as follows:

```
updateAllocatedBandwidth(pLaReg) {
  tTaDec = getAssociatedTaDec(pLaReg);
  tRaClass = getLocalRaClass(pTaDec.attr.priority);
  tPort = pLaReg.portRef;
  tStreamBandwidth =
            ceil(1e8 * pTaDec.attr.TokenBucketTSpec.CommittedInformationRate /
                                                   port[tPort].portTransmitRate);
  if (pLaReg.isReserved) {
    portRaClass[tPort, tRaClass.id].allocatedBandwidth += tStreamBandwidth;
  } else {
    portRaClass[tPort, tRaClass.id].allocatedBandwidth -= tStreamBandwidth;
  }
}
```

51.8.5.44 checkLatencyConstraintSP(pTaDec)

This subclause provides an example of checking the latency constraint for resource allocation in RA classes that use Strict Priority. The procedure is described as part of the **checkReservationConstraints**() procedure (51.8.5.39) invoked during processing of a Talker Announce declaration element (pTaDec) in taDec.

```
checkLatencyConstraintSP(pTaDec) {
  for (tObsvRaClass: localRaClass[*]) {
    tTxPort = pTaDec.portRef;
```

```
      tRxPort = pTaDec.origTaReg.portRef;
      tMaxLowerPrioFrameSize = port[tTxPort].maxInterferingFrameSize; // bytes
      tSumBursts = getInterferingBurstSizeSP(tObsvRaClass, pTaDec);
      // collect bursts from existing reservations on this port
      for (tLaReg: laReg[tTxPort, *, *]) {
        if (tLaReg.isReserved) {
          tTaDec = getAssociatedTaDec(tLaReg);
          tSumBursts += getInterferingBurstSizeSP(tObsvRaClass, tTaDec);
        }
      }
      tSumBursts += tMaxLowerPrioFrameSize * 8;
      tMaxQueuingDelay = ceil(1e9 * tSumBursts / port[tRxPort].portTransmitRate);
      tMaxFrameSize = pTaDec.attr.TokenBucketTSpec.MaxTransmittedFrameLength;
      tMaxSFDelay = tMaxFrameSize * 8 * 1e9 / port[tRxPort].portTransmitRate;
      tCurrMaxHopLatency = tMaxQueuingDelay + port[tRxPort].maxPropagationDelay +
                                    tMaxSFDelay + maxProcessingDelay;
      if (tCurrMaxHopLatency > portPairRaClass[tRxPort, tTxPort,
                                            tObsvRaClass.id].maxHopLatency) {
        return 0x01; // latency constraint not met
      }
    }
    return 0; // latency constraint met
  }
```

51.8.5.45 getInterferingBurstSizeSP(pObservedRaClass, pTaDec)

As part of the example in 51.8.5.44 for Strict Priority, this procedure computes the amount of interference generated by transmission of a stream on a Port referenced by a taDec element (pTaDec), to a given RA class referenced by a localRaClass element (pObservedRaClass) on that Port, as follows:

```
getInterferingBurstSizeSP(pObservedRaClass, pTaDec) {
  tAttr = pTaDec.attr;
  tTxPort = pTaDec.portRef;
  tRxPort = pTaDec.origTaReg.portRef;
  tObsvTrafficClass = getTrafficClass(tTxPort, pObservedRaClass.priority);
  tInterferingTrafficClass = getTrafficClass(tTxPort, tAttr.priority);
  tCurrentHopMinLatency = minProcessingDelay + port[tTxPort].minPropagationDelay
               + tAttr.TokenBucketTSpec.MinTransmittedFrameLength * 8 * 1e9 /
                                           port[tTxPort].portTransmitRate;
  if (tObsvTrafficClass == tInterferingTrafficClass) {
    tTimeFrame = tAttr.AccumulatedMaximumLatency –
              (tAttr.AccumulatedMinimumLatency – tCurrentHopMinLatency);
    tInterferingBurstSize = tAttr.TokenBucketTSpec.CommittedBurstSize +
                tAttr.TokenBucketTSpec.CommittedInformationRate * tTimeFrame;
  } else if (tObsvTrafficClass < tInterferingTrafficClass) {
      tTimeFrame = tAttr.AccumulatedMaximumLatency –
              (tAttr.AccumulatedMinimumLatency – tCurrentHopMinLatency) +
          portPairRaClass[tRxPort, tTxPort, tObsvTrafficClass.id].maxHopLatency;
      tInterferingBurstSize = tAttr.TokenBucketTSpec.CommittedBurstSize +
                tAttr.TokenBucketTSpec.CommittedInformationRate * tTimeFrame;
  } else {
    tInterferingBurstSize = 0;
  }
  return tInterferingBurstSize;
}
```

51.8.5.46 checkLatencyConstraintATS(pTaDec)

This procedure provides an example of checking the latency constraint for resource allocation in RA classes that use Asynchronous Traffic Shaping defined in this standard. The procedure is described as part of the **checkReservationConstraints**() procedure (51.8.5.39) invoked during processing of a Talker Announce declaration element (pTaDec) in taDec.

```
checkLatencyConstraintATS(pTaDec) {
  for (tObsvRaClass: localRaClass[*]) {
    tTxPort = pTaDec.portRef;
    tRxPort = pTaDec.origTaReg.portRef;
    tAttr = pTaDec.attr;
    tObservedTrafficClass = getTrafficClass(tTxPort, tObsvRaClass.priority);
    tSumRates = 0; // bits/s
    tSumBursts = tAttr.TokenBucketTSpec.CommittedBurstSize; // bits
    tMaxLowerPrioFrameSize = port[tTxPort].maxInterferingFrameSize; // bytes
    tMinEqualPrioFrameSize = tAttr.TokenBucketTSpec.MinTransmittedFrameLength;
    for (tLaReg: laReg[tTxPort, *, *]) {
      if (tLaReg.isReserved) {
        tTaDec = getAssociatedTaDec(tLaReg);
        tInterferTrafficClass = getTrafficClass(tTxPort, tTaDec.attr.priority);
        if (tInterferTrafficClass > tObservedTrafficClass) {
          tSumBursts += tTaDec.attr.TokenBucketTSpec.CommittedBurstSize;
          tSumeRates += tTaDec.attr.TokenBucketTSpec.CommittedInformationRate;
        }
        else if (tInterferTrafficClass == tObservedTrafficClass) {
          tSumBursts += tTaDec.attr.TokenBucketTSpec.CommittedBurstSize;
          if (tTaDec.attr.TokenBucketTSpec.MinTransmittedFrameLength <
                                                    tMinEqualPrioFrameSize) {
            tMinEqualPrioFrameSize =
                        tTaDec.attr.TokenBucketTSpec.MinTransmittedFrameLength;
          }
        }
      }
    }
    tSumBursts = tSumBursts + (tMaxLowerPrioFrameSize-tMinEqualPrioFrameSize)*8;
    tRemainingDateRate = port[tTxPort].portTransmitRate - tSumRates;
    tMaxQueuingDelay = ceil(tSumBursts / tRemainingDateRate +
            tMinEqualPrioFrameSize * 8 / port[tTxPort].portTransmitRate) * 1e9;
    tMaxFrameSize = pTaDec.attr.TokenBucketTSpec.MaxTransmittedFrameLength;
    tMaxSFDelay = tMaxFrameSize * 8 * 1e9 / port[tRxPort].portTransmitRate;
    tCurrMaxHopLatency = tMaxQueuingDelay + port[tRxPort].maxPropagationDelay +
                                                     tMaxSFDelay + maxProcessingDelay;
    if (tCurrMaxHopLatency > portPairRaClass[tRxPort, tTxPort,
                                                    tObsvRaClass.id].maxHopLatency) {
      return 0x01; // latency constraint not met
    }
  }
  return 0; // latency constraint met
}
```

51.9 RAP Failure Codes

This subclause defines a set of RAP Failure Codes, each with a non-zero integer value and associated with a cause of the failure, as listed in Table 51-13.

Table 51-13—RAP Failure Codes

RAP Failure Code	Description of cause
0x01	Latency constraint not met
0x02	Bandwidth constraint not met
0x03	Resource Constraint not met
0x04	Talker Announcement across RA Class domain boundary
0x05	FRER-capable Bridge at edge required
0x06	Resource allocation failed
0x07	Reservation preempted by Rank zero streams
0x08	Talker Announce in this VLAN Context not received
0x09	Stream splitting failed

Annex A

(normative)

PICS proforma—Bridge implementations

Annex B

(normative)

PICS proforma—End station implementations

Annex D

(normative)

IEEE 802.1 Organizationally Specific TLVs

D.2.12 EVB TLV

Change the text in D.2.12.5 as follows:

D.2.12.5R

This field carries ~~the~~ a proposed maxRetries value for the ECP state machine (43.3.7.4). ~~Both sides transmit the local value, and use the largest of the two values of R. If no remote value is available, then the local value is used.~~ The value of maxRetries is the largest of the local EVB R, remote EVB R, and the ECP management object ecpProposedR. The value determined for maxRetries will be reflected in the ECP management object ecpOperMaxRetries.

Change the text in D.2.12.6 as follows:

D.2.12.6 RTE (retransmission exponent)

RTE is an EVB link ~~or S-channel~~ attribute used to calculate the minimum ECPDU retransmission time, ackTimerInit. The value of ackTimerInit, in ECP timer tics of 10 usec, is calculated as:

2^{RTE} ECP timer tics

Both sides transmit the local value, and use the largest of the ~~two values of RTE for this calculation~~ three values which are the local and remote RTE and the management object ecpProposedRTE. If no remote value is available, then the greater of 2 ms ~~and~~, local value and ecpProposedRTE is used.

Annex Y

(informative)

Resource allocation examples

This Annex provides some examples of resource allocation using the Resource Allocation Protocol (RAP) specified in Clause 51. These examples are chosen only for the purpose of illustrating the operation of resource allocation under different scenarios, and are not intended to constraint the range of applicability of RAP in terms of all of the possible usage scenarios.

Note that, unless otherwise stated, the examples in this Annex are based on the following assumptions:

a) In Talker Announce propagation (51.3.5.2), no further conditions, other than those related to the VLAN Context rule as defined in 51.3.4.1.1, that would prevent a Talker Announce registration from being propagated to other Port(s), are met.
b) Resource allocation is successful without encountering any problems that would cause Talker Announcement or Listener Attachment to fail (except the example in Y.5)

Y.1 Single-path stream example

This subclause gives an example of resource allocation for transmission of a stream along a single path between the Talker and each Listener. Figure Y-1 shows the VLAN topology configured in the network for use as a VLAN Context in the resource allocation for the stream.

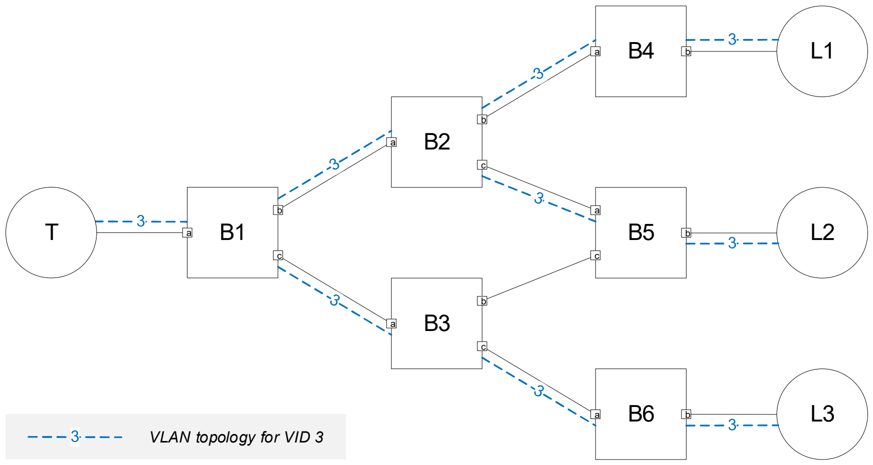


Figure Y-1—Single-path stream example: topology and VLAN configuration

Figure Y-2 illustrates the propagation of a Single-Context Talker Announcement [item a) in 51.3.4.1.1] initiated by the Talker within the VLAN Context. Note that the Talker Announcement does not go through the link between B3 and B5, as this link is not part of the VLAN Context in use.

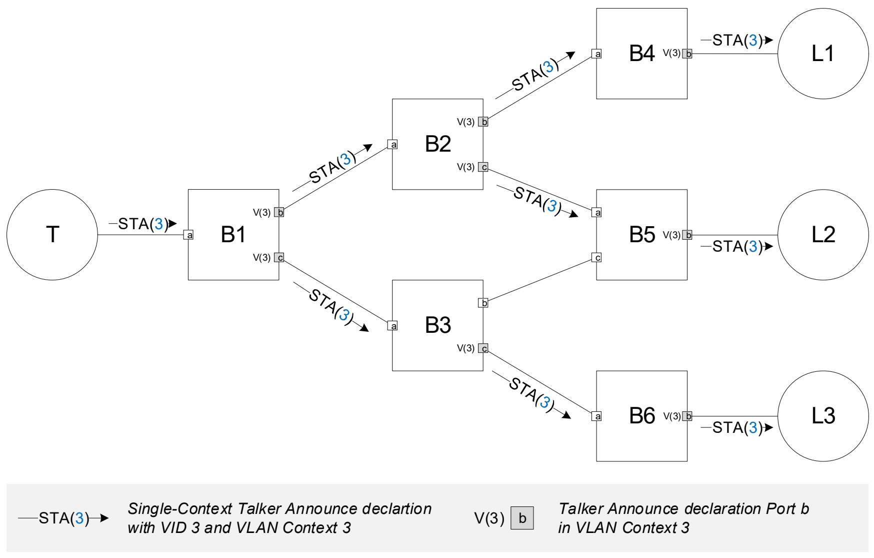

T
B1
B2
B3
B4
B5
B6
L1
L2
L3
STA(3)
V(3)
Single-Context Talker Announce declartion with VID 3 and VLAN Context 3
Talker Announce declaration Port b in VLAN Context 3

Figure Y-2—Single-path stream example: Talker Announcement

As illustrated in Figure Y-3, three Listeners participate in the Single-Context Listener Attachment [item a) in 51.3.5.1], which is propagated along the path previously traversed by the associated Talker Announcement in reversed direction and is subject to Listener Attach merge (51.3.5.3) on both Port B2.a and Port B1.a.

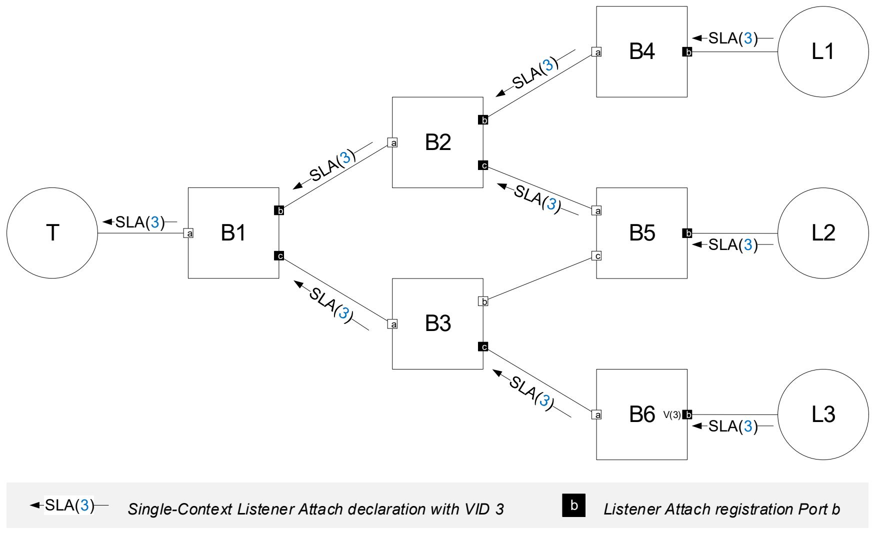

T
B1
B2
B3
B4
B5
B6
L1
L2
L3
SLA(3)
V(3)
Single-Context Listener Attach declaration with VID 3
Listener Attach registration Port b

Figure Y-3—Single-path stream example: Listener Attachment

Figure Y-4 shows the single path established for transmission of the stream from the Talker to each Listener after successful resource allocation.

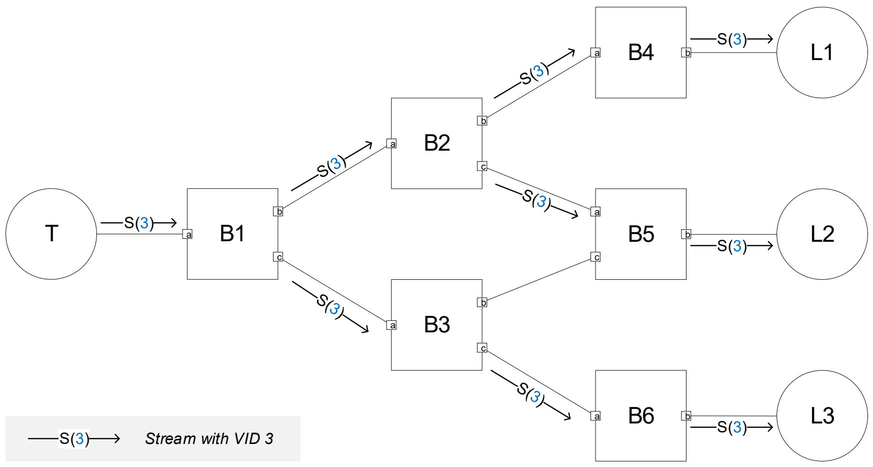


Figure Y-4—Single-path stream example: established stream

Y.2 Multi-path stream example 1

This subclause gives an example of resource allocation for transmission of a Compound Stream in a network analogous to that described in Annex C.1 of IEEE Std 802.1CB. As shown in Figure Y-5, three VLAN topologies representing an instance of redundant trees constitute a Redundancy Context used in the resource allocation. In this example, FRER functionality is provided only in three FRER-capable end stations.

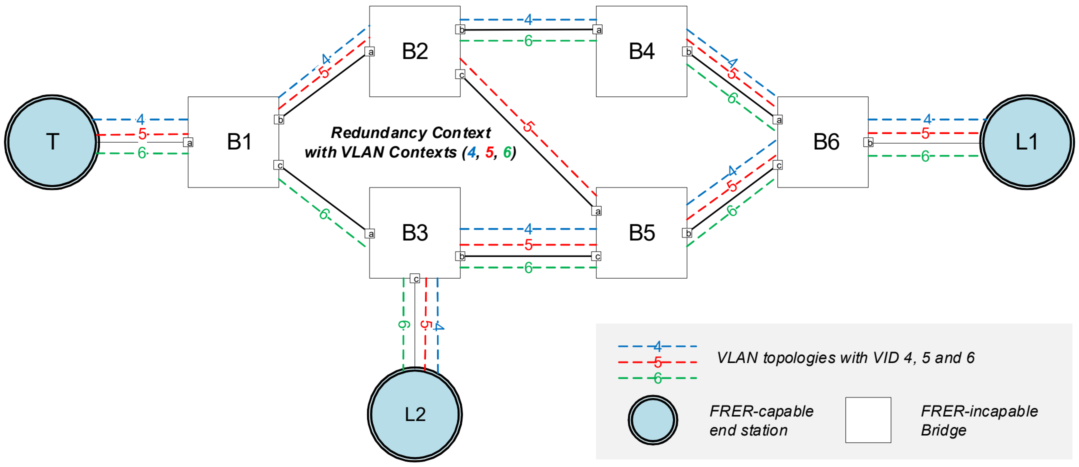

T
B1
B2
B3
B4
B5
B6
L1
L2
Redundancy Context with VLAN Contexts (4, 5, 6)
4
5
6
VLAN topologies with VID 4, 5 and 6
FRER-capable end station
FRER-incapable Bridge

Figure Y-5—Multi-path stream example 1: topology and VLAN configuration

As illustrated in Figure Y-6, the FRER-capable Talker initiates a Multi-Context Talker Announcement [item b) in 51.3.4.1.1] with three separate Talker Announce declarations, each using a different VID and VLAN Context in the Redundancy Context to represent a Member Stream split from the Compound Stream. Since there are no FRER-capable Bridges in the network, each of these Talker Announce declarations is propagated and processed by the Bridges in the network independently.

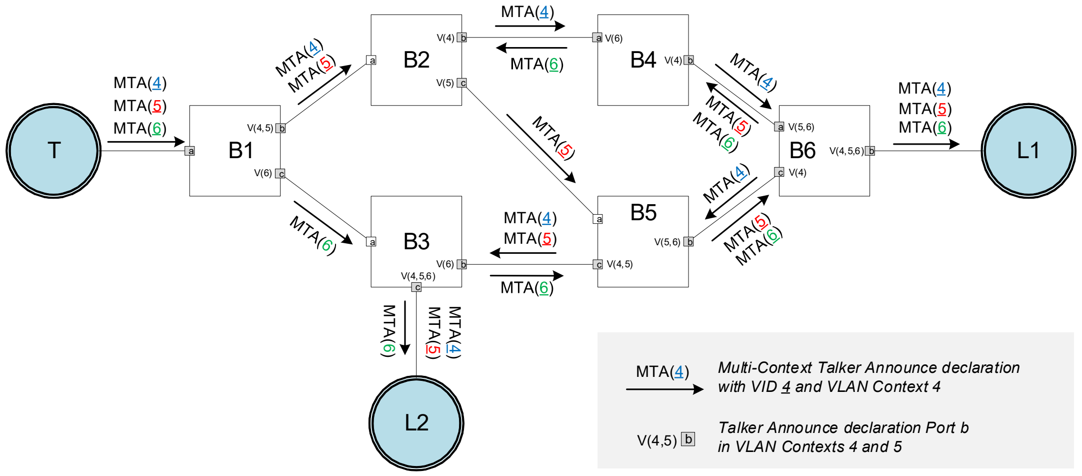

T
B1
B2
B3
B4
B5
B6
L1
L2
MTA(4)
MTA(5)
MTA(6)
V(4,5)
V(6)
V(4)
V(5)
V(5,6)
V(4,5,6)
Multi-Context Talker Announce declaration with VID 4 and VLAN Context 4
Talker Announce declaration Port b in VLAN Contexts 4 and 5

Figure Y-6—Multi-path stream example 1: Talker Announcement

Figure Y-7 illustrates the Multi-Context Listener Attachment associated with the Multi-Context Talker Announcement shown in Figure Y-6. Each Listener makes three Listener Attach declarations, each with the same VID and VLAN Context as received in one of the three Talker Announce registrations. Each Listener Attach declaration made by a Listener is propagated along the path formed by the set of Bridge Ports that contain the associated Taker Announce registration. Listener Attach merge (51.3.5.3) occurs on the Bridge Ports B5.a and B6.a where two Listener Attach registrations propagated from the different Listeners in the same VLAN Context are merged into a joint Listener Attach declaration, indicating the need for the Bridge to multicast the Member Stream received on that Port.

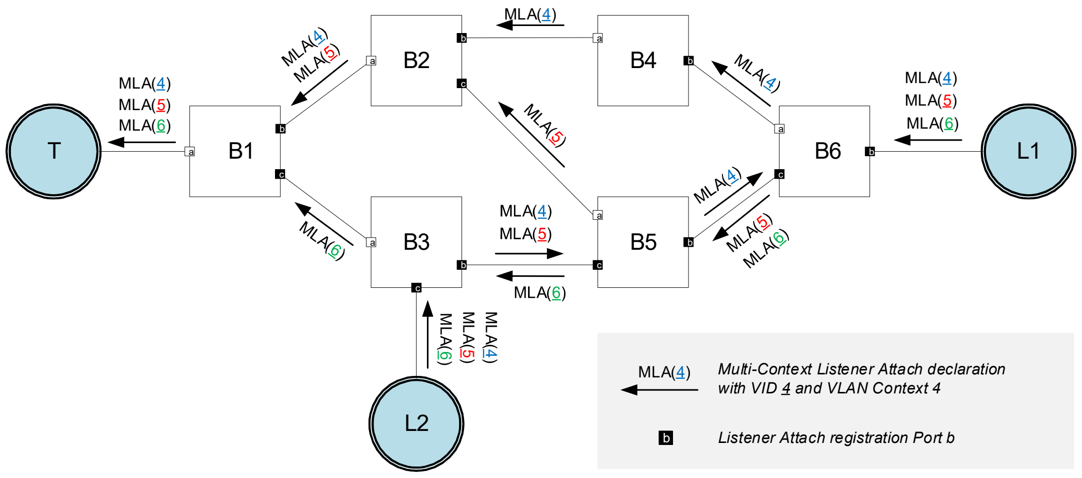

T
B1
B2
B3
B4
B5
B6
L1
L2
MLA(4)
MLA(5)
MLA(6)
Multi-Context Listener Attach declaration with VID 4 and VLAN Context 4
Listener Attach registration Port b

Figure Y-7—Multi-path stream example 1: Listener Attachment

Figure Y-8 shows the paths established and the FRER functions configured for redundant transmission of the Compound Stream with three Member Streams after successful resource allocation.

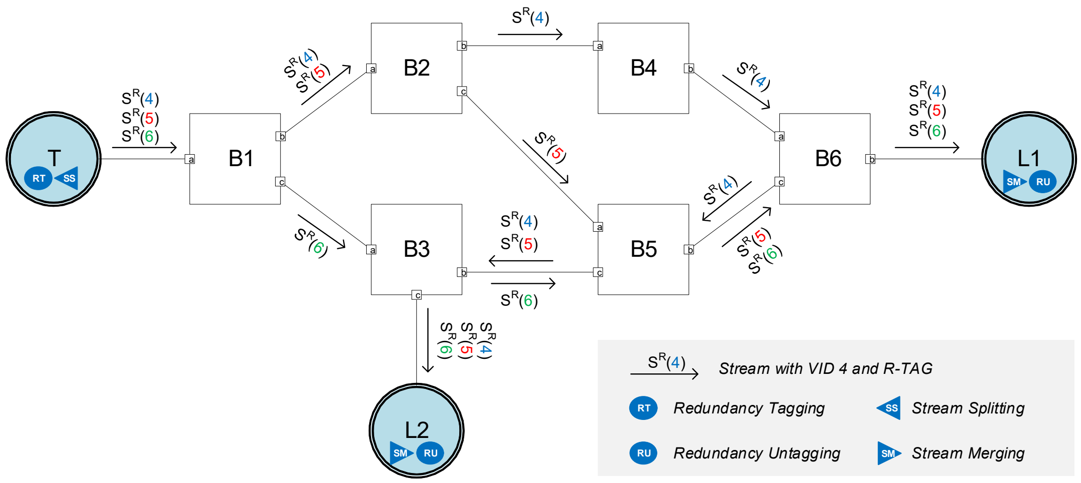


Figure Y-8—Multi-path stream example 1: established stream

Y.3 Multi-path stream example 2

This subclause gives an example of resource allocation for transmission of a Compound Stream in a network analogous to that described in Annex C.2 of IEEE Std 802.1CB. As shown in Figure Y-9, this example has the same VLAN configuration as that in Figure Y-5, and FRER functionality is provided only in some of the Bridges. Three FRER-capable Bridges B1, B5 and B6 are placed at the edge of the network as proxies to offer FRER functionality to the FRER-incapable end station. Additionally, another FRER-capable Bridge B2 is placed within the network to split the VLAN topologies, in conjunction with B1, into three disjoint paths.

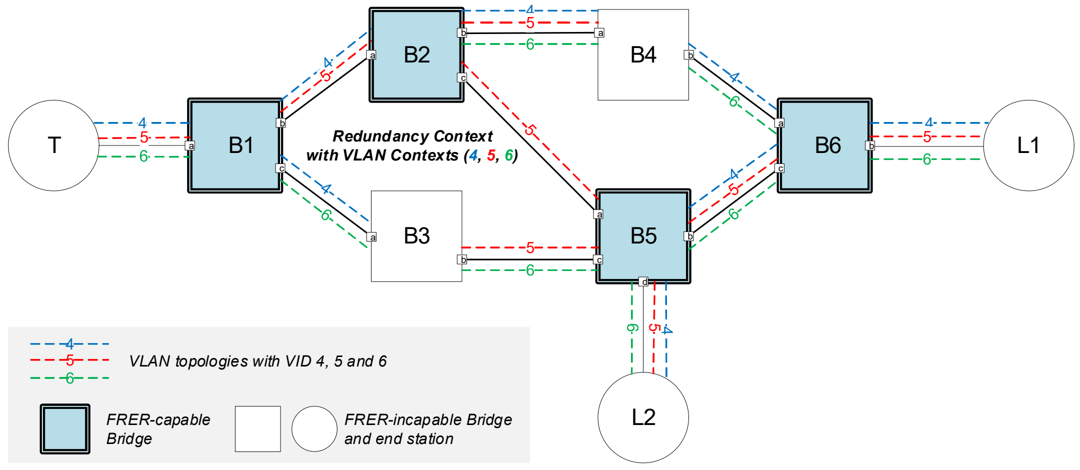

T
B1
B2
B3
B4
B5
B6
L1
L2
4
5
6
Redundancy Context with VLAN Contexts (4, 5, 6)
VLAN topologies with VID 4, 5 and 6
FRER-capable Bridge
FRER-incapable Bridge and end station

Figure Y-9—Multi-path stream example 2: topology and VLAN configuration

As illustrated in Figure Y-10, the FRER-incapable Talker initiates a Multi-Context Talker Announcement with a single Talker Announce declaration that contains a value of FALSE in RTagStatus (51.5.3.8.1) and all of the three VLAN Contexts in (51.5.3.9), to request the network to convert the stream to a Compound Stream for redundant transmission in the three indicated VLAN Contexts. The Talker Announcement is first adjusted by B1 to set the RTagStatus to TURE, and then is propagated by B1 and B2, which are determined to be the locations for stream splitting (51.3.8.4), to two Ports with different VLAN Contexts. Talker Announce merge (51.3.4.3) is applied to the Talker Announcement on the Bridge Ports B5.b, B5.d and B6.b, indicating the location for stream merging (51.3.8.5). The Talker Announcement is adjusted on both B5.d and B6.b to set the RTagStatus to FALSE, indicating the locations for redundancy untagging (51.3.8.3), as these Ports are located where the three VLAN Contexts converge.

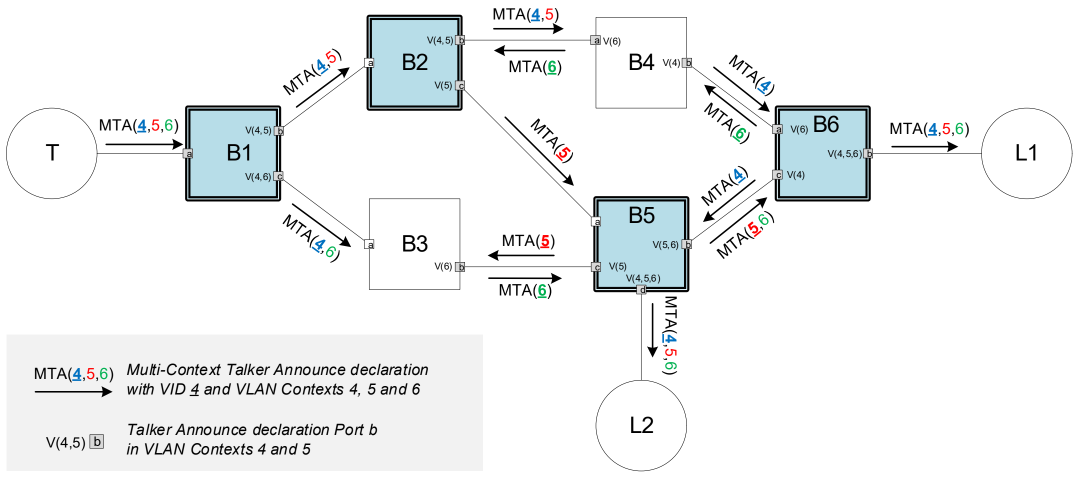

MTA(4,5)
MTA(6)
MTA(4,5)
MTA(4,5,6)
MTA(5)
MTA(4)
MTA(6)
MTA(4,5,6)
MTA(4)
MTA(5,6)
MTA(4,6)
MTA(5)
MTA(6)
MTA(4,5,6)
T
B1
B2
B3
B4
B5
B6
L1
L2
V(4,5)
V(4,6)
V(5)
V(6)
V(4)
V(4,5,6)
V(5,6)
MTA(4,5,6) Multi-Context Talker Announce declaration with VID 4 and VLAN Contexts 4, 5 and 6
V(4,5) b Talker Announce declaration Port b in VLAN Contexts 4 and 5

Figure Y-10—Multi-path stream example 2: Talker Announcement

Figure Y-11 illustrates the Multi-Context Listener Attachment associated with the Multi-Context Talker Announcement shown in Figure Y-10. Each Listener makes a single Listener Attach declaration with the same VLAN Contexts as received in the Talker Announce registration and a desired VID chosen from among these VLAN Context IDs. Similar to that in Figure Y-7, the Listener Attach declaration on both B5.a and B6.a is merged from the two Listener Attach registrations propagated from different Listeners in the same VLAN Context. Additionally in this example, Listener Attach merge (51.3.5.3) occurs on the Bridge Ports B2.a and B1.a where two Listener Attach registrations containing the split VLAN Contexts are merged into a single Listener Attach declaration with joint VLAN Contexts, as a reserve operation of splitting the Talk Announcement previously performed on the same Port. Note that the Listener Attach declaration on B2.a and B1.a takes a VID that is contained in the VLAN Context(s) of both of the Listener Attach registrations merging into that Listener Attach declaration, because the multicast forwarding mechanism used in stream splitting by FRER-capable Bridges requires stream transmission Ports to be in the member set for a common VID.

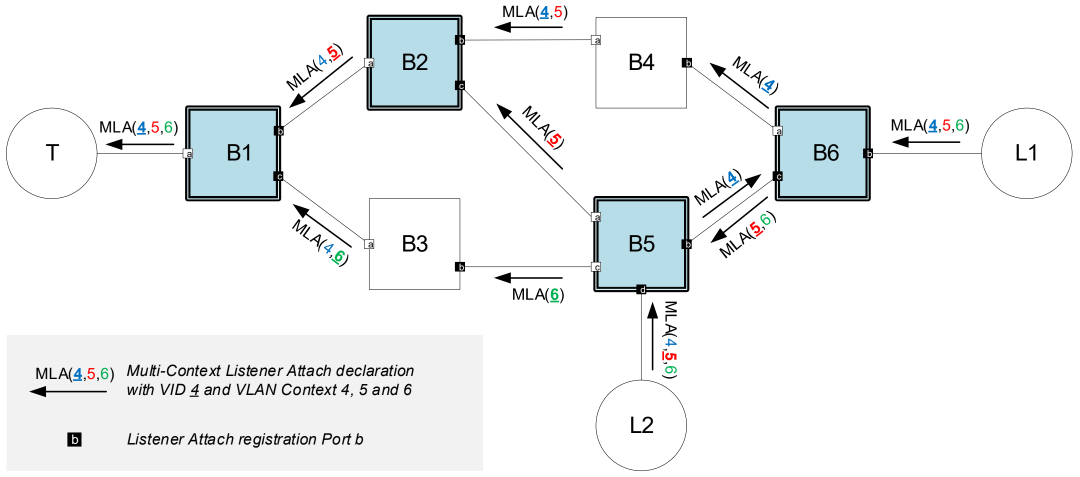

MLA(4,5)
MLA(4,5)
MLA(4,5,6)
MLA(5)
MLA(4)
MLA(4,5,6)
MLA(4)
MLA(5,6)
MLA(4,6)
MLA(6)
MLA(4,5,6)
T
B1
B2
B3
B4
B5
B6
L1
L2
MLA(4,5,6) Multi-Context Listener Attach declaration with VID 4 and VLAN Context 4, 5 and 6
b Listener Attach registration Port b

Figure Y-11—Multi-path stream example 2: Listener Attachment

Figure Y-12 shows the paths established and the FRER functions configured for redundant transmission of the Compound Stream with three Member Streams after successful resource allocation.

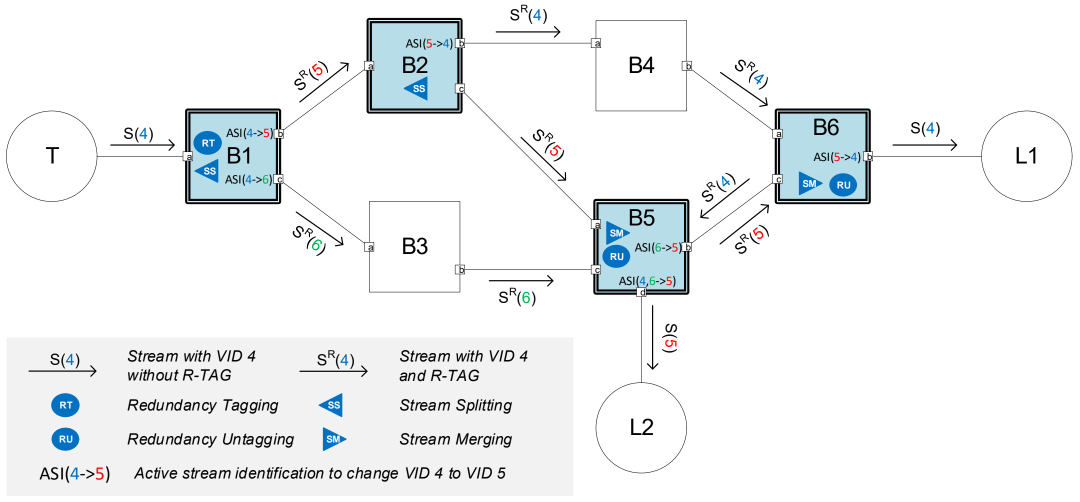


Figure Y-12—Multi-path stream example 2: established stream

Y.4 Multi-path stream example 3

This subclause gives an example of resource allocation for transmission of a Compound Stream in a network analogous to that described in Annex C.3 of IEEE Std 802.1CB. As shown in Figure Y-13, FRER functionality is provided in each end station and some of the Bridges. Four FRER-capable Bridges are connected as a ring, each acting as both a stream splitting point and a stream merging point, to provide protection against multiple simultaneous failures. A Redundancy Context with two VLAN topologies is configured in the network. Note that in this particular example, each of those Bridge Ports marked as VLAN topology termination point is required to be excluded from the membership for the indicated VLAN and to have ingress filtering (8.6.2) enabled for that VLAN, in order to ensure no more than one Talker Announce registration is propagated with the same VLAN Context by a Bridge.

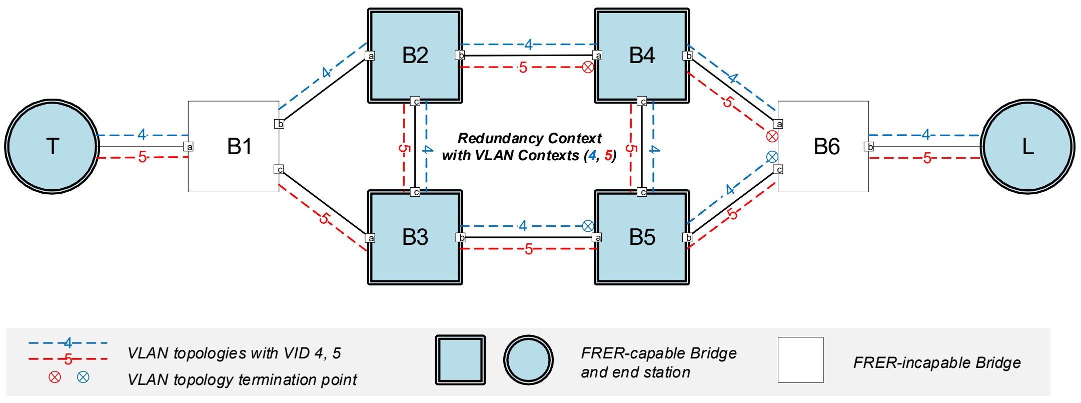

T
B1
B2
B3
B4
B5
B6
L
4
5
Redundancy Context with VLAN Contexts (4, 5)
VLAN topologies with VID 4, 5
VLAN topology termination point
FRER-capable Bridge and end station
FRER-incapable Bridge

Figure Y-13—Multi-path stream example 3: topology and VLAN configuration

As illustrated in Figure Y-14, two Talker Announce declarations initiated by the FRER-capable Talker, each with a distinct VLAN Context, are propagated by B1 separately. According to the VLAN configuration, each FRER-capable Bridge propagates the Talker Announce registration on Port a with a single VLAN Context to the other two Ports, and meanwhile merges on Port b two Talker Announce registrations propagated from the other two Ports into a joint Talker Announce declaration with both VLAN contexts.

Note that on each Bridge Port that is excluded from the member set for one VLAN Context, ingress blocking (51.3.4.1.3) ensures that the Talker Announce registration received with two VLAN Contexts is propagated only with the other VLAN Context for which the Port in the member set.

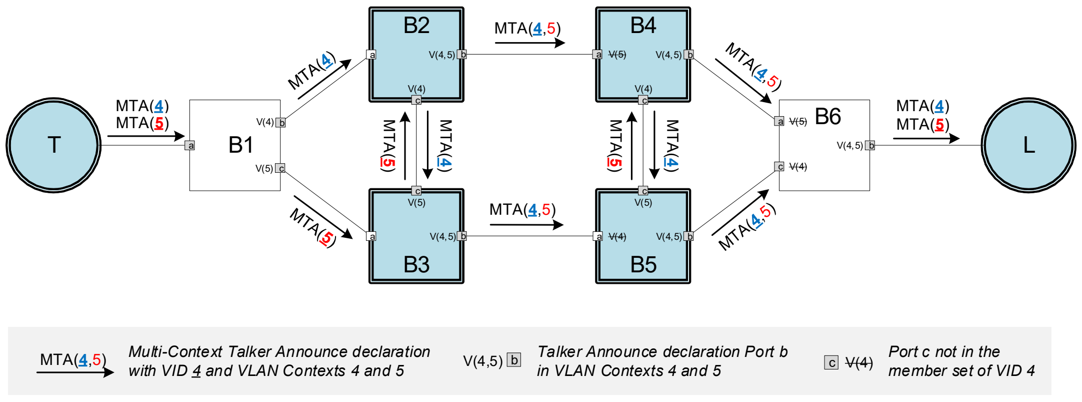

T
B1
B2
B3
B4
B5
B6
L
MTA(4)
MTA(5)
MTA(4,5)
V(4)
V(5)
V(4,5)
~~V(5)~~
~~V(4)~~
MTA(4,5) Multi-Context Talker Announce declaration with VID 4 and VLAN Contexts 4 and 5
V(4,5) b Talker Announce declaration Port b in VLAN Contexts 4 and 5
c ~~V(4)~~ Port c not in the member set of VID 4

Figure Y-14—Multi-path stream example 3: Talker Announcement

Figure Y-15 illustrates the Multi-Context Listener Attachment associated with the Multi-Context Talker Announcement shown in Figure Y-14. The FRER-capable Listener makes two Listener Attach declarations, each with a different VLAN Context. Listener Attach merge occurs on Port a of each FRER-capable Bridge, resulting in a joint Listener Attach declaration with the same VLAN Context as that contained in the associated Talker Announce registration on the same Port. Finally, B1 passes two Listener Attach declarations, each with a different VLAN Context, to the Talker.

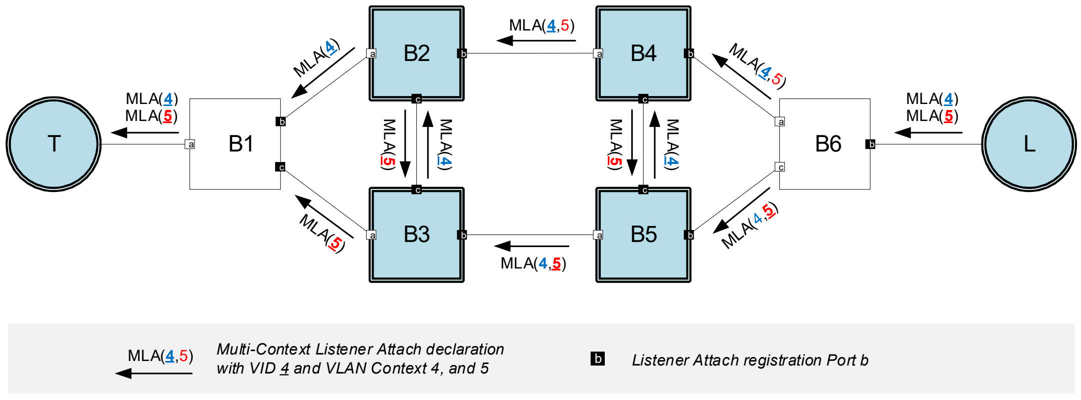

T
B1
B2
B3
B4
B5
B6
L
MLA(4)
MLA(5)
MLA(4,5)
MLA(4,5) Multi-Context Listener Attach declaration with VID 4 and VLAN Context 4, and 5
b Listener Attach registration Port b

Figure Y-15—Multi-path stream example 3: Listener Attachment

Figure Y-16 shows the paths established and the required FRER functions configured for redundant transmission of the Compound Stream with two Member Streams after successful resource allocation

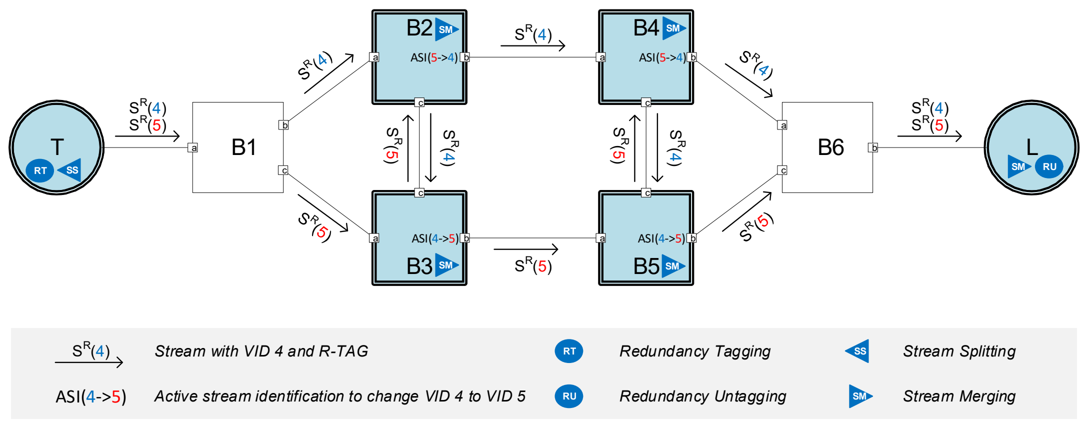


Figure Y-16—Multi-path stream example 3: established stream

Y.5 Example of status notification and failure diagnosis

This subclause uses the example network of Y.4 to illustrate the status notification and failure diagnosis mechanisms in resource allocation. As shown in Figure Y-17, it is assumed that the Multi-Context Talker Announcement encounters resource problems on three Bridge Ports.

The problem on Port B1.b causes the Talker Announce declaration on that Port to change to the Talker Announce status of Announce Fail [item b) in 51.3.4.2] and to attach the failure information about the problem on that Port to VLAN Context 4. The failure information generated on B1.b remains attached to VLAN Context 4 in the subsequent propagation of the Talker Announcement across the network toward the Listener, regardless of the situations encountered on any downstream Port. Similarly, the problems on B3.b and B4.b lead to Announce Fail in the Talker Announce declarations on those Ports. The failure information of VLAN Context 5 is generated on B3.b, and is later merged along with that of VLAN Context 4 to the Talker Announce declarations on B4.b and B5.b. The reason for the Announce Success status on both B2.b and B5.b is that the Talker Announce declaration merges one Talker Announce registration propagated as Announce Success, meaning that a reservation can be made for a stream on a merged path as long as at least one of the paths merging into that merged path is available for reservation.

In the end, the Listener receives two Talker Announce declarations, which indicate that at least one path in the network can be reserved for transmission of the Compound Stream as indicated by the Announce Success status of one Talker Announce registration, even though there are problems detected at the locations associated with both VLAN topologies as indicated in the supplied failure information.

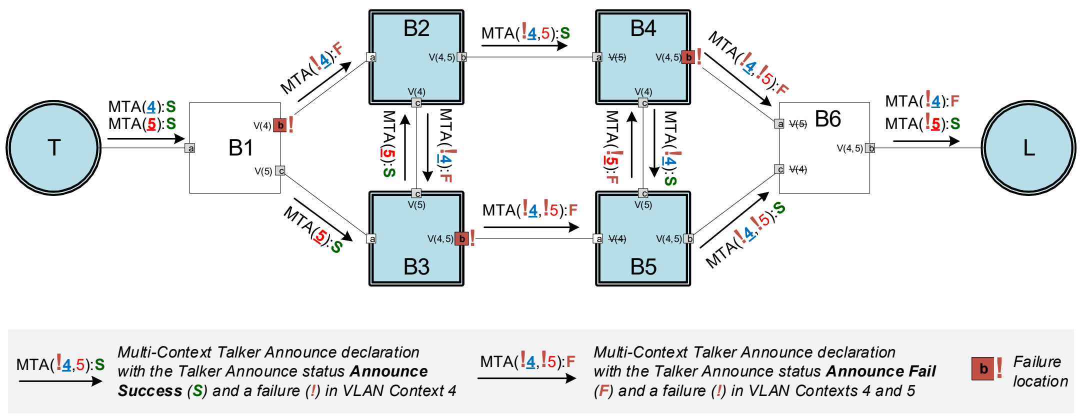

T
B1
B2
B3
B4
B5
B6
L
MTA(4):S
MTA(5):S
MTA(!4):F
MTA(5):S
MTA(!4,5):S
MTA(!4,!5):F
MTA(!4,!5):F
MTA(!4,!5):S
MTA(5):S
MTA(!4):F
MTA(!5):F
MTA(!4):S
MTA(!4):F
MTA(!5):S
Multi-Context Talker Announce declaration with the Talker Announce status Announce Success (S) and a failure (!) in VLAN Context 4
Multi-Context Talker Announce declaration with the Talker Announce status Announce Fail (F) and a failure (!) in VLAN Contexts 4 and 5
Failure location

Figure Y-17—Example Talker Announce status and failure information

Figure Y-18 illustrates the Multi-Context Listener Attachment associated with the Talker Announcement shown in Figure Y-17. The Listener makes two Listener Attach declarations with Attach Ready contained in the Listener Attach status (51.3.5.4), indicating it is ready to receive both Member Streams. As the Listener Attachment is propagated along each stream path back to the Talker, a corresponding reservation is made on each Port for a Listener Attach registration with the Attach Ready status and its associated Talker Announce declaration with the Announce Success status. A Listener Attach registration for which a reservation cannot be made is propagated as Attach Fail. On each Bridge Port where Listener Attach merge occurs in this example, i.e., Port a or each FRER-capable Bridge, the Listener Attach declaration is made with Attach Ready because one of the two Listener Attach registrations being merged on that Port is propagated as Attach Ready. Note that, on those Ports with a terminated VLAN, such as B6.a and B6.c, the Listener Attach declaration still contains the terminated VLAN (e.g., VLAN Context 5 on B6.a) in order to match the VLAN Contexts contained in the associated Talker Announce registration on the same Port, but needs to set the corresponding Path status to Unresponsive Path to indicate that Listener Attach declaration does not originate from that VLAN Context.

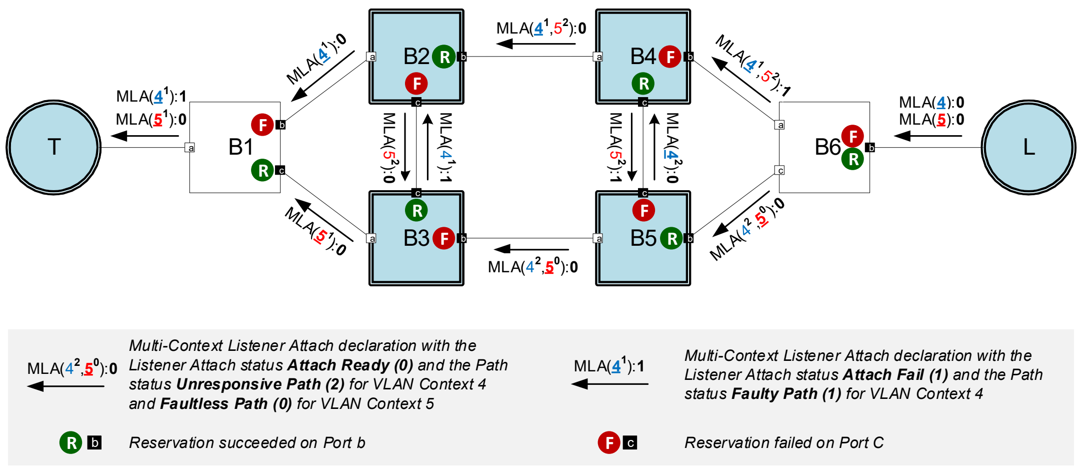

T
B1
B2
B3
B4
B5
B6
L
MLA(4¹):1
MLA(5¹):0
MLA(4¹):0
MLA(5¹):0
MLA(4¹,5²):0
MLA(4¹,5²):1
MLA(4²,5⁰):0
MLA(4²,5⁰):0
MLA(4):0
MLA(5):0
MLA(5²):0
MLA(4¹):1
MLA(5²):1
MLA(4²):0
Multi-Context Listener Attach declaration with the Listener Attach status Attach Ready (0) and the Path status Unresponsive Path (2) for VLAN Context 4 and Faultless Path (0) for VLAN Context 5
Multi-Context Listener Attach declaration with the Listener Attach status Attach Fail (1) and the Path status Faulty Path (1) for VLAN Context 4
Reservation succeeded on Port b
Reservation failed on Port C

Figure Y-18—Example Listener Attach status and Path status

Figure Y-19 shows the single path established and the FRER functions configured for the Compound Stream in the case of three problems in the network.

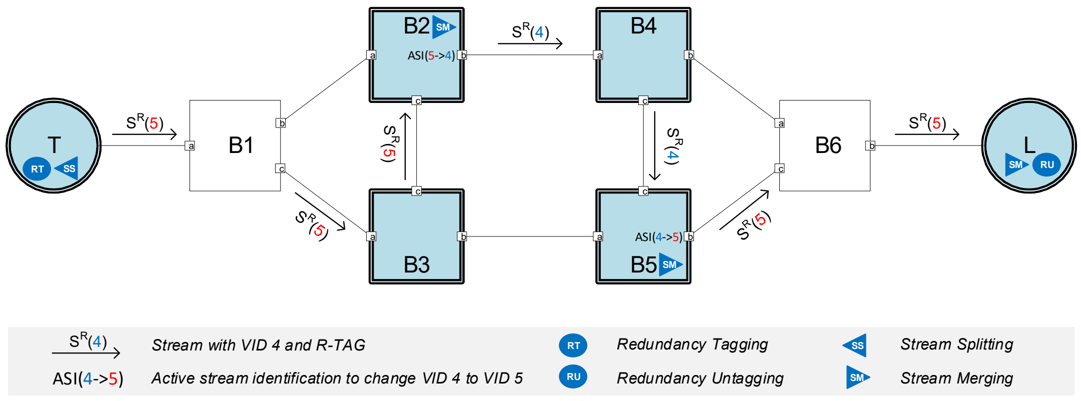


Figure Y-19—Example partially established Compound Stream

Annex Z

(informative)

Commentary

This is a temporary Annex intended to record issues and their resolutions as the project proceeds. It will be removed prior to SA ballot.

Z.1 Objectives and Non-objectives

Z.1.1 Objectives

a) Support for both LRP native and proxy/slave systems
b) Support for use of either ECP or TCP as LRP-DT mechanism (TCP is required for RAP proxy)
c) Restricted to use of “Talker Uni” (see presentation: dd-finn-RAP-LRP-MSRP-Qcc-0918-v03.pdf). Limit RAP/LRP capabilities to things that can be done with a peer-to-peer implementation.
d) Support for various transmission selection algorithms (CBS, ATS, CQF, TAS, SP, etc.))
e) Support for reservation of streams transmitted over redundant paths, e.g. using 802.1CB-FRER
f) Backward compatibility with MSRP

Z.1.2 Non-objectives

a) using RAP to carry “third-party UNI” data between CUC and “fully-service” CNC
b) using RAP also as the protocol between a RAP proxy and a RAP slave

Z.2 LRP-related open issues

Contributions are welcome for the following issues related to LRP.

Z.2.1 Target link monitoring

Target link is the physical link between target ports through which data streams are transmitted. However, LRPDUs that carry control plane data could be exchanged along a path other than the target link, for examples, when TCP is used to connect to a RAP proxy system. In such cases, a target port failure will not cause a hello timeout. Thus, target link monitoring would be essential for the operation of RAP.

The localTargetPortOper variable defined in 51.7.4.9 can be used for purpose of target link monitoring, and its value could be controlled by a specific connectivity monitoring and failure detection mechanism such as CFM.

It is intended to provide an informative description for this issue in a subclause in 99.3.

Z.2.2 Use of LRP TCP mechanism

What needs to be specified regarding the use of TCP.

Z.2.3 completeListTimerReset

What range and default value should be used?

Z.2.4 Handling of neighborRegistrarOverflow

The support for this LRP feature has been removed from P802.1 Qdd D0.6.

Z.2.5 Maximum Record Size

The maximum record size is determined by the LRP-DT mechanism in use. It seems to the editor that LRP does not provide information to its application, either via an indication primitive or a managed object, of which LRP-DT mechanism has been chosen to transport LRPDUs, if the application indicated support for both ECP and TCP at requesting a Portal creation.

Z.3 Support for queuing/transmission functions

Each of the following mechanism corresponds to a potential RA class template for use with RAP.

Z.3.1 Asynchronous Traffic Shaping (ATS)

contribution:

— Reservation with RAP for Time-Sensitive Streams using ATS (dd-chen-RAP-ATS-0619-v02.pdf)

Z.3.2 Cyclic Queuing and Forwarding (CQF)

contribution:

— Support for µStream Aggregation in RAP (dd-chen-flow-aggregation-0119-v03.pdf)
— Cyclic Queuing and Forwarding, Paternoster, and TSpecs (dd-finn-CQF-and-shaping-0120-v01.pdf)

Z.3.3 Enhancements for Scheduled Traffic (EST)

contribution:

— Distributed Stream Configuration in Industrial Automation (60802-dorr-distributedConfiguration-0319-v01.pdf)

Z.3.4 Strict Priority for TSN

contribution:

— Bridge-Local Guaranteed Latency with Strict Priority Scheduling (dd-grigorjew-strict-priority-latency-0320-v02.pdf)

Z.4 Backward compatibility with MSRP

The editor is unclear what goals relating to backward compatibility with MSRP are to be met by RAP. Contributions are needed.

Per resolution of D0.5 ballot comment #22:

Issues to be discussed:

a) Does RAP need to support compatibility with MSRP, including MSRP Tspec? If yes, what level of compatibility with MSRP should be provided by RAP.
b) Does any TSA other than CBS require use of MSRP Tspec (interval based) instead of Token-bucket Tspec in RAP?

MSRP Tspec has the following “constraints”:

— Only used to describe the stream traffic output from the Talker, not intended to be adjustable by bridges while propagation through the network.
— Containing in "Max frame size" parameter a payload size, which can be ambiguous, e.g. whether a 802.1CB R-Tag is accounted.

There have been several contributions that show how to use Token-bucket Tspec in RAP such as for SP and ATS, but still lack of contributions as to the use of MSRP Tspec in RAP for either backwards compatibility or use by specific TSAs.

Z.5 Traffic Specification

Z.5.1 TSpec types and parameters

Contribution:

— Traffic Specification Types in Qdd (dd-grigorjew-tspec-types-0721-v01.pdf)

(see open issues listed in the presentation)

Z.5.2 Support for updating TSpec at Talkers

See comment #17 in the disposition of TG ballot comments on D0.4 (802-1Qdd-d0-4-dis-v01.pdf).

Further discussion and contribution is needed for this potential feature.

Z.6 Support for stream identifications

See comment #33 in the disposition of TG ballot comments on D0.4 (802-1Qdd-d0-4-dis-v01.pdf).

Z.7 Error detection and handling

Z.7.1 Validation of Talker Announce registrations

The RAP Propagator needs to validate each received Talker Announce registration for detection and appropriate handling of errors that may be caused by the following reasons:

a) Looping in the network topology (e.g., in the progress of spanning tree reconfiguration).
b) Violation of uniqueness rules in use of Stream ID, Stream DA, etc.
c) Misbehaving Bridge/end stations.

For Single-Context TA (non-redundant stream), the following error conditions may occur:

```
(newTaReg.StreamId == oldTaReg.StreamId && newTaReg.portRef != oldTaReg.portRef)
||
(newTaReg.StreamId == oldTaReg.StreamId && newTaReg.portRef == oldTaReg.portRef
&& (newTaReg.DA != oldTaReg.DA || newTaReg.VID != oldTaReg.VID ||
newTaReg.Priority != oldTaReg.Priority))
```

Error cases for multiple-context TA (redundant streams) need to be further considered. Validation of LA registrations is also necessary.

Z.8 Stream protection inside RA class domain

Per comment #11 on D0.6, there exists a gap relating to stream protection against malicious behaviors inside an RA class domain. Currently, RAP makes use of Dynamic Reservation Entries (8.8.7) in FDB to control forwarding of streams based on their reservation status. However, this alone cannot prevent unauthorized access to stream queues, if a Talker within an RA class domain intentionally starts stream transmission prior to initiating the reservation process. The reason for this gap is that there is no FDB function that can filter the frames that carry a stream priority but unknown DA/VID.