JggDvj20050416
2007-08-13
(August 13, 2007)

DVJ Perspective on: Timing and synchronization for time-sensitive applications in bridges local area networks

Draft 0.727

Contributors:
See page xx.

Abstract: This working paper provides background and introduces possible higher level concepts for the development of Audio/Video bridges (AVB).
Keywords: audio, visual, bridge, Ethernet, time-sensitive

IEEE Standards documents are developed within the IEEE Societies and the Standards Coordinating Committees of the IEEE Standards Association (IEEE-SA) Standards Board. The IEEE develops its standards through a consensus development process, approved by the American National Standards Institute, which brings together volunteers representing varied viewpoints and interests to achieve the final product. Volunteers are not necessarily members of the Institute and serve without compensation. While the IEEE administers the process and establishes rules to promote fairness in the consensus development process, the IEEE does not independently evaluate, test, or verify the accuracy of any of the information contained in its standards.

Use of an IEEE Standard is wholly voluntary. The IEEE disclaims liability for any personal injury, property or other damage, of any nature whatsoever, whether special, indirect, consequential, or compensatory, directly or indirectly resulting from the publication, use of, or reliance upon this, or any other IEEE Standard document.

The IEEE does not warrant or represent the accuracy or content of the material contained herein, and expressly disclaims any express or implied warranty, including any implied warranty of merchantability or fitness for a specific purpose, or that the use of the material contained herein is free from patent infringement. IEEE Standards documents are supplied "**AS IS**."

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. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation. When a document is more than five years old and has not been reaffirmed, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

In publishing and making this document available, the IEEE is not suggesting or rendering professional or other services for, or on behalf of, any person or entity. Nor is the IEEE undertaking to perform any duty owed by any other person or entity to another. Any person utilizing this, and any other IEEE Standards document, should rely upon the advice of a competent professional in determining the exercise of reasonable care in any given circumstances.

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Comments on standards and requests for interpretations should be addressed to:

Secretary, IEEE-SA Standards Board
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
USA.

Note: Attention is called to the possibility that implementation of this standard may require use of subject matter covered by patent rights. By publication of this standard, no position is taken with respect to the existence or validity of any patent rights in connection therewith. The IEEE shall not be responsible for identifying patents for which a license may be required by an IEEE standard or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Editors' Foreword

Comments on this draft are encouraged. **PLEASE NOTE: All issues related to IEEE standards presentation style, formatting, spelling, etc. should be addressed, as their presence can often obfuscate relevant technical details.**

By fixing these errors in early drafts, readers can devote their valuable time and energy 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.

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 had a policy of considering ballot 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. Non-members are advised that the email lists exist primarily to allow the members of the working group to develop standards, and are not a general forum.

Comments on this document may be sent to the 802.1 email reflector, to the editors, or to the Chairs of the 802.1 Working Group and Interworking Task Group.

This draft was prepared by:

David V James
JGG
3180 South Court
Palo Alto, CA 94306
+1.650.494.0926 (Tel)
+1.650.954.6906 (Mobile)
Email: dvj@alum.mit.edu

Chairs of the 802.1 Working Group and Audio/Video Bridging Task Group:.

Michael Johas Teener
Chair, 802.1 Audio/Video Bridging Task
Broadcom Corporation
3151 Zanker Road
San Jose, CA
95134-1933
USA
+1 408 922 7542 (Tel)
+1 831 247 9666 (Mobile)
Email:mikejt@broadcom.com

Tony Jeffree
Group Chair, 802.1 Working Group
11A Poplar Grove
Sale
Cheshire
M33 3AX
UK
+44 161 973 4278 (Tel)
+44 161 973 6534 (Fax)
Email: tony@jeffree.co.uk

Introduction to IEEE Std 802.1AS™

(This introduction is not part of P802.1AS, IEEE Standard for Local and metropolitan area networks—Timing and synchronization for time-sensitive applications in bridged local area networks.)

This standard specifies the protocol and procedures used to ensure that the synchronization requirements are met for time sensitive applications, such as audio and video, across bridged and virtual bridged local area networks consisting of LAN media where the transmission delays are fixed and symmetrical; for example, IEEE 802.3 full duplex links. This includes the maintenance of synchronized time during normal operation and following addition, removal, or failure of network components and network reconfiguration. The design is based on concepts developed within the IEEE Std 1588, and is applicable in the context of IEEE Std 802.1D and IEEE Std 802.1Q.

Synchronization to an externally provided timing signal (e.g., a recognized timing standard such as UTC or TAI) is not part of this standard but is not precluded.

Version history

Version	Date	Edits by	Comments
0.082	2005-04-28	DVJ	Updates based on 2005Apr27 meeting discussions
0.085	2005-05-11	DVJ	– Updated list-of-contributors, page numbering, editorial fixes.
0.088	2005-06-03	DVJ	– Application latency scenarios clarified.
0.090	2005-06-06	DVJ	– Misc. editorials in bursting and bunching annex.
0.092	2005-06-10	DVJ	– Extensive cleanup of Clause 5 subscription protocols.
0.121	2005-06-24	DVJ	– Extensive cleanup of clock-synchronization protocols.
0.127	2005-07-04	DVJ	– Pacing descriptions greatly enhanced.
0.200	2007-01-23	DVJ	Removal of non time-sync related information, initial layering proposal.
0.207	2007-02-01	DVJ	Updates based on feedback from Monterey 802.1 meeting. – Common entity terminology; Ethernet type code expanded.
0.216	2007-02-17	DVJ	Updates based on feedback from Chuck Harrison: – linkDelay based only on syntonization to one's neighbor. – Time adjustments based on observed grandMaster rate differences.
0.224	2007-03-03	DVJ	Updates for whiplash free PLL cascading.
0.230	2007-03-05	DVJ	Major changes: – simplified back-interpolation – first iteration on an Ethernet-PON interface – client-level clock-master and clock-slave interfaces defined
0.243	2007-04-20	DVJ	– Revised GrandSync entity illustrations – General cleanup
0.708	2007-05-30	DVJ	– Simulation results provided within an annex – Extensive code revisions for simplicity & clarity. – Interpolation better described.
0.724	2007-08-11	DVJ	Clause 5 updates for clarity. Clause 6 updates include initialize-phase processing.

Formats

In many cases, readers may elect to provide contributions in the form of exact text replacements and/or additions. To simplify document maintenance, contributors are requested to use the standard formats and provide checklist reviews before submission. Relevant URLs are listed below:

General: http://grouper.ieee.org/groups/msc/WordProcessors.html
Templates: http://grouper.ieee.org/groups/msc/TemplateTools/FrameMaker/
Checklist: http://grouper.ieee.org/groups/msc/TemplateTools/Checks2004Oct18.pdf

TBDs

Further definitions are needed in the following areas:

a) Should low-rate *leapSeconds* occupy space in timeSync frames, if this information rarely changes?
b) What other (than *leapSeconds*) low-rate information should be transferred between stations?
c) When the grandmaster changes, how should the new grandmaster affect change:
    1) Transition immediately to the rate of its reference clock.
    2) Transition slowly (perhaps 1ppm/s) between previous and reference clock rates.

Contents

Contribution from: dvj@alum.mit.edu.

List of figures

List of tables

DVJ Perspective on: Timing and synchronization for time-sensitive applications in bridges local area networks

1. Overview

1.1 Scope

This draft specifies the protocol and procedures used to ensure that the synchronization requirements are met for time sensitive applications, such as audio and video, across bridged and virtual bridged local area networks consisting of LAN media where the transmission delays are fixed and symmetrical; for example, IEEE 802.3 full duplex links. This includes the maintenance of synchronized time during normal operation and following addition, removal, or failure of network components and network reconfiguration. It specifies the use of IEEE 1588 specifications where applicable in the context of IEEE Std 802.1D and IEEE Std 802.1Q. Synchronization to an externally provided timing signal (e.g., a recognized timing standard such as UTC or TAI) is not part of this standard but is not precluded.

1.2 Purpose

This draft enables stations attached to bridged LANs to meet the respective jitter, wander, and time synchronization requirements for time-sensitive applications. This includes applications that involve multiple streams delivered to multiple endpoints. To facilitate the widespread use of bridged LANs for these applications, synchronization information is one of the components needed at each network element where time-sensitive application data are mapped or demapped or a time sensitive function is performed. This standard leverages the work of the IEEE 1588 WG by developing the additional specifications needed to address these requirements.

1.3 Introduction

1.3.1 Background

Ethernet has successfully propagated from the data center to the home, becoming the wired home computer interconnect of choice. However, insufficient support of real-time services has limited Ethernet's success as a consumer audio-video interconnects, where IEEE Std 1394 Serial Bus and Universal Serial Bus (USB) have dominated the marketplace. Success in this arena requires solutions to multiple topics:

a) Discovery. A controller discovers the proper devices and related streamID/bandwidth parameters to allow the listener to subscribe to the desired talker-sourced stream.

b) Subscription. The controller commands the listener to establish a path from the talker. Subscription may pass or fail, based on availability of routing-table and link-bandwidth resources.

c) Synchronization. The distributed clocks in talkers and listeners are accurately synchronized. Synchronized clocks avoid cycle slips and playback-phase distortions.

d) Pacing. The transmitted classA traffic is paced to avoid other classA traffic disruptions.

This draft covers the “Synchronization” component, assuming solutions for the other topics will be developed within other drafts or forums.

1.3.2 Interoperability

AVB time synchronization interoperates with existing Ethernet, but the scope of time-synchronization is limited to the AVB cloud, as illustrated in Figure 1.1; less-precise time-synchronization services are available everywhere else. The scope of the AVB cloud is limited by a non-AVB capable bridge or a half-duplex link, neither of which can support AVB services.

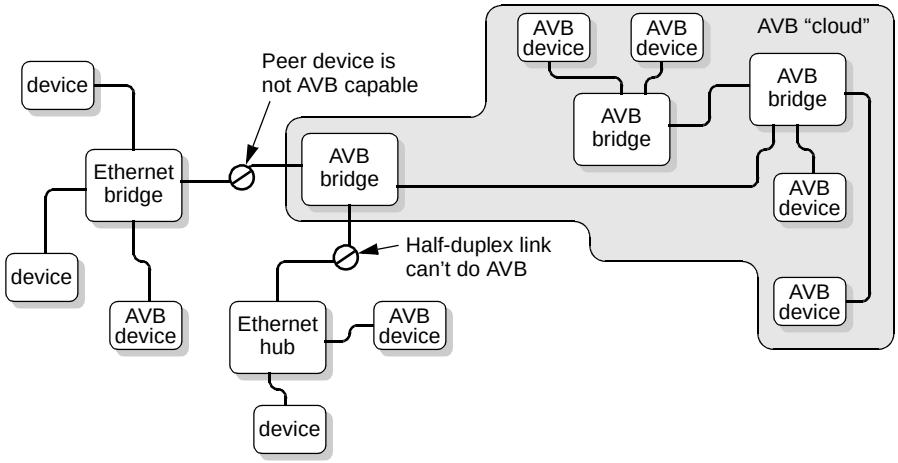

Figure 1.1—Topology and connectivity

Separation of AVB devices is driven by the requirements of AVB bridges to support subscription (bandwidth allocation) and pacing of time-sensitive transmissions, as well as time-of-day clock-synchronization.

1.3.3 Document structure

The clauses and annexes of this working paper are listed below.

- Clause 1: Overview
- Clause 2: References
- Clause 3: Terms, definitions, and notation
- Clause 4: Abbreviations and acronyms
- Clause 5: Architecture overview
- Clause 6: GrandSync operation
- Clause 7: Attached-clock state machines
- Clause 8: Ethernet full duplex (EFDX) state machines
- Clause 9: Wireless state machines
- Clause 10: Ethernet passive optical network (EPON) state machines
- Annex A: Bibliography
- Annex B: Time-scale conversions
- Annex C: Reclocked ClockSync requirements
- Annex D: Simulation results (preliminary)
- Annex E: Bridging to IEEE Std 1394
- Annex F: Time-of-day format considerations
- Annex G: C-code illustrations

2. References

The following documents contain provisions that, through reference in this working paper, constitute provisions of this working paper. All the standards listed are normative references. Informative references are given in Annex A. At the time of publication, the editions indicated were valid. All standards are subject to revision, and parties to agreements based on this working paper are encouraged to investigate the possibility of applying the most recent editions of the standards indicated below.

ANSI/ISO 9899-1990, Programming Language-C.[1,2]

IEEE Std 802.1D-2004, IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges.

[1]Replaces ANSI X3.159-1989

[2]ISO documents are available from ISO Central Secretariat, 1 Rue de Varembe, Case Postale 56, CH-1211, Geneve 20, Switzerland/Suisse; and from the Sales Department, American National Standards Institute, 11 West 42 Street, 13th Floor, New York, NY 10036-8002, USA

3. Terms, definitions, and notation

3.1 Conformance levels

Several key words are used to differentiate between different levels of requirements and options, as described in this subclause.

3.1.1 may: Indicates a course of action permissible within the limits of the standard with no implied preference (“may” means “is permitted to”).

3.1.2 shall: Indicates mandatory requirements to be strictly followed in order to conform to the standard and from which no deviation is permitted (“shall” means “is required to”).

3.1.3 should: An indication that among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others; or that a certain course of action is preferred but not necessarily required; or that (in the negative form) a certain course of action is deprecated but not prohibited (“should” means “is recommended to”).

3.2 Terms and definitions

For the purposes of this working paper, the following terms and definitions apply. The Authoritative Dictionary of IEEE Standards Terms [B2] should be referenced for terms not defined in the clause.

3.2.1 bridge: A functional unit interconnecting two or more networks at the data link layer of the OSI reference model.

3.2.2 clock master: A bridge or end station that provides the link clock reference.

3.2.3 clock slave: A bridge or end station that tracks the link clock reference provided by the clock master.

3.2.4 cyclic redundancy check (CRC): A specific type of frame check sequence computed using a generator polynomial.

3.2.5 grandmaster: Refers to the station that is selected to provide the network time reference.

3.2.6 link: A unidirectional channel connecting adjacent stations (half of a span).

3.2.7 listener: A sink of a stream, such as a television or acoustic speaker.

3.2.8 local area network (LAN): A communications network designed for a small geographic area, typically not exceeding a few kilometers in extent, and characterized by moderate to high data transmission rates, low delay, and low bit error rates.

3.2.9 MAC client: The layer entity that invokes the MAC service interface.

3.2.10 medium (plural: **media):** The material on which information signals are carried; e.g., optical fiber, coaxial cable, and twisted-wire pairs.

3.2.11 medium access control (MAC) sublayer: The portion of the data link layer that controls and mediates the access to the network medium. In this working paper, the MAC sublayer comprises the MAC datapath sublayer and the MAC control sublayer.

3.2.12 network: A set of communicating stations and the media and equipment providing connectivity among the stations.

3.2.13 plug-and-play: The requirement that a station perform classA transfers without operator intervention (except for any intervention needed for connection to the cable).

3.2.14 protocol implementation conformance statement (PICS): A statement of which capabilities and options have been implemented for a given Open Systems Interconnection (OSI) protocol.

3.2.15 span: A bidirectional channel connecting adjacent stations (two links).

3.2.16 station: A device attached to a network for the purpose of transmitting and receiving information on that network.

3.2.17 topology: The arrangement of links and stations forming a network, together with information on station attributes.

3.2.18 transmit (transmission): The action of a station placing a frame on the medium.

3.2.19 unicast: The act of sending a frame addressed to a single station.

3.3 State machines

3.3.1 State machine behavior

The operation of a protocol can be described by subdividing the protocol into a number of interrelated functions. The operation of the functions can be described by state machines. Each state machine represents the domain of a function and consists of a group of connected, mutually exclusive states. Only one state of a function is active at any given time. A transition from one state to another is assumed to take place in zero time (i.e., no time period is associated with the execution of a state), based on some condition of the inputs to the state machine.

The state machines contain the authoritative statement of the functions they depict. When apparent conflicts between descriptive text and state machines arise, the order of precedence shall be formal state tables first, followed by the descriptive text, over any explanatory figures. This does not override, however, any explicit description in the text that has no parallel in the state tables.

The models presented by state machines are intended as the primary specifications of the functions to be provided. It is important to distinguish, however, between a model and a real implementation. The models are optimized for simplicity and clarity of presentation, while any realistic implementation might place heavier emphasis on efficiency and suitability to a particular implementation technology. It is the functional behavior of any unit that has to match the standard, not its internal structure. The internal details of the model are useful only to the extent that they specify the external behavior clearly and precisely.

3.3.2 State table notation

NOTE—The following state machine notation was used within 802.17, due to the exactness of C-code conditions and the simplicity of updating table entries (as opposed to 2-dimensional graphics).
Early state table descriptions can be converted (if necessary) into other formats before publication.

Each row of the table is preferably provided with a brief description of the condition and/or action for that row. The descriptions are placed after the table itself, and linked back to the rows of the table using numeric tags.

State machines may be represented in tabular form. The table is organized into two columns: a left hand side representing all of the possible states of the state machine and all of the possible conditions that cause transitions out of each state, and the right hand side giving all of the permissible next states of the state machine as well as all of the actions to be performed in the various states, as illustrated in Table 3.1. The syntax of the expressions follows standard C notation (see 3.12). No time period is associated with the transition from one state to the next.

Table 3.1—State table notation example

Current		Row	Next	
state	**condition**		**action**	**state**
START	sizeOfMacControl > spaceInQueue	1	—	START
	passM == 0	2		
	—	3	TransmitFromControlQueue();	FINAL
FINAL	SelectedTransferCompletes()	4	—	START
	—	5	—	FINAL

Row 3.1-1: Do nothing if the size of the queued MAC control frame is larger than the PTQ space.
Row 3.1-2: Do nothing in the absence of MAC control transmission credits.
Row 3.1-3: Otherwise, transmit a MAC control frame.

Row 3.1-4: When the transmission completes, start over from the initial state (i.e., START).
Row 3.1-5: Until the transmission completes, remain in this state.

Each combination of current state, next state, and transition condition linking the two is assigned to a different row of the table. Each row of the table, read left to right, provides: the name of the current state; a condition causing a transition out of the current state; an action to perform (if the condition is satisfied); and, finally, the next state to which the state machine transitions, but only if the condition is satisfied. The symbol "—" signifies the default condition (i.e., operative when no other condition is active) when placed in the condition column, and signifies that no action is to be performed when placed in the action column. Conditions are evaluated in order, top to bottom, and the first condition that evaluates to a result of TRUE is used to determine the transition to the next state. If no condition evaluates to a result of TRUE, then the state machine remains in the current state. The starting or initialization state of a state machine is always labeled "START" in the table (though it need not be the first state in the table). Every state table has such a labeled state.

Each row of the table is preferably provided with a brief description of the condition and/or action for that row. The descriptions are placed after the table itself, and linked back to the rows of the table using numeric tags.

3.4 Arithmetic and logical operators

In addition to commonly accepted notation for mathematical operators, Table 3.2 summarizes the symbols used to represent arithmetic and logical (boolean) operations. Note that the syntax of operators follows standard C notation (see 3.12).

Table 3.2—Special symbols and operators

Printed character	Meaning
&&	Boolean AND
||	Boolean OR
!	Boolean NOT (negation)
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to
=	Assignment operator
//	Comment delimiter

3.5 Numerical representation

NOTE—The following notation was taken from 802.17, where it was found to have benefits:
– The subscript notation is consistent with common mathematical/logic equations.
– The subscript notation can be used consistently for all possible radix values.

Decimal, hexadecimal, and binary numbers are used within this working paper. For clarity, decimal numbers are generally used to represent counts, hexadecimal numbers are used to represent addresses, and binary numbers are used to describe bit patterns within binary fields.

Decimal numbers are represented in their usual 0, 1, 2, … format. Hexadecimal numbers are represented by a string of one or more hexadecimal (0-9,A-F) digits followed by the subscript 16, except in C-code contexts, where they are written as `0x123EF2` etc. Binary numbers are represented by a string of one or more binary (0,1) digits, followed by the subscript 2. Thus the decimal number “26” may also be represented as “$1A_{16}$” or “11010_2”.

MAC addresses and OUI/EUI values are represented as strings of 8-bit hexadecimal numbers separated by hyphens and without a subscript, as for example “01-80-C2-00-00-15” or “AA-55-11”.

3.6 Field notations

3.6.1 Use of italics

All field names or variable names (such as *level* or *myMacAddress*), and sub-fields within variables (such as *thisState.level*) are italicized within text, figures and tables, to avoid confusion between such names and similarly spelled words without special meanings. A variable or field name that is used in a subclause heading or a figure or table caption is also italicized. Variable or field names are not italicized within C code, however, since their special meaning is implied by their context. Names used as nouns (e.g., subclassA0) are also not italicized.

3.6.2 Field conventions

This working paper describes fields within packets or included in state-machine state. To avoid confusion with English names, such fields have an italics font, as illustrated in Table 3.3.

Table 3.3—Names of fields and sub-fields

Name	Description
newCRC	Field within a register or frame
thisState.level	Sub-field within field *thisState*
thatState.rateC[*n*].*c*	Sub-field within array element *rateC[n]*

Run-together names (e.g., *thisState*) are used for fields because of their compactness when compared to equivalent underscore-separated names (e.g., *this_state*). The use of multiword names with spaces (e.g., “This State”) is avoided, to avoid confusion between commonly used capitalized key words and the capitalized word used at the start of each sentence.

A sub-field of a field is referenced by suffixing the field name with the sub-field name, separated by a period. For example, *thisState.level* refers to the sub-field *level* of the field *thisState*. This notation can be continued in order to represent sub-fields of sub-fields (e.g., *thisState.level.next* is interpreted to mean the sub-field *next* of the sub-field *level* of the field *thisState*).

Two special field names are defined for use throughout this working paper. The name *frame* is used to denote the data structure comprising the complete MAC sublayer PDU. Any valid element of the MAC sublayer PDU, can be referenced using the notation *frame.xx* (where *xx* denotes the specific element); thus, for instance, *frame.serviceDataUnit* is used to indicate the *serviceDataUnit* element of a frame.

Unless specifically specified otherwise, reserved fields are reserved for the purpose of allowing extended features to be defined in future revisions of this working paper. For devices conforming to this version of this working paper, nonzero reserved fields are not generated; values within reserved fields (whether zero or nonzero) are to be ignored.

3.6.3 Field value conventions

This working paper describes values of fields. For clarity, names can be associated with each of these defined values, as illustrated in Table 3.4. A symbolic name, consisting of upper case letters with underscore separators, allows other portions of this working paper to reference the value by its symbolic name, rather than a numerical value.

Table 3.4—*wrap* field values

Value	Name	Description
0	STANDARD	Standard processing selected
1	SPECIAL	Special processing selected
2,3	—	Reserved

Unless otherwise specified, reserved values allow extended features to be defined in future revisions of this working paper. Devices conforming to this version of this working paper do not generate nonzero reserved values, and process reserved fields as though their values were zero.

A field value of TRUE shall always be interpreted as being equivalent to a numeric value of 1 (one), unless otherwise indicated. A field value of FALSE shall always be interpreted as being equivalent to a numeric value of 0 (zero), unless otherwise indicated.

3.7 Bit numbering and ordering

Data transfer sequences normally involve one or more cycles, where the number of bytes transmitted in each cycle depends on the number of byte lanes within the interconnecting link. Data byte sequences are shown in figures using the conventions illustrated by Figure 3.1, which represents a link with four byte lanes. For multi-byte objects, the first (left-most) data byte is the most significant, and the last (right-most) data byte is the least significant.

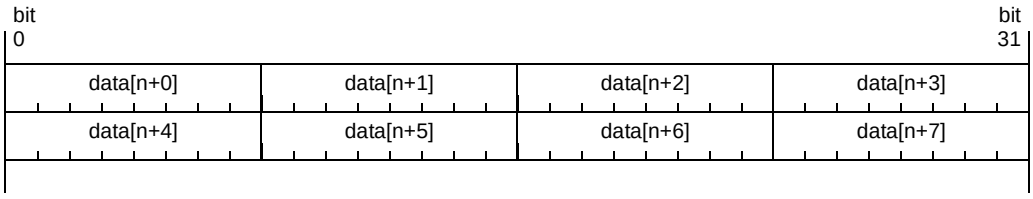

Figure 3.1—Bit numbering and ordering

Figures are drawn such that the counting order of data bytes is from left to right within each cycle, and from top to bottom between cycles. For consistency, bits and bytes are numbered in the same fashion.

NOTE—The transmission ordering of data bits and data bytes is not necessarily the same as their counting order; the translation between the counting order and the transmission order is specified by the appropriate reconciliation sublayer.

3.8 Byte sequential formats

Figure 3.2 provides an illustrative example of the conventions to be used for drawing frame formats and other byte sequential representations. These representations are drawn as fields (of arbitrary size) ordered along a vertical axis, with numbers along the left sides of the fields indicating the field sizes in bytes. Fields are drawn contiguously such that the transmission order across fields is from top to bottom. The example shows that *field1*, *field2*, and *field3* are 1-, 1- and 6-byte fields, respectively, transmitted in order starting with the *field1* field first. As illustrated on the right hand side of Figure 3.2, a multi-byte field represents a sequence of ordered bytes, where the first through last bytes correspond to the most significant through least significant portions of the multi-byte field, and the MSB of each byte is drawn to be on the left hand side.

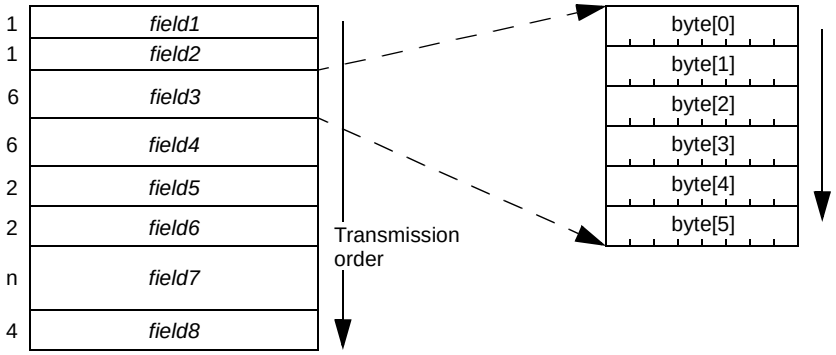

Figure 3.2—Byte sequential field format illustrations

NOTE—Only the left-hand diagram in Figure 3.2 is required for representation of byte-sequential formats. The right-hand diagram is provided in this description for explanatory purposes only, for illustrating how a multi-byte field within a byte sequential representation is expected to be ordered. The tag “Transmission order” and the associated arrows are not required to be replicated in the figures.

3.9 Ordering of multibyte fields

In many cases, bit fields within byte or multibyte objects are expanded in a horizontal fashion, as illustrated in the right side of Figure 3.3. The fields within these objects are illustrated as follows: left-to-right is the byte transmission order; the left-through-right bits are the most significant through least significant bits respectively.

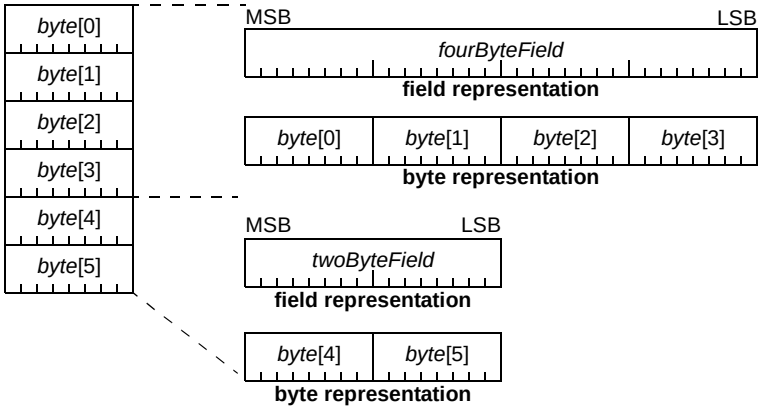

Figure 3.3—Multibyte field illustrations

The first *fourByteField* can be illustrated as a single entity or a 4-byte multibyte entity. Similarly, the second *twoByteField* can be illustrated as a single entity or a 2-byte multibyte entity.

To minimize potential for confusion, four equivalent methods for illustrating frame contents are illustrated in Figure 3.4. Binary, hex, and decimal values are always shown with a left-to-right significance order, regardless of their bit-transmission order.

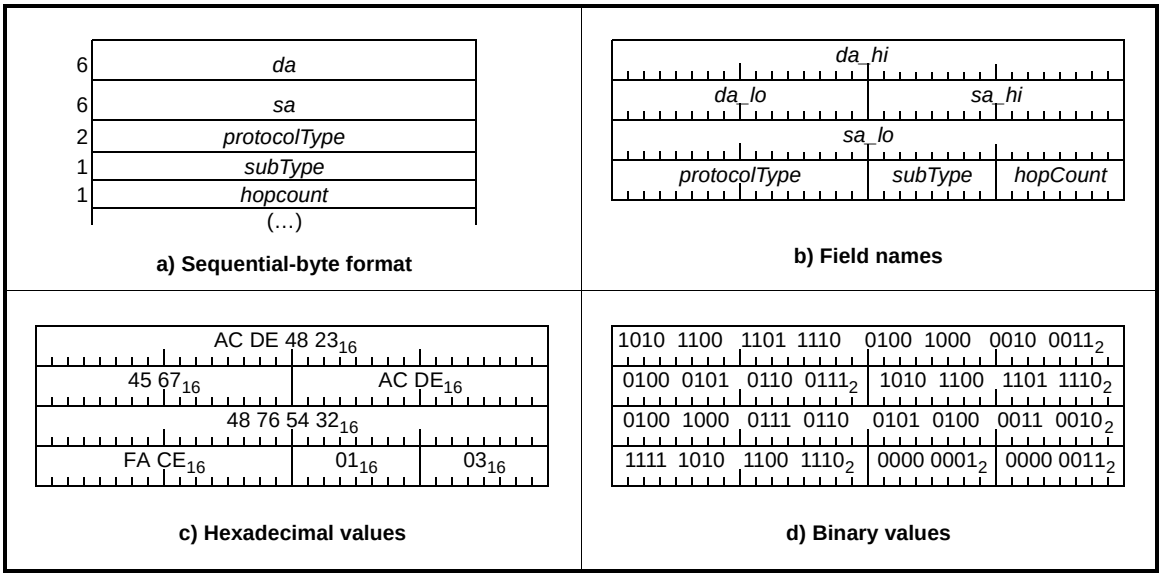

Figure 3.4—Illustration of fairness-frame structure

3.10 MAC address formats

The format of MAC address fields within frames is illustrated in Figure 3.5.

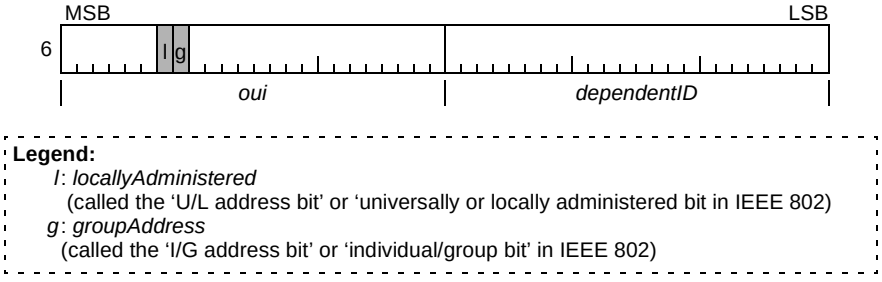

Figure 3.5—MAC address format

3.10.1 *oui*: A 24-bit organizationally unique identifier (OUI) field supplied by the IEEE/RAC for the purpose of identifying the organization supplying the (unique within the organization, for this specific context) 24-bit *dependentID*. (For clarity, the *locallyAdministered* and *groupAddress* bits are illustrated by the shaded bit locations.)

3.10.2 *dependentID*: An 24-bit field supplied by the *oui*-specified organization. The concatenation of the *oui* and *dependentID* provide a unique (within this context) identifier.

To reduce the likelihood of error, the mapping of OUI values to the *oui/dependentID* fields are illustrated in Figure 3.6. For the purposes of illustration, specific OUI and *dependentID* example values have been assumed. The two shaded bits correspond to the *locallyAdministered* and *groupAddress* bit positions illustrated in Figure 3.5.

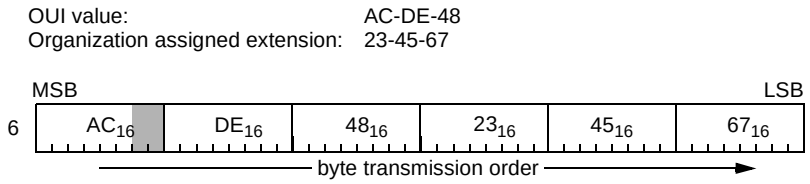

Figure 3.6—48-bit MAC address format

3.11 Informative notes

Informative notes are used in this working paper to provide guidance to implementers and also to supply useful background material. Such notes never contain normative information, and implementers are not required to adhere to any of their provisions. An example of such a note follows.

NOTE—This is an example of an informative note.

3.12 Conventions for C code used in state machines

Many of the state machines contained in this working paper utilize C code functions, operators, expressions and structures for the description of their functionality. Conventions for such C code can be found in Annex G.

4. Abbreviations and acronyms

This working paper contains the following abbreviations and acronyms:

AP	access point
AV	audio/video
AVB	audio/video bridging
AVB network	audio/video bridged network
BER	bit error ratio
BMC	best master clock
BMCA	best master clock algorithm
CRC	cyclic redundancy check
EFDX	Ethernet full duplex
EPON	Ethernet passive optical network
FIFO	first in first out
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
ISO	International Organization for Standardization
ITU	International Telecommunication Union
LAN	local area network
LSB	least significant bit
MAC	medium access control
MAN	metropolitan area network
MSB	most significant bit
OSI	open systems interconnect
PDU	protocol data unit
PHY	physical layer
PLL	phase-locked loop
PPM	parts per million
PTP	Precision Time Protocol
R11V	radio 802.11v
RFC	request for comment
RPR	resilient packet ring
TS	time-synchronization
VOIP	voice over internet protocol

5. Architecture overview

5.1 Application scenarios

5.1.1 Garage jam session

As an illustrative example, consider AVB usage for a garage jam session, as illustrated in Figure 5.1. The audio inputs (microphone and guitar) are converted, passed through a guitar effects processor, two bridges, mixed within an audio console, return through two bridges, and return to the ear through headphones.

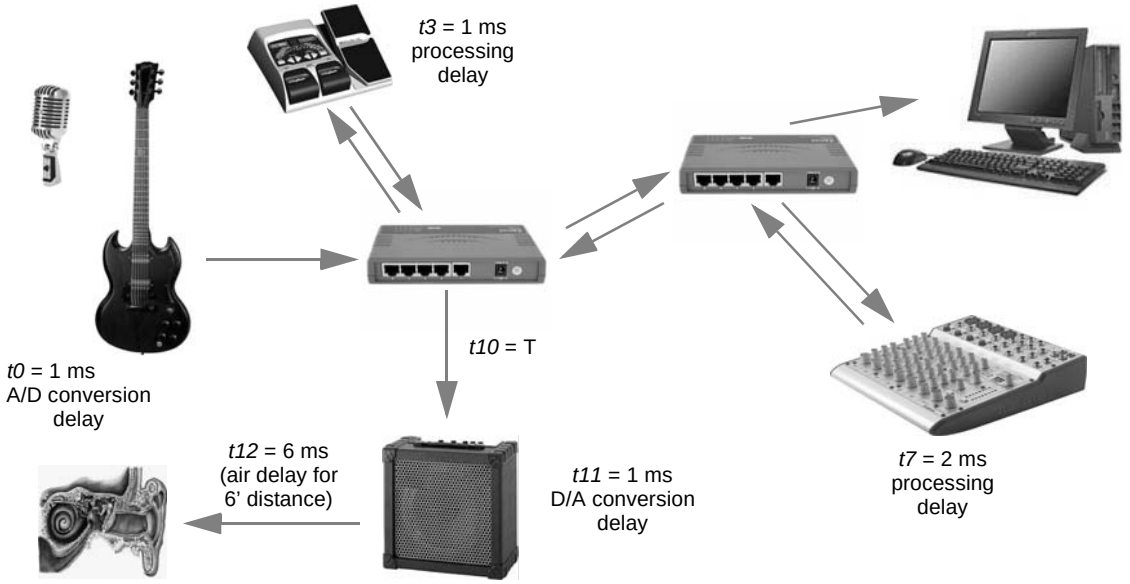

Figure 5.1—Garage jam session

Using Ethernet within such systems has multiple challenges: low-latency and tight time-synchronization. Tight time synchronization is necessary to avoid cycle slips when passing through multiple processing components and (ultimately) to avoid under-run/over-run at the final D/A converter's FIFO. The challenge of low-latency transfers is being addressed in other forums and is outside the scope of this draft.

5.1.2 Looping topologies

Bridged Ethernet networks currently have no loops, but bridging extensions are contemplating looping topologies. The time-synchronization protocols assume that looping physical topologies could be present, as illustrated by the dotted-line connection in Figure 5.2, but logical loops would be eliminated by invoking the existing IEEE 802.1D rapid-spanning tree protocols.

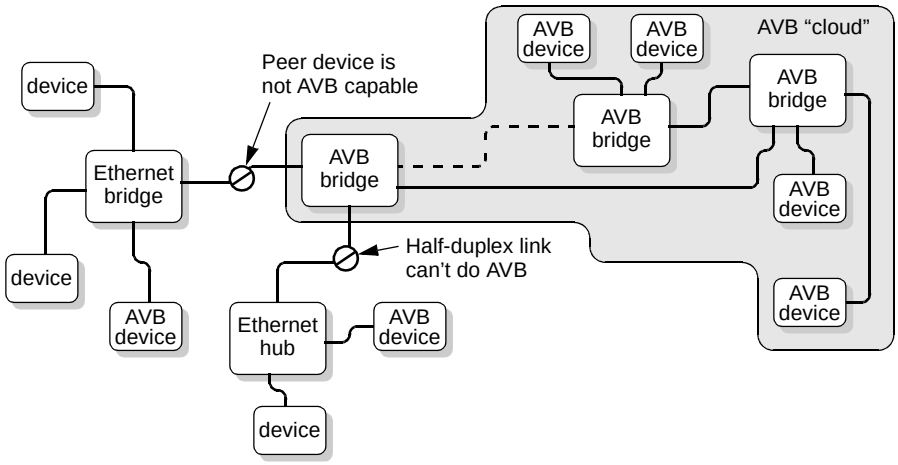

Figure 5.2—Possible looping topology

Grandmaster-selection, time-synchronization, and link-delay calibration information across the (illustrated as solid lines) active spans. Only link-delay calibration information is assumed to flow across the (illustrated as dotted lines) inactive spans; knowledge of these delays reduces clock-discontinuities in the event of spanning-tree topology changes.

Separation of AVB devices is driven by the requirements of AVB bridges to support subscription (bandwidth allocation) and pacing of time-sensitive transmissions, as well as time-of-day clock-synchronization.

5.2 Design methodology

5.2.1 Assumptions

This working paper specifies a protocol to synchronize independent timers running on separate stations of a distributed networked system, based on concepts specified within IEEE Std 1588-2002. Although a high degree of accuracy and precision is specified, the technology is applicable to low-cost consumer devices. The protocols are based on the following design assumptions:

a) Each end station and intermediate bridges provide independent clocks.

b) All clocks are accurate, typically to within ±100PPM.

c) Details of the best time-synchronization protocols are physical-layer dependent.

5.2.2 Objectives

With these assumptions in mind, the time synchronization objectives include the following:

a) Precise. Multiple timers can be synchronized to within 10's of nanoseconds.

b) Inexpensive. For consumer AVB devices, the costs of synchronized timers are minimal. (GPS, atomic clocks, or 1 PPM clock accuracies would be inconsistent with this criteria.)

c) Scalable. The protocol is independent of the networking technology:

1) Transmission intervals can be independently specified and optimized, on a per-link basis.
2) Long distance links (up to 2 kM) are allowed.

d) Plug-and-play. The system topology is self-configuring; no system administrator is required.

5.2.3 Strategies

Strategies used to meet these objectives include the following:

a) Precision is achieved by calibrating and adjusting *grandTime* clocks.
   1) Offsets. Offset value adjustments eliminate immediate clock-value errors.
   2) Rates. Rate value adjustments reduce long-term clock-drift errors.

b) Simplicity is achieved by the following:
   1) Concurrence. Most configuration and adjustment operations are performed concurrently.
   2) Firmware friendly. Clock offset and rate adjustments can be performed by low-rate firmware; analog PLLs are unnecessary within bridges (although necessary within some applications).
   3) Frequent. Frequent ~100 Hz interchanges eliminates needs for expensive/precise clocks.

5.3 Network topologies

Clock synchronization involves streaming of timing information from a grandmaster clock to one or more slave clocks. The synchronization protocols assume a spanning-tree topology, but function correctly on cyclical physical topologies, if spanning-tree protocols (802.1D) have activated only a non-cyclical subset of the physical topology.

The grandmaster station (GM) provides a time reference to locally attached clock-slave station (S), as illustrated in Figure 5.3a. Bridges synchronize their clock-master (M) ports to their clock-slave (S) port, via internal communications. These clock-master (M) ports typically connect to another bridge's clock-slave (S) port, or to an attached end-station clock-slave station (ES).

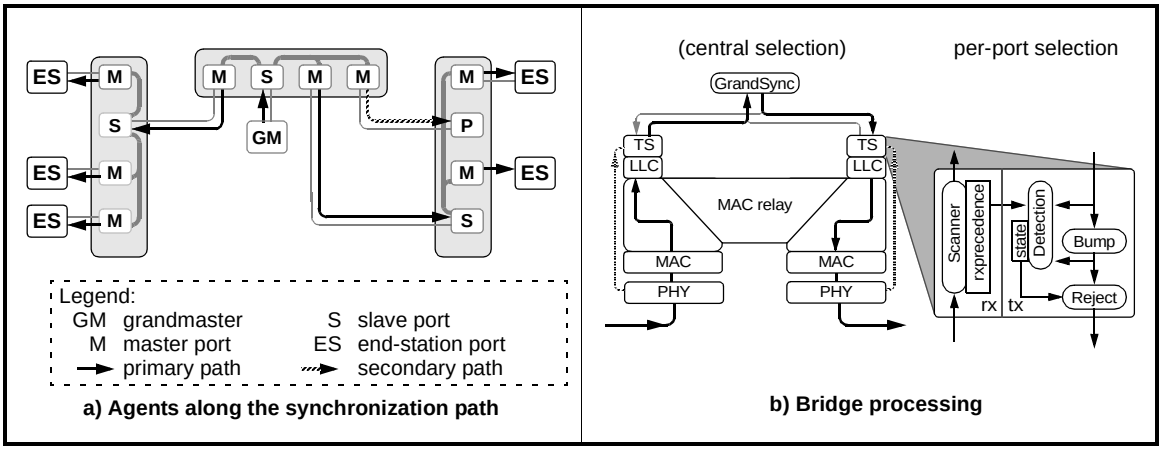

Figure 5.3—Synchronized-system topologies

In concept, network-wide clock-synchronization protocol starts with the selection of the reference-clock station, called a grandmaster station (oftentimes abbreviated as grandmaster). Every AVB-capable station is grandmaster capable, but only one is selected to become the grandmaster station within each network grandmaster selection involves the transmission of ClockSync messages between grandmaster capable stations.

In the presence of redundant loops, a bridge port can also be placed in a passive (P) state, wherein it monitors ClockSync messages but does not synchronize its clock to its associated clock-master port.

After grandmaster selection, time-reference information flows from this selected grandmaster station to attached clock-slave stations. Thus, time-synchronization involve two subservices, as follows:

a) Selection. Topologies are pruned into spanning tree with the grandmaster at the root (see 5.5).
b) Distribution. Synchronized time is distributed through the spanning tree from the root (see 5.6).

5.4 Information parameters

Clock-synchronization information includes clock-selection parameters (for selecting the grandmaster) and clock-synchronization parameters (for synchronizing between clock-master and clock-slave ports), as illustrated in Figure 5.4. A portion of the clock-synchronization parameters are media-independent and pass across the bridge-internal service interface; the remainder are link-media dependent and used for calibration of clock-master to clock-slave port delays.

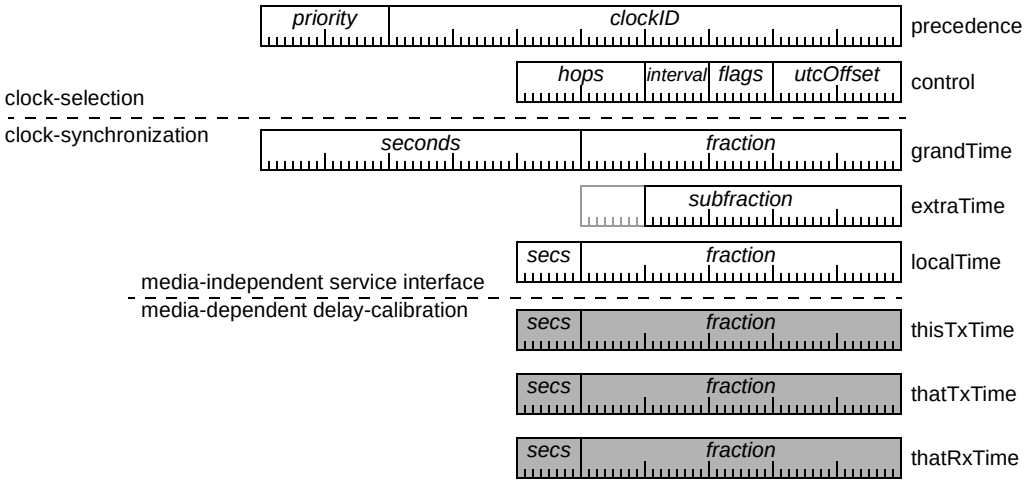

Figure 5.4—GrandSync service-interface components

NOTE—Depending on the media specification, the clock-selection and clock-synchronization parameters could be a single ClockSync packet (as currently assumed) or transmitted in distinct frames. The optimal partitioning into frames types (hopefully, no more than two) is an outstanding topic of discussion.

The concatenation of ***priority*** and ***clockID*** values forms a precedence value for the selection of the preferred grandmaster candidate. The concatenation of ***hops***, ***interval***, ***flags***, and ***utcOffset*** values provide additional information to the clock-slave devices.

The {*grandTime*,*extraTime*,*localTime*} triad percolates from the grand-master to the clock-slave stations, with adjustments along the way, to provide clock-slave stations with estimates of the grandmaster's clock time. The ***grandTime*** represents a station's estimate of the grandmaster's clock sampled at that stations's ***localTime*** instant. The ***extraTime*** value represents a cumulative deviation error that (for the purposes of accuracy and responsiveness) is maintained separately from the *grandTime* value.

The remaining media-dependent parameters allow a bridge's receive-port to calibrate and compensate for delays introduced by the receive link of the span connecting its neighbor. For full-duplex Ethernet, these include ***thatTxTime*** and ***thatRxTime*** (snapshots taken on opposing-link frames), and ***thisTxTime*** (a snapshot taken on the neighbor's transmission). Different values are applicable to alternative media types.

5.5 Grandmaster selection

Intermediate bridge entities are responsible for providing standard/centralized precedence-selection and media-specific/per-port rejection entities, as illustrated in Figure 5.3b. The central GrandSync precedence-selection entity is responsible for selecting and echoing only the best ClockSync messages. The per-port rejection entity is responsible for discarding (unnecessary) reverse-flow ClockSync messages.

As part of the grandmaster selection process, ClockSync messages are generated by clock-master capable stations; the ClockSync messages transport clock-precedence values (see Figure 5.5) to other stations. Only ClockSync messages with the best observed clock-precedence value are forwarded to neighbor stations, allowing the overall best-precedence value to be ultimately selected and known by all. The station with that best precedence value is called the grandmaster.

The grandmaster precedence is set to be the concatenation of *priority* and *clockID* fields. The value of each station's ***priority*** field can be set by the application, for the purposes of biasing the grandmaster precedence in an application-specific manner. The ***clockID*** field is a globally unique number assigned to each station.

The ***hops*** field is incremented by each bridge and thus represents the distance from the grandmaster. The following interval and flags fields consist of multiple components, as illustrated in Figure 5.5.

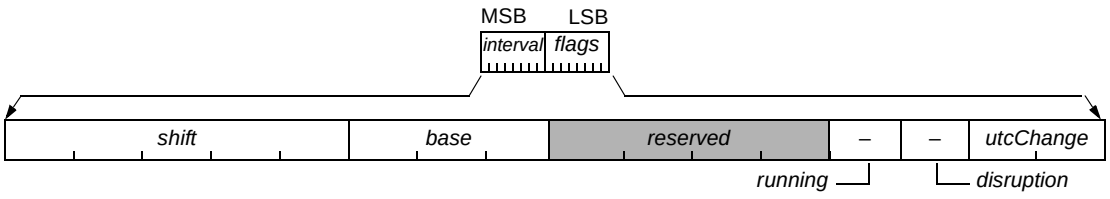

Figure 5.5—*interval* and *flags* parameters

The ***interval*** field specifies the source-station's nominal interval between ClockSync frames, and is used for timeout purposes. For compactness, its value is encoded as simple floating point value, as summarized below (see 6.2.1.5).

```
value = base << shift;
```

The ***flags*** field (in conjunction with the *utcOffset* field) supports the management of leap-second updates and timeouts. The two-bit value of *utcChange* specifies whether the *utcOffset* value should be unchanged, incremented, or decremented at the stroke of midnight (see 6.2.1.6).

5.5.1 Central ClockSync-message processing

The central/standard GrandSync entity (see Figure 5.3b) is responsible for processing ClockSync PDUs provided by attached ports, as summarized in Equation 5.1. If the ClockSync message precedence equals or exceeds the previously saved *sync.precedence* value: *sync.precedence* is updated with the ClockSync-message precedence, the ClockSync-message timeout is cleared, and the ClockSync message is echoed to all bridge ports. A timeout clears the *sync.precedence* history in the absence of expected (normally periodic) ClockSync messages.

(5.1)

```
// Performed by the GrandSync entity
// Compare(x,y) operates on multiple-precision arguments x and y
// The returned integer result is:
     1 if x-y > 0
     0 if x-y == 0
    -1 if x-y < 0
while (FOREVER) {                                          // Check constantly
  clockSync = Dequeue(GS_RX);                              // Get ClockSync message
  if (clockSync != NULL) {                                 // if one is available
    gsprecedence = Gsprecedence(clockSync);                // Form precedence value
    if (Compare(gsprecedence,sync.precedence) <= 0) {      // The best precedence
      sync.precedence = gsprecedence;                      // is observed and saved
      Enqueue(GS_TX, clockSync);                           // Echoed the PDU to others
      sync.lastTime = currentTime;                         // Restart the timeout
      sync.timeout = GsTimeout(clockSync.interval);        // Form precedence value
    }
  }
  if (currentTime >= sync.lastTime + sync.timeout)         // Inactivity timeout;
    grandprecedence = ONES;                                // clear saved precedence
}
```

5.5.2 Per-port received ClockSync-message processing

The per-port/media-dependent TS.rx entity (see Figure 5.3b) is responsible for processing rxClockSync PDUs provided by attached ports, as summarized in Equation 5.2. If the rxClockSync-PDU precedence equals or exceeds the previously saved *port.precedence* value: *port.precedence* is updated with the rxClockSync-PDU precedence and the ClockSync-message timeout is cleared. Regardless, the (media dependent) rxClockSync-PDU is converted to a generic clockSync-PDU and sent to the central GrandSync entity. A timeout clears the *port.precedence* history in the absence of expected (normally periodic) ClockSync messages.

(5.2)

```
// Performed by the TS.receive entity
while (FOREVER) {                                          // Check constantly
  clockSync = Dequeue(ES_RX);                              // Get a ClockSync
  if (clockSync != NULL) {                                 // if one is available
    rxprecedence = Rxprecedence(clockSync);                // Form precedence value
    if (Compare(rxprecedence, port.precedence) < 0) {      // The best precedence
      port.precedence = rxprecedence;                      // is observed and saved
      port.lastTime = currentTime;                         // Restart the timeout
    }
    Enqueue(GS_RX, EsToGs(rxClockSync));                   // Send PDU to GrandSync
  }
  if (currentTime >= port.lastTime + rxTimeout)            // Inactivity timeout;
    port.precedence = ONES;                                // clear saved precedence
}
```

5.5.3 Per-port transmit ClockSync-message processing

The per-port TS.rx entity (see Figure 5.3b) is responsible for setting the port state and selectively forwarding clockSync PDUs provided by the GrandSync entity, as summarized in Equation 5.3. If *port.precedence* compares favorably to the clockSync-PDU (indicating this is the best-precedence port), the *port.state* is set to CLOCK_SLAVE and ClockSync messages are discard. Otherwise, port.state (and its associated behaviors) are dependent on comparisons between port and to-be-transmitted PDU precedences.

(5.3)

```
// Performed by the TS.transmit entity
while (FOREVER) {                                          // Check constantly
  clockSync = Dequeue(GS_TX);                              // Get received ClockSync
  if (clockSync != NULL) {                                 // if one is available
    if (clockSync.hops == HOP_LIMIT) return;               // Ignore aged frames
    g0Precedence = Gsprecedence(clockSync);                // Form precedence value
    gsClockSync.hops += 1;                                 // Increment hop count
    g1Precedence = Gsprecedence(clockSync);                // Form precedence value
    if (Compare(port.precedence, g0Precedence) <= 0)       // Rx precedence is best;
      port.state = CLOCK_SLAVE, return;                    // Rx ClockSync and time
    switch (Compare(g1Precedence, port.precedence)) {      // Tx precedence compare
    case -1:
      port.state = CLOCK_PASSIVE, return;                  // Rx ClockSync, ignore time
    case 1:
      port.state = CLOCK_MASTER, break;                    // Tx ClockSync and time
    case 0:
      port.state = CLOCK_IDLED, break;                     // Tx ClockSync, block time
    }
    Enqueue(ES_TX, GsToEs(clockSync));                   // Transmit the ClockSync
  }
}
```

5.6 Synchronized time distribution

5.6.1 Clock-time synchronization flows

As background for understanding, consider the flow of clock-time synchronization information within a simple/stable three-bridge topology, as illustrated in Figure 5.6. The clock-time information flows from the selected application-level ClockSource entity (located on the grandmaster station), through intermediate bridges (when present), and terminates at application-level ClockTarget entities.

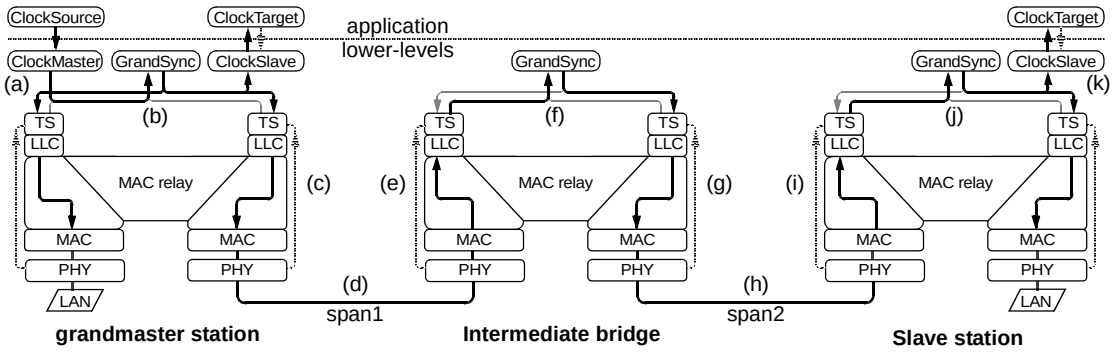

Figure 5.6—Clock-time synchronization flows

The clock-master station (containing the ClockSource entity) and the clock-slave station (containing the ClockTarget entity) are illustrated as multipurpose bridges; either could also be end stations (not illustrated).

Time synchronization yields distributed but closely-matched *grandTime* values within stations and bridges. No attempt is made to eliminate intermediate jitter with bridge-resident jitter-reducing phase-lock loops (PLLs) but application-level phase locked loops are expected to filter high-frequency jitter from the supplied *grandTime* values.

Maintaining an accurate time reference relies on the presence of accurate time-stamp hardware capabilities in or near the media-dependent PHY. A bypass path (illustrated as hashed PHY-to-TS lines within Figure 5.6) allows the time-stamp information to be affiliated with the arriving ClockSync-frame information, before the PDU is processed by the time-synchronization (TS) entity above the MAC. A similar bypass path is also required at the transmitter, so that an accurate time-stamp of a transmitted frame can be placed into the next-transmitted ClockSync frame.

This grandmaster station comprises client-level ClockTarget as well as ClockSource entities. The ClockTarget entity allows the application-level clock to be synchronized to the network clock, whenever another station becomes the grandmaster station. (The ClockSource entity on the grandmaster station provides the network-synchronized clock-time reference.)

5.6.2 Steps in the *grandTime* flow

Processing steps along the path of the time-synchronization flow of Figure 5.6 include the following:

a) The ClockSource's time-reference parameters are converted into a standard format, supplemented with additional parameters (such as the station-local arrival time), packaged into a PDU, and sent to the GrandSync entity.

b) The GrandSync entity echoes the time-synchronization information from (what it determines to be) the preferred port. Information from lower-precedence ports is continuously monitored to detect precedence changes (typically due to attach or detach of clock-master capable stations).
Clock-time information in PDUs from lower-precedence ports is ignored.

c) Clock-time parameters within the echoed PDUs are saved in transmit-port time-value array. When the next ClockSync frame is transmitted, the transmission time is computed based on these array values. The computed transmission time is queued for inclusion in the following ClockSync transmission.

d) The ClockSync frame travels over span1 to the intermediate bridge, incurring a media-dependent transmission *delay* and arriving at (station-local) time *rxTime*.

e) The (station-local) frame-arrival snapshot value, *rxTime*, is compensated by subtracting the estimated transmission delay to create an *rcTime* value, where:
rcTime=rxTime-delay0, where *delay0* is an estimate of span *delay.*
The *grandTime* value is extracted from the arriving frame; the {*grandTime,rcTime*} time-pair values are encapsulated within a PDU; that PDU is sent to the GrandSync entity.

f) The GrandSync entity echoes the ClockSync PDU provided by the preferred port, as in step (c).

g) Time parameters within the echoed PDUs are saved and eventually transmitted, as in step (d).

h) The ClockSync frame travels over span2 to the final bridge, incurring a media-dependent transmission *delay* and arriving at (station-local) time *rxTime*.

i) The (station-local) frame-arrival snapshot value, *rxTime*, is compensated by subtracting the estimated transmission delay to created an *rcTime* value, as in step (d).
The {*grandTime,extraTime,rcTime*} affiliations are placed in a PDU and sent to the GrandSync.

j) The GrandSync entity echoes the ClockSync PDU from the preferred port, as in step (c).

k) The GrandSync's time-reference parameters are converted from the standard PDU format, unnecessary parameters are discarded, and *grandTime* is communicated to the GrandSync entity.

5.6.3 Intermediate bridge entities

Entities within the intermediate bridge (see Figure 5.6) are responsible for performing three distinct (and largely decoupled) functions, as illustrated in Figure 5.7. A port in the CLOCK_SLAVE state is responsible for compensating for link-transmission delays between this station and its neighbor, as described in 5.6.4. The GrandSync entity is responsible for selecting the ClockSync PDUs from the grandmaster station; only thus selected PDUs are forwarded to transmitter ports.

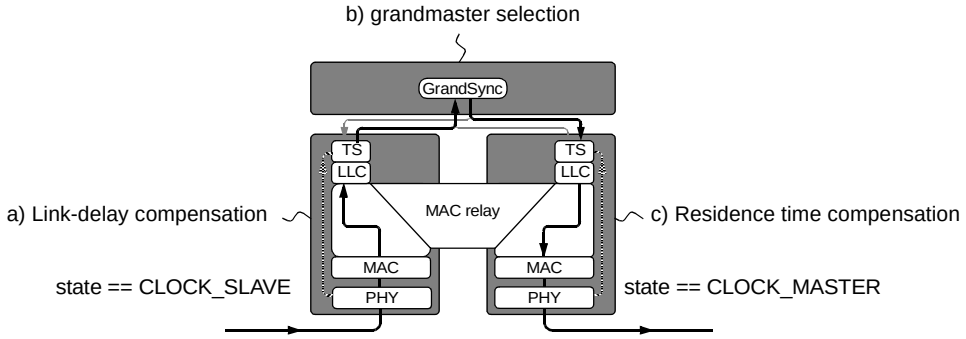

Figure 5.7—Intermediate-bridge responsibilities

Ports in the CLOCK_MASTER state are responsible for revising the GrandSync-supplied ClockSync PDUs to supply the appropriate media-dependent service-interface parameters and/or frames. Since the transmission times and rates may differ from those on the clock-slave port, the CLOCK_MASTER-state port is responsible for interpolating/extrapolating between previously received time samples to generate parameters corresponding to the recently observed transmit-snapshot, as described in 5.6.4

The concept of a *residence time* is based on a simplistic assumption that ClockSync frames are received, pass through the station, and are then transmitted on the other port. However, the phase and/or frequency of receive and transmit ClockSync frames may differ. A more general model of value resampling (see 5.6.5) is therefore assumed within this document.

5.6.4 CLOCK_SLAVE link-delay compensation

When a ClockSync frame is received at a CLOCK_SLAVE-state port, a snapshot of the local *localTime* is saved. This snapshot time is compensated by the measured link delay. Although these time-delay adjustments are media-dependent, the concepts described within this subclause are applicable to most transmission media.

Processing of the *rxTime* snapshot to account for link-transmission delays involves the subtraction of a precomputed link *delay* value, as illustrated in Figure 5.8a. In parallel with this subtraction, the received *grandTime* value is extracted from the received ClockSync frame. The *grandTime* and *rcTime* (the delay-compensated *txTime*) values provide an accurate time-affiliation pair that is packaged into the PDU and sent to the GrandSync entity.

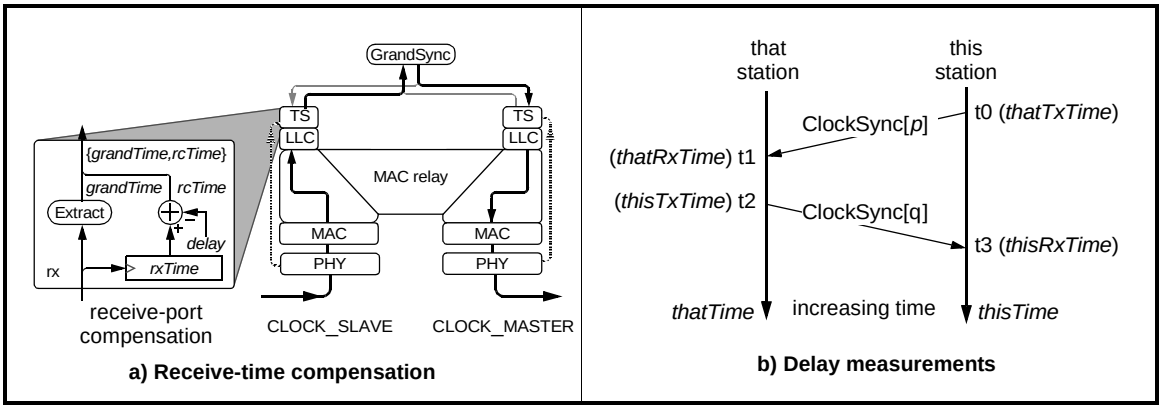

Figure 5.8—CLOCK_SLAVE link-delay compensation

Link delays can be accurately precomputed if the delays are constant, accurate time-snapshot values are present, and the clocks are stable. The computations are based on time-stamp measurements performed on distinct ClockSync[*p*] and following ClockSync[*q*] frame transmissions, as illustrated in Figure 5.8b.

These computations are based on the free-running station-local timers, illustrated as *thisTime* and *thatTime*. For stability and accuracy, computations use time differences measured with respect to the local timers; constant offset or frequency errors between local and neighbor timers thus have no effect on the accuracy of the computations.

To improve accuracy, a ratio of local-to-neighbor clock-intervals is precomputed over N (perhaps 16) frame transmissions, as specified in Equation 5.4. For hardware accuracy and simplicity, the *t2* transmission-time snapshot are held and transmitted in the following frame. Thus, the *t2* snapshot for ClockSync[n] is transmitted within the following ClockSync[n+1] frame.

```
#define N 16                                          // A typical value
ratio = (t3[n+N]-t3[n]) / (t2[n+N]-t2[n]);            // Neighbor's rate ratio
```
(5.4)

For symmetry and accounting simplicity, the *t0* transmission-time snapshot is also held and transmitted in the following frame. Affiliated {*t0*,*t1*} pairs are then returned in following ClockSync[*q*] frame transmissions. The process is symmetric and thus allows either station to (otherwise independently) compute the average link transmission delay, as specified in Equation 5.5.

```
delay = ((t3[q]-t0[p]) - rateRatio*(t2[q]-t1[p]))/2;  // Average link delay
```
(5.5)

5.6.5 CLOCK_MASTER time-sample interpolation

A port in the CLOCK_MASTER state receives the {*grandTime*,*extraTime*,*rcTime*} triads that are echoed by the GrandSync entity. Since these (oftentimes) cannot be transmitted immediately, the received time-triads are saved in a *times*[ ] array, for access after the next ClockSync frame transmission, as illustrated in Figure 5.9a.

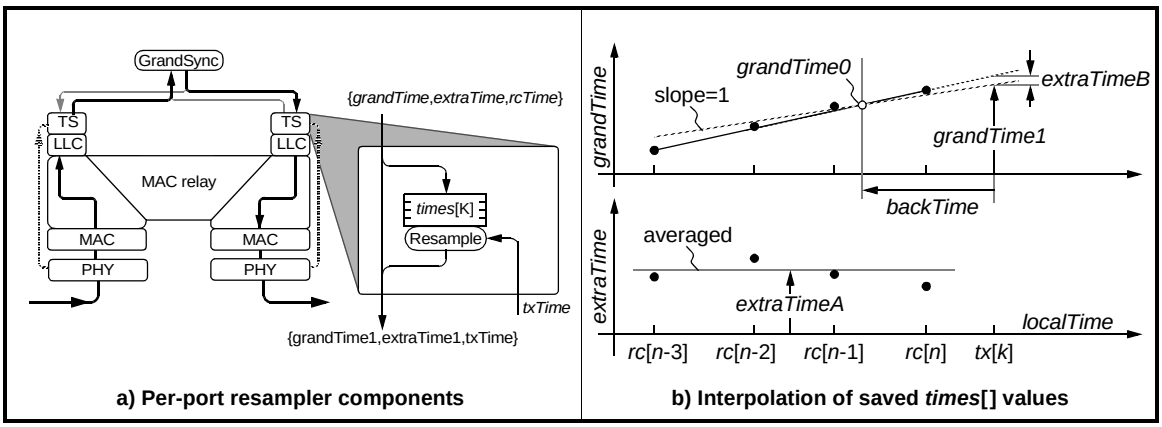

Figure 5.9—CLOCK_MASTER time-sample interpolation

NOTE—The distribution of distinct *times*[ ] storage entities is an architectural model; implementations could emulate the specified behavior by providing access to shared storage and intermediate results.

During the next ClockSync frame transmission, a *txTime* snapshot is produced. The Resample logic operates on the *times*[ ] array values to compute the {*grandTime1*,*extraTime1*,*txTime*} time-triad that is placed within the following ClockSync frame transmission, based on interpolation and averaging strategies illustrated in Figure 5.9b. Within Figure 5.9b, *rc* and *tx* are abbreviations for *rcTime* and *txTime* values, respectively.

To avoid traditional whiplash and gain-peaking problems associated with cascaded PLLs, an interpolation (as opposed to extrapolation) strategy yields distinct *grandTime1* and *extraTime1* values. These two times are forwarded independently but are recombined at the ClockTarget entity, so that a single time can be passed to the higher-level application.

In concept, the computation of *grandTime1* involves computing *tbTime= txTime-backTime*, using *tbTime* to generate grandTime0 by interpolating between previously saved *times*[ ] values, and then generating *grandTime1* by extrapolating *grandTime0* forward to time *txTime* (assuming a constant *grandTime* slope of one). The value of *extraTimeB* is set to the difference between an extrapolated value (based on saved *times*[ ] values) and the computed *grandTime1* value.

Computation of the *extraTime1* involves computing *extraTimeA*, the average of previously saved *extraTime* values within the *times*[ ] array. The sum of *extraTimeA* (upstream station's contributions) and *extraTimeB* (this station's contribution) yields the *extraTime1* value, as summarized in Equation 5.6.

```
#define N 4                                                    // Typical value (5.6)
rateRatio=                                                     // Relative rates
  (grandTime[n] - grandTime[n-N]) / (rcTime[n] - rcTime[n-N]); // GM's rate ratio
for (extraTimeA= i= 0; i < N; i += 1)                          // Use samples to
  extraTimeA += extraTime[n-i]/N;                              // find an average
extraTimeB = backTime * (rateRatio - 1.0);                     // Contributed value
grandTime1 = rc[n] + (txTime - rc[n]) * rateRatio - extraTimeB; // Tx grandTime value
extraTime1 = extraTimeA/N + extraTimeB;                        // Tx extraTime value
```

5.7 Comparison to IEEE Std 1588

5.7.1 Distinction summary

Advantageous properties of this protocol that distinguish it from other protocols (including portions of IEEE Std 1588) include the following:

a) Synchronization between grandmaster and local clocks occurs at each station:
   1) All bridges have a lightly filtered synchronized image of the grandmaster time.
   2) End-point stations have a heavily filtered synchronized image of the grandmaster time.

b) Time is uniformly represented as scaled integers: seconds and fractions-of-a-second.

c) Locally media-dependent synchronized networks don't require extra time-snapshot hardware.

d) Error magnitudes are sublinear with hop distances; PLL-whiplash and $O(n^2)$ errors are avoided.

e) Multicast (one-to-many) services are not required; only adjacent-neighbor addressing is required.

f) A relatively frequent 100 Hz (as compared to 1 Hz) update frequency is assumed:
   1) This rate can be readily implemented (in today's technology) for minimal cost.
   2) The more-frequent rate improves accuracy and reduces transient-recovery delays.
   3) The more-frequent rate reduces transient-recovery delays.

g) Only one frame type simplifies the protocols and reduces transient-recovery times. Specifically:
   1) Cable delay computations, based on local clocks, are unaffected by *grandTime* transients.
   2) Rogue frames are quickly scrubbed (2.6 seconds maximum, for 256 stations).
   3) Drift-induced errors are greatly reduced.

5.7.2 Common snapshot hardware

For precise clock tracking, both IEEE 802.1as and IEEE 1588 working groups have assumed the presence of accurate timestamp hardware at their receive and transmit PHY ports. Through cooperative efforts, IEEE 802.1as and IEEE 1588 working groups have worked to ensure the same hardware can be used to support both designs, as illustrated in Figure 5.10a.

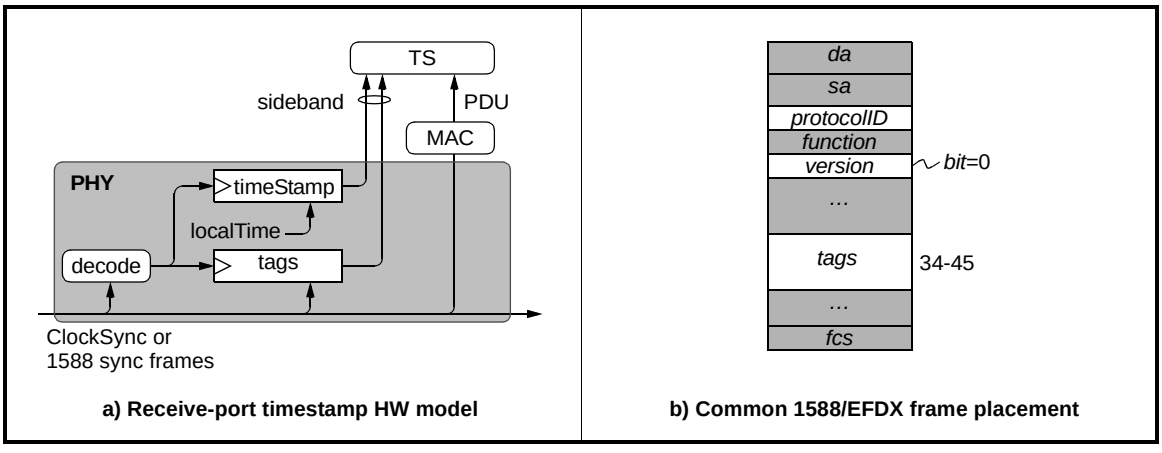

Figure 5.10—Common snapshot hardware

In IEEE p1588, the timestamp hardware decodes the *protocolID* and a selected bit in the following version field, as illustrated in Figure 5.10b (requirements on this bit are specified in 6.2.1.3). Because only one time-stamped frame type (and its transmission rate) is specified, the use of special tags with a matching frame-sequence placement is unnecessary.

5.7.3 1588 grandmaster precedence

Within 1588, the grandmaster precedence is based on the concatenation of multiple fields, including clock-master identifiers, as illustrated in Figure 5.11. The gray-shaded values are not supported in this document.

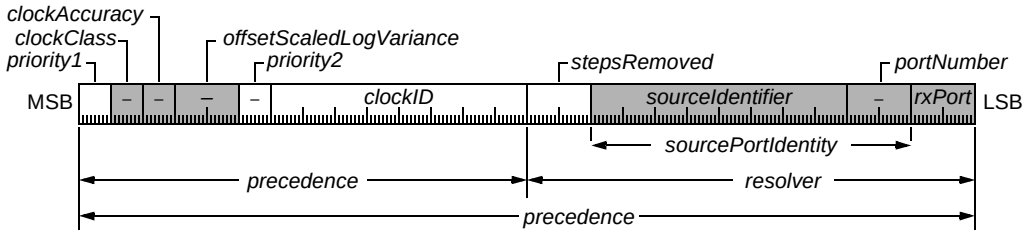

Figure 5.11—1588 grandmaster precedence

The ***clockClass***, ***clockAccuracy***, and ***offsetScaledLogVariance*** values are sparsely-encoded and application-dependent and thus not supported in this document. Similar functionality can be achieved by higher-level conventions that specify the partitioning/assignment of distinct 16-bit *priority* values.

The ***sourceIdentifier***, ***portNumber***, and ***rxPort*** values are unnecessary in the presence of a spanning-tree protocol and therefore are not supported within this document.

5.7.4 *grandTime* formats

Within this document, *grandTime* is represented as a scaled integer seconds value, consisting of 40-bit signed integer (approximately 1,000 generations) and a precise binary fraction, as illustrated in the top of Figure 5.12. All times are located in adjacent bytes, allowing their values to be readily extracted by bridge-resident firmware. All times are represented as binary integers, so no (expensive/imprecise) conversions are necessary before adding and/or multiplying these values.

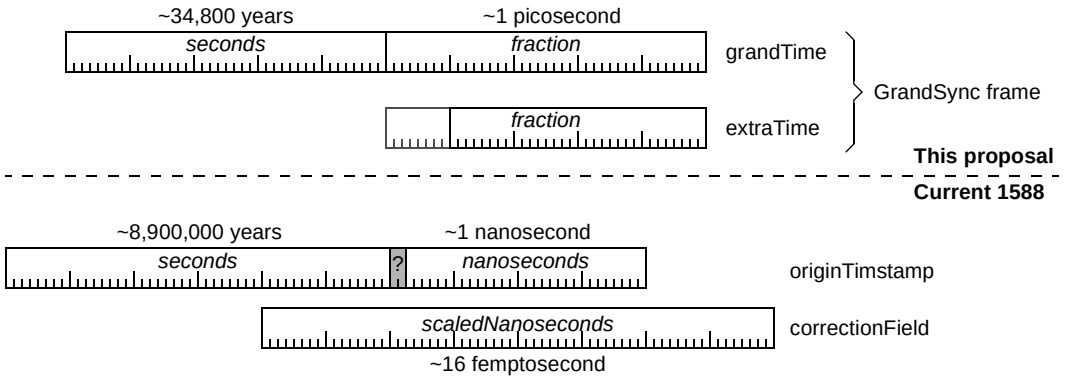

Figure 5.12—*grandTime* formats

Within 1588, (perceived) legacy compatibility requirements led to an interesting collection of formats, as illustrated in the bottom of Figure 5.12. The *originTimestamp* has two unused bits in its center, which wastes accuracy and complicates the processing of special-case values. The *correctionField* further complicates processing requirements, due to its delayed availability and unique overflow-to-seconds behaviors.

5.7.5 Frame sequence comparison

Within this document, all functional are encapsulated into a single slightly-greater-than-minimal frame, slightly greater than the minimal frame size. Each station emits these frames at their link-dependent interval, as illustrated in Figure 5.13.

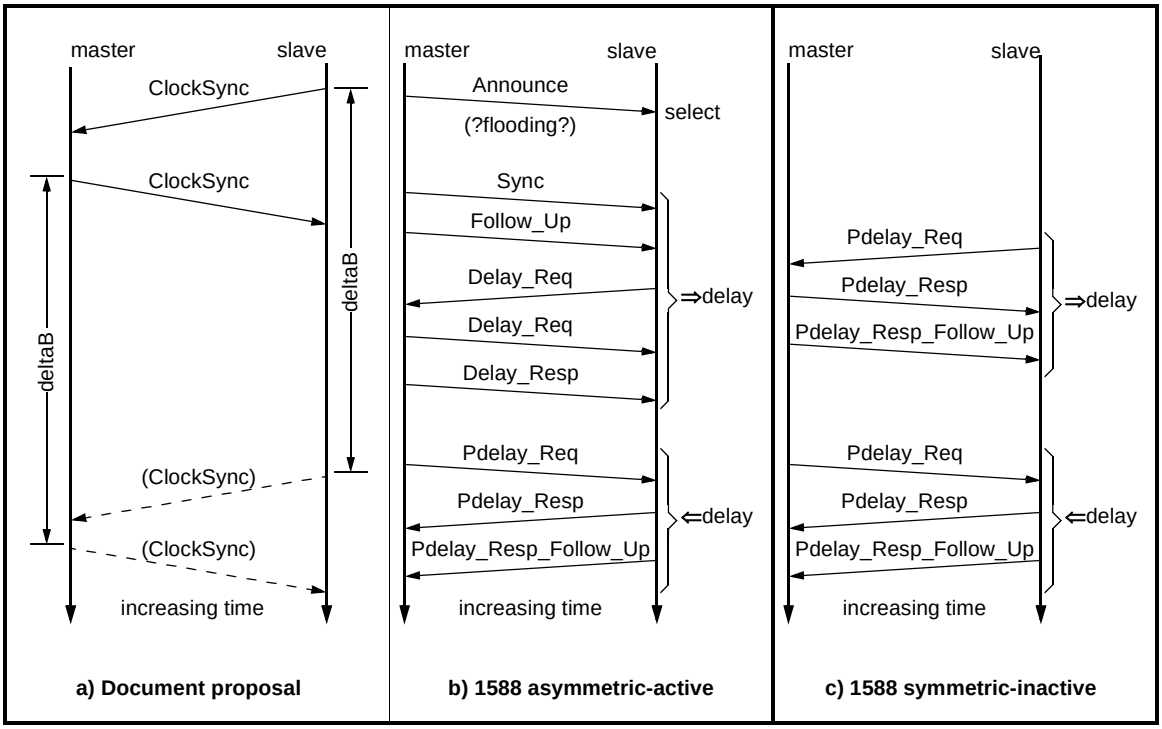

Figure 5.13—Frame sequence comparison

Within 1588, a variety of function/direction specific frames are transmitted. For an asymmetric-active span, this involves nine separate frame transmissions (as opposed to two), as illustrated in Figure 5.13b. For a symmetric-inactive span, this involves six separate frame transmissions, as illustrated in Figure 5.13c. For this illustration, we assume that both master and slave stations desire to calibrate link delays, to minimize topology-change transients.

6. GrandSync operation

6.1 Overview

6.1.1 GrandSync behavior

This clause specifies the state machines that specify GrandSync-entity processing. The operations are described in an abstract way and do not imply any particular implementations or any exposed interfaces. There is not necessarily a one-to-one correspondence between the primitives and formal procedures and the interfaces in any particular implementation.

The GrandSync entity is responsible for monitoring time-sync PDUs via the CLOCK_SYNC.indication service primitive, selectively echoing a subset of these PDUs via the CLOCK_SYNC.response service primitive, as follows:

a) When a preferred time-sync related CLOCK_SYNC.indication arrives:
   1) The grandmaster precedence and port-timeout parameters are saved.
   2) CLOCK_SYNC.indication parameters are echoed in CLOCK_SYNC.response parameters.
   3) The arrival time is recorded, for the purpose of monitoring port timeouts.

b) Arriving non-preferred CLOCK_SYNC.indications are discarded.
   The intent is to echo only PDUs from the currently selected grandmaster port.

c) If the preferred-port timeout is exceeded, the preferred-port parameters are reset.
   The intent is to restart grandmaster selection based on the remaining candidate ports.

6.1.2 GrandSync interface model

The time-synchronization service model assumes the presence of one or more time-synchronized AVB ports communicating with a MAC relay, as illustrated in Figure 6.1. All components are assumed to have access to a common free-running (not adjustable) *localTime* value.

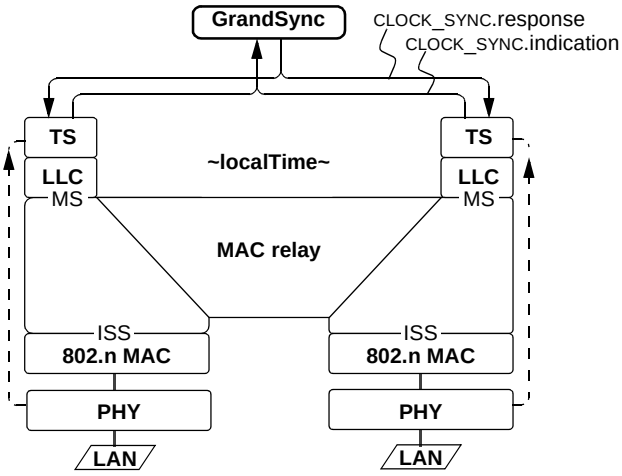

Figure 6.1—GrandSync interface model

A received MAC frame is associated with link-dependent timing information, processed within the TimeSync (TS) state machine, and passed to the GrandSync protocol entity. The GrandSync state machine (illustrated with a darker boundary) is responsible for saving precedence parameters from observed CLOCK_SYNC.indication PDUs and generating CLOCK_SYNC.response PDUs for delivery to other ports.

The ClockSync information exchanged with the GrandSync entity includes {*priority,clockID*} grandmaster-precedence, {*hops,interval,flags,utcOffset*} control, and {*grandTime,extraTime,localTime*} clock-synchronization information, as illustrated in Figure 6.2. A clock-slave end-point can filter the sum of *grandTime* and *extraTime* values, thereby yielding a more accurate image of the globally synchronized *grandTime* value.

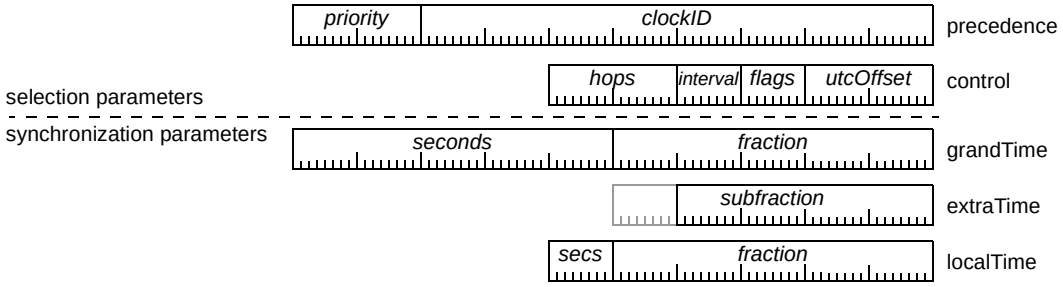

Figure 6.2—GrandSync service-interface components

6.2 Service interface primitives

6.2.1 CLOCK_SYNC.indication

6.2.1.1 Function

Provides the GrandSync protocol entity with clock-synchronization parameters derived from PDUs sent from attached media-dependent ports. The information is sufficient to identify a single clock-slave port (typically the closest-to-grandmaster port) and to disseminate grandmaster supplied clock-synchronization information to other ports.

6.2.1.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
CLOCK_SYNC.indication {
    destination_address,      // Destination address
    source_address,           // Optional
    priority,                 // Forwarding priority
    service_data_unit,        // Delivered content
    {                         // Contents of the service_data_unit
        protocolType,         // Distinguishes AVB frames from others
        function,             // Distinguishes between ClockSync and other AVB frames
        version,              // Distinguishes between ClockSync frame versions
        priority,             // Precedence for grandmaster selection
        clockID,              // Precedence for grandmaster selection
        hops,                 // Distance from the grandmaster station
        interval,             // Nominal ClockSync transmission interval
        flags,                // Control flags
        utcOffset,            // Difference between UTC and TAI timescales
        grandTime,            // Global-time snapshot (1-cycle delayed)
        extraTime,            // Accumulated grandTime error
        localTime             // Local-time snapshot (1-cycle delayed)
    }
}
```

NOTE—The *grandTime* field has a range of approximately 36,000 years, far exceeding expected equipment life-spans. The *localTime* and *linkTime* fields have a range of 256 seconds, far exceeding the expected ClockSync frame transmission interval. These fields have a 1 pico-second resolution, more precise than the expected hardware snapshot capabilities. Future time-field extensions are therefore unlikely to be necessary in the future.

The parameters of the CLOCK_SYNC.indication are described as follows:

6.2.1.2.1 destination_address: A 48-bit field that allows the frame to be conveniently stripped by its downstream neighbor. The destination_address field contains an otherwise-reserved group 48-bit MAC address (TBD).

6.2.1.2.2 source_address: A 48-bit field that specifies the local station sending the frame. The source_address field contains an individual 48-bit MAC address (see 3.10), as specified in 9.2 of IEEE Std 802-2001.

6.2.1.2.3 *priority*: Specifies the (802.3) priority associated with content delivery.

6.2.1.2.4 *serviceDataUnit*: A multi-byte field that provides information content.

For GrandSync-entity time-sync interchanges, the *serviceDataUnit* consists of the following subfields:

6.2.1.2.5 *protocolType*: A 16-bit field contained within the payload that identifies the format and function of the following fields.

6.2.1.2.6 *function*: An 8-bit field that distinguishes the ClockSync frame from other AVB frame type.

6.2.1.2.7 *version*: An 8-bit field that identifies the version number associated with of the following fields.
TBD—A more exact definition of version is needed.

6.2.1.2.8 *priority*: A 16-bit field that can be configured by the user and overrides the remaining *precedence*-group field value.

6.2.1.2.9 *clockID*: A 64-bit globally-unique field that ensures a unique precedence value for each potential grandmaster, should the *priority* field have the same value (see 6.2.1.4).

6.2.1.2.10 *hops*: A 16-bit field that represents the number of hops from the grandmaster.

6.2.1.2.11 *interval*: An 8-bit pseudo floating-point field that specifies the nominal interval between ClockSync frame transmissions (see 6.2.1.5).

6.2.1.2.12 *flags*: An 8-bit field that warns/triggers changes to the end-station *utcOffset* value.

6.2.1.2.13 *utcOffset*: A 16-bit field that represents the current leap-seconds value.

NOTE—Due to the reduced-value and non-time-sensitive nature of the *utcOffset* field, perhaps this could be considered to be a value that is updated through infrequent TLV-like updates.

6.2.1.2.14 *grandTime*: An 80-bit field that specifies a grandmaster synchronized time (see 6.2.1.6).

6.2.1.2.15 *extraTime*: A 32-bit field that specifies the cumulative grandmaster synchronized-time error. (Propagating *extraTime* and *grandTime* separately eliminates whiplash associated with cascaded PLLs.)

6.2.1.2.16 *localTime*: A 48-bit field that specifies the local free-running time within this station, when the previous ClockSync frame was received (see 6.2.1.9).

6.2.1.3 Version format

For compatibility with existing 1588 time-snapshot, a single bit within the version field is constrained to be zero, as illustrated in Figure 6.3. The remaining *versionHi* and *versionLo* fields shall have the values of 0 and 1 respectively.

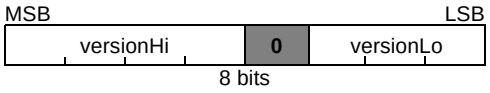

Figure 6.3—*version* subfield format

6.2.1.4 *clockID* subfields

The 64-bit *clockID* field is a unique identifier. For stations that have a uniquely assigned 48-bit *macAddress*, the 64-bit *clockID* field is derived from the 48-bit MAC address, as illustrated in Figure 6.4.

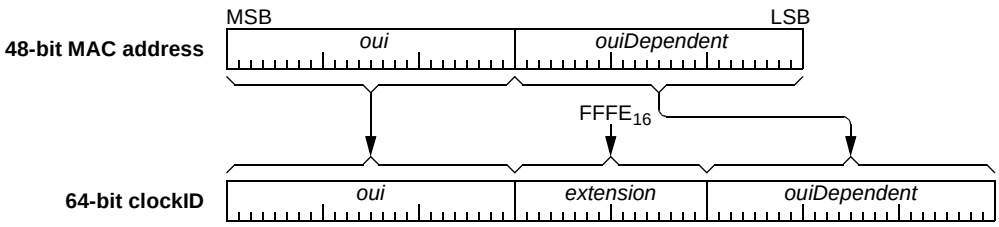

Figure 6.4—*clockID* as derived from a 48-bit MAC address

6.2.1.4.1 *oui*: A 24-bit field assigned by the IEEE/RAC (see 3.10.1).

6.2.1.4.2 *extension*: A 16-bit field assigned to encapsulated EUI-48 values.

6.2.1.4.3 *ouiDependent*: A 24-bit field assigned by the owner of the *oui* field (see 3.10.2).

6.2.1.5 *interval* subfields

The ***interval*** field specifies the source-station's nominal interval between ClockSync frames, and is used for timeout purposes. The ***interval*** field consists of two subfields, as illustrated in Figure 6.5.

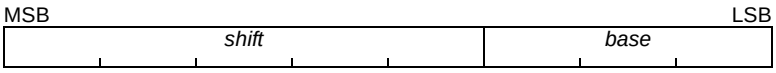

Figure 6.5—*interval* subfields

The two subfields specify the interval duration as a simple floating-point like value:

```
value = (shift >= 0) ? (base << shift) : (base >> -shift);
```

6.2.1.6 *flags* format

The ***flags*** field consists of multiple fields that support the warnings of change indications, as illustrated in Figure 6.6.

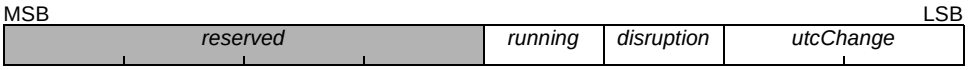

Figure 6.6—*flags* parameters

The ***running*** bit shall be set to 1 when a the receiver has been initialized (cable-length parameters are OK). Otherwise, the *running* bit shall be 0.

The ***disruption*** bit shall be set to 1 when a disruption in distribution of the *grandTime* value is sensed, due to a discontinuity in the *grandTime* source or the selection of the *grandTime* source. Otherwise, the *disruption* bit shall be 0.

The two-bit difference between ***utcChange*** and *utcOffset*, `diff=(utcChange-utcOffset)%4,` specifies the *utcOffset* properties, as follows:

Value	Property
0	Known and stable
1	Increment at midnight UTC
2	Decrement at midnight UTC
3	Unknown

The *utcOffset* field represents the grand-master's vision of the difference between UTC (wall clock) time and TAI (continuous) times.

NOTE—When the value of *utcOffset* is decremented, the event of "midnight" can occur twice, one second apart. The specified/idempotent *utcChange* encoding (that specifies the changed value, not the change amount), eliminates undesirable effects that might otherwise occur due to such "double-clocking" events.

NOTE—To ensure correctness, an exact equation (not simply an English statement) should be provided to ensure proper/consistent updating of UTC time.

6.2.1.7 *grandTime* subfields

The *grandTime* (time-of-day) field within a frame are based on seconds and fractions-of-second values, consistent with IETF specified NTP[B7] and SNTP[B8] protocols, as illustrated in Figure 6.7.

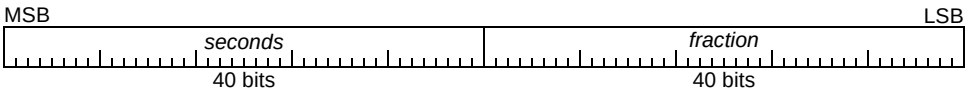

Figure 6.7—*grandTime* subfields

The 40-bit signed ***seconds*** field that specifies time in seconds. The 40-bit unsigned ***fraction*** field that specifies a time offset within each *second*, in units of 2^{-40} second. The concatenation of these fields specifies a 80-bit *grandTime* value, as specified by Equation 6.1.

$$grandTime = seconds + (fraction / 2^{40}) \quad (6.1)$$

6.2.1.8 *extraTime*

The error-time values within a frame are based on a selected portion of a fractions-of-second value, as illustrated in Figure 6.8. The 40-bit signed *fraction* field specifies the time offset within a *second*, in units of 2^{-40} second.

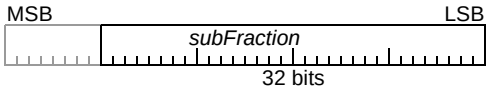

Figure 6.8—extraTime format

6.2.1.9 *localTime* format

The *localTime* value within a frame is based on seconds and fractions-of-second field values, as illustrated in Figure 6.9. The 48-bit *fraction* field specifies the time offset within the *second*, in units of 2^{-48} second.

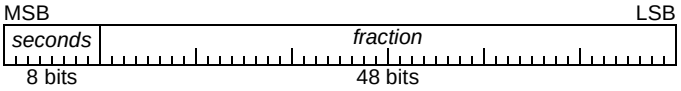

Figure 6.9—*localTime* format

6.2.1.10 When generated

The CLOCK_SYNC.indication service primitive is generated when new grandmaster selection or clock distribution information is available. Such information could change the selection of the grandmaster or could provide a more-recent {*grandTime, localTime*} affiliation necessary for maintaining accurate grandmaster synchronized time references.

6.2.1.11 Effect of receipt

Receipt of the service primitive by the GrandSync entity triggers an update of the grandmaster selection information. If the grandmaster selection determines the source-port to be the preferred port, its provided {*grandTime,localTime*} time affiliation is also echoed to the attached entities, via invocation of the CLOCK_SYNC.response service primitive.

6.2.2 CLOCK_SYNC.response

6.2.2.1 Function

Communicates GrandSync protocol-entity supplied information to attached media-dependent ports. The information is sufficient for attached ports to update/propagate grandmaster clock-synchronization parameters.

6.2.2.2 Semantics of the service primitive

The semantics of the primitives are as specified for CLOCK_SYNC.indication (see 6.2.1).

6.2.2.3 When generated

Generated by the GrandSync entity upon receipt of a time-sync related CLOCK_SYNC.indication from a preferred (by grandmaster selection protocol) source port.

6.2.2.4 Effect of receipt

Receipt of the service primitive by a ClockSlave or TS entity updates entity storage. This storage update allows the destination-port to provide accurate {*grandTime*,*extraTime*,*localTime*} affiliations during later time-sync information transmissions.

6.3 GrandSync state machine

6.3.1 Function

The GrandSync state machine is responsible for observing CLOCK_SYNC.indication parameters, selecting PDUs with preferred time-sync content, and echoing this content in following CLOCK_SYNC.response parameters.

6.3.2 State machine definitions

6.3.2.1 AVB identifiers: Assigned constants used to specify AVB frame parameters.

6.3.2.1.1 AVB_FUNCTION: The function code that corresponds to a time-sync frame.
value—TBD.

6.3.2.1.2 AVB_MCAST: The multicast destination address corresponding to the adjacent neighbor.
value—TBD.

6.3.2.1.3 AVB_TYPE: The *protocolType* corresponding that uniquely identifies time-sync SDUs.
value—TBD.

6.3.2.1.4 AVB_VERSION: The number that uniquely identifies this version of time-sync SDUs.
value—TBD.

6.3.2.2 MAX_HOPS: A constant that specifies the largest possible *hops* value.
value—65536

6.3.2.3 NULL: A constant that (by design) cannot be confused with a valid value.

6.3.2.4 ONES: A large constant wherein all binary bits of the numerical representation are set to one.

6.3.2.5 Q_GS_RX: The queue identifier associated with PDUs sent to the GrandSync.

6.3.2.6 Q_GS_TX: The queue identifier associated with PDUs sent from the GrandSync.

6.3.3 State machine variables

6.3.3.1 *ePtr*: A pointer to entity-dependent storage, where that storage comprises the following:
lastTime—Time of the last best-precedence update, used for timeout purposes.
priority, *clockID*—Copies of like-named fields within the last best-precedence GrandSync PDU.

6.3.3.2 *localTime*: A shared value representing current time within each station.
Within the state machines of this standard, this is assumed to have two components, as follows:
seconds—An 8-bit unsigned value representing seconds.
fraction—An 40-bit unsigned value representing portions of a second, in units of 2^{-40} second.

6.3.3.3 *new, old*: Local variables consisting of concatenated *precedence*, *hops*, and *port* parameters.

6.3.3.4 *rsPtr*: A pointer to the service-data-unit portion of *rxInfo* storage.

6.3.3.5 *rxInfo*: Parameters associated with an CLOCK_SYNC.indication (see 6.2.1.2), comprising:
destination_address, *source_address*, *service_data_unit*
Where *service_data_unit* comprises:
protocolType, *function*, *version*, *priority*, *clockID*,
hops, *interval*, *flags*, *utcOffset*, *grandTime*, *extraTime*,

6.3.3.6 *rxPtr*: A pointer to the *rxInfo* storage.

6.3.3.7 *tsPtr*: A pointer to the service-data-unit portion of *txInfo* storage.

6.3.3.8 *txInfo*: Parameters associated with an CLOCK_SYNC.response (see 6.2.1.2), comprising:
destination_address, *source_address*, *service_data_unit*
Where *service_data_unit* comprises:
protocolType, *function*, *version*, *priority*, *clockID*, *hops*,
interval, *flags*, *utcOffset*, *grandTime*, *extraTime*,

6.3.3.9 *txPtr*: A pointer to the *txInfo* storage.

6.3.4 State machine routines

6.3.4.1 *ClockSyncSdu*(*info*): Checks the frame contents to identify CLOCK_SYNC.indication frames.
TRUE—The frame is a ClockSync frame.
FALSE—Otherwise.

6.3.4.2 *Dequeue*(*queue*): Returns the next available frame from the specified queue.
info—The next available parameters.
NULL—No parameters available.

6.3.4.3 *Enqueue*(*queue, info*): Places the *info* parameters at the tail of the specified queue on all ports.

6.3.4.4 *Expand*(*interval*): Expands the 1-byte encoded *interval* argument into a scaled seconds value.

6.3.4.5 *Precedence*(*priority, clockID*):
Forms a 10-byte *precedence* by concatenating:
priority (2 bytes)
clockID (8 bytes)

6.3.4.6 *StationTime*(*ePtr*): Returns the value of the station's shared local timer, encoded as follows:
seconds—A 2-byte unsigned value representing seconds.
fraction—A 6-byte unsigned value representing portions of a second, in units of 2^{-48} second.

6.3.5 GrandSync state table

The GrandSync state machine includes a media-dependent timeout, which effectively restarts the grandmaster selection process in the absence of received ClockSync frames, as specified by Table 6.1.

Table 6.1—GrandSync state table

Current		Row	Next	
state	**condition**		**action**	**state**
START	(rxInfo = Dequeue(Q_GS_RX)) != NULL	1	—	TEST

	(localTime – ePtr->timer) > 4 * ePtr->interval	2	ePtr->lastTime = localTime;
			ePtr->priority = ONES;
			ePtr->clockID = ONES;
				START

	—	3	localTime = StationTime();	
TEST	ClockSyncSdu(rsPtr)	4	test = Precedence(rsPtr->priority, rsPtr->clockID); last = Precedence(ePtr->priority, ePtr->clockID);	SERVE
	—	5	—	START

SERVE	test <= last	6	ePtr->lastTime = localTime;
			ePtr->interval = Expand(rsPtr->interval);
			ePtr->priority = rsPtr->priority;
			ePtr->clockID = rsPtr->clockID;
			ePtr->hops = rsPtr->hops;
			*txPtr = *rxPtr;
			txPtr->destination_address = AVB_MCAST;
			txPtr->source_address = MacAddress(ePtr);
			tsPtr->protocolType = AVB_TYPE;
			tsPtr->function = AVB_FUNCTION;
			tsPtr->version = AVB_VERSION;
			Enqueue(Q_GS_TX, txPtr);
				START

	—	7	—	

Row 6.1-1: Available indication parameters are processed.
Row 6.1-2: The absence of indications forces a timeout, after a entity-dependent delay
Row 6.1-3: Wait for changes of conditions.

Row 6.1-4: Still-active time-sync PDUs are processed further, based on grandmaster precedences. The *test* and *last* precedence values consist of {*priority,clockID*} components.
Row 6.1-5: Other PDUs and over-aged indications are discarded.

Row 6.1-6: Reset the timeout timer; broadcast saved parameters to all ports (including the source).
Row 6.1-7: Lower precedence indications are ignored.

7. Attached-clock state machines

7.1 Overview

This clause specifies the state machines that specify ClockMaster entity processing. The operations are described in an abstract way and do not imply any particular implementations or any exposed interfaces. There is not necessarily a one-to-one correspondence between the primitives and formal procedures and the interfaces in any particular implementation.

7.1.1 ClockMaster behaviors

The ClockMaster entity is responsible for forwarding the grandmaster time supplied by the ClockSource via the CLOCK_MASTER.request service primitive, as follows:

a) A count value (that is incremented in sequential CLOCK_MASTER.request PDUs) is checked.

b) The grandmaster time parameter within the CLOCK_SYNC.response[*n*+1] PDU is associated with the CLOCK_SYNC.response[*n*] PDU arrival time.

c) The CLOCK_SYNC.response parameters are supplemented to form a CLOCK_SYNC.response PDU, which is then passed to the GrandSync entity.

7.1.2 ClockSlave behaviors

The ClockSlave entity is responsible for extracting the grandmaster time from CLOCK_SYNC.indications and supplying the current value to the ClockTarget entity through the CLOCK_SLAVE service interfaces, as follows:

a) Grandmaster time samples are extracted from GrandSync-supplied CLOCK_SYNC.response PDUs, and saved for computing grandmaster times in following CLOCK_SLAVE.indication PDUs.

b) When triggered by a CLOCK_SLAVE.request indication, a CLOCK_SLAVE.indication PDU is delivered to the ClockTarget state machine. That returned CLOCK_SLAVE.indication PDU supplies the grandmaster time associated with the CLOCK_SLAVE.request invocation time.

7.1.3 ClockMaster interface model

The time-synchronization service model assumes the presence of one or more grandmaster capable entities communicating with the GrandSync state machine, as illustrated on the left side of Figure 7.1. Most grandmaster capable ports are expected to also provide clock-slave functionality, so that any non-selected grandmaster-capable station can synchronize to the selected grandmaster station. However, a station may have only ClockSlave (not ClockMaster) capabilities.

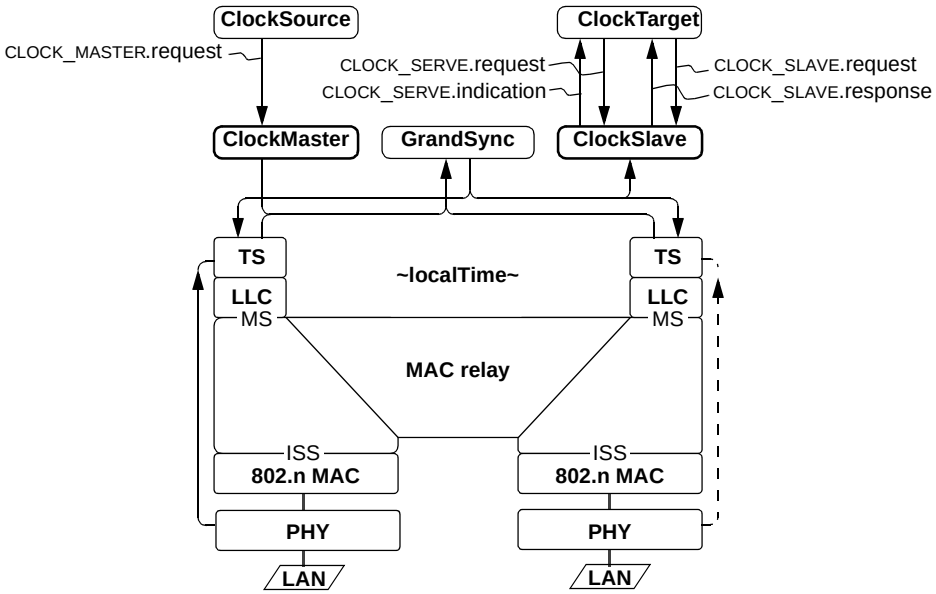

Figure 7.1—ClockMaster interface model

The clock-master ClockMaster state machine (illustrated with an italics name and darker boundary) is responsible for monitoring its port's CLOCK_MASTER.request PDUs and sending CLOCK_SYNC.indication PDUs. The sequencing of this state machine is specified by Table 7.1.

The ClockSlave state machine (illustrated with an italics name and darker boundary) is responsible for saving time parameters from echoed CLOCK_SYNC.response frames and servicing CLOCK_MASTER.request PDUs supplied by the associated clock-slave interface. The sequencing of this state machine is specified by Table 7.2.

The time-synchronization service model assumes the presence of one or more clock-slave capable time-sync entities communicating with a GrandSync protocol entity, as illustrated on the top-side of Figure 7.1. A non-talker clock-slave capable entity is not required to be grandmaster capable.

7.2 ClockMaster service interfaces

7.2.1 Shared service interfaces

The ClockMaster entity is coupled to the bridge ports TS entities via the defined time-sync related CLOCK_SYNC.indication service interface (see 6.2.1).

7.2.2 CLOCK_SOURCE.request service interface

7.2.2.1 Function

Provides the ClockMaster entity with clock-synchronization parameters derived from the reference clock. The information is sufficient to provide the ClockMaster with accurate {*grandTime,localTime*} associations.

7.2.2.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
CLOCK_MASTER.request {
    interval,          // Nominal ClockSync transmission interval
    flags,             // Control flags
    utcOffset,         // Difference between UTC and TAI timescales
    grandTime,         // Global-time snapshot
}
```

The parameters of the CLOCK_MASTER.request service-interface primitive are described as follows:

7.2.2.2.1 *interval*: An 8-bit pseudo floating-point field that specifies the nominal interval between ClockSync frame transmissions (see 6.2.1.5).

7.2.2.2.2 *flags*: An 8-bit field that warns/triggers changes to the end-station *utcOffset* value.

7.2.2.2.3 *utcOffset*: A 16-bit field that represents the current difference between UTC and TAI time scales (leap-seconds).

7.2.2.2.4 *grandTime*: An 80-bit field that specifies the ClockSource provided grandmaster time when the CLOCK_MASTER.request interface was invoked (see 6.2.1.7).

7.2.2.3 When generated

The CLOCK_MASTER.request service primitive is invoked by a client-resident ClockSource entity. The intent is to provide the ClockMaster with continuous/accurate updates from a ClockSource-resident clock reference.

7.2.2.4 Effect of receipt

Upon receipt by the ClockMaster entity, the encapsulated *grandTime* value is supplemented and affiliated with a *localTime* snapshot; the resulting {*grandTime,extraTime,localTime*} affiliation is passed to the GrandSync entity for redistribution to other ClockSlave and TS entities.

7.3 ClockMaster state machine

7.3.1 State machine definitions

7.3.1.1 AVB identifiers: Assigned constants used to specify AVB frame parameters (see 6.3.2).
AVB_FUNCTION, AVB_MCAST, AVB_TYPE, AVB_VERSION

7.3.1.2 NULL: A constant value that (by design) cannot be confused with a valid value.

7.3.1.3 Q_CM_RX: The queue identifier associated with PDUs sent to the ClockMaster.

7.3.1.4 Q_GS_RX: The queue identifier associated with PDUs sent to the GrandSync.

7.3.2 State machine variables

7.3.2.1 *localTime*: See 6.3.3.

7.3.2.2 *sxPtr*: A pointer to the service-data-unit portion of the received PDU storage.

7.3.2.3 *tsPtr*: A pointer to the service-data-unit portion of *txInfo* storage.

7.3.2.4 *txInfo*: Storage for to-be-transmitted CLOCK_SYNC.response parameters (see 6.2.2.2), comprising:
destination_address, source_address, service_data_unit
Where *service_data_unit* comprises:
protocolType, function,version, priority, clockID, hops,
grandTime, extraTime, localTime, interval, flags, utcOffset

7.3.2.5 *txPtr*: A pointer to *txInfo* storage.

7.3.3 State machine routines

7.3.3.1 *Dequeue*(*queue*): See 6.3.4

7.3.3.2 *Enqueue*(*queue, info*): See 6.3.4

7.3.3.3 *StationTime*(*entity*): See 6.3.4

7.3.3.4 *ClockSyncSdu*(*info*): See 6.3.4.

7.3.4 ClockMaster state table

The ClockMaster state table encapsulates clock-provided sync information into a ClockSync PDU, as illustrated in Table 7.1.

Table 7.1—ClockMaster state machine table

Current		Row	Next	
state	**condition**		**action**	**state**

START	(rxInfo = Dequeue(Q_CM_RX)) != NULL	1	txPtr->destination_address = AVB_MCAST;
			txPtr->source_address = MacAddress(ePtr);
			tsPtr->prototolType = AVB_TYPE;
			tsPtr->function = AVB_FUNCTION;
			tsPtr->version = AVB_VERSION;
			tsPtr->priority = ePtr->priority;
			tsPtr->clockID = ePtr->clockID;
			tsPtr->hops = 0;
			tsPtr->grandTime = sxPtr->grandTime;
			tsPtr->extraTime = 0;
			tsPtr->localTime = localTime;
			tsPtr->interval = sxPtr->interval;
			tsPtr->flags = sxPtr->flags;
			tsPtr->utcOffset = sxPtr->utcOffset;
			Enqueue(Q_GS_RX, txInfo);
				START

	—	2	localTime = StationTime(ePtr);	

Row 7.1-1: Supplement and retransmit on CLOCK_MASTER.request PDU arrival.
Row 7.1-2: Wait for the next change of state.

7.4 ClockSlave service interfaces

7.4.1 Shared service interfaces

The ClockSlave entity is coupled to the GrandSync entity, via the defined CLOCK_SYNC.response service interface (see 6.2.2).

7.4.2 CLOCK_SLAVE.request service interface

7.4.2.1 Function

Triggers the ClockSlave entity to provide the current *grandTime* value.

7.4.2.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
CLOCK_SLAVE.request {        // The invocation has no parameters
}
```

7.4.2.3 When generated

The CLOCK_SLAVE.request service primitive is invoked by a client-resident ClockTarget entity. The intent is to trigger the ClockSlave's invocation of a following CLOCK_SLAVE.response primitive, thus providing the ClockTarget entity with a recent *grandTime* value.

7.4.2.4 Effect of receipt

Upon receipt by a ClockSlave entity, a copy of the current *grandTime* value is returned.

7.4.3 CLOCK_SLAVE.response service interface

7.4.3.1 Function

Provides the ClockTarget entity with current *grandTime* value derived from the reference clock.

7.4.3.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
CLOCK_SLAVE.response {
    interval,           // Nominal ClockSync transmission interval
    flags,              // Control flags
    utcOffset,          // Difference between UTC and TAI timescales
    grandTime           // The grandmaster time, when the indication is sent.
}
```

The parameters of the CLOCK_SLAVE.response service-interface primitive are described as follows:

7.4.3.2.1 *grandTime*: An 80-bit field that specifies the grandmaster synchronized time within the ClockSlave entity, when the previous CLOCK_SLAVE.request service-interface was invoked.

7.4.3.3 When generated

The invocation of the CLOCK_SLAVE.response service primitive is invoked by the receipt of a ClockTarget supplied CLOCK_SLAVE.request PDU. The intent is to provide the ClockTarget entity with a current *grandTime* value.

7.4.3.4 Effect of receipt

Upon receipt by a ClockTarget entity, the {*grandTime*,*localTime*} affiliation is expected to be saved and (along with previously saved copies) used to adjust the rate of the grandmaster synchronized ClockTarget-resident clock.

7.4.4 CLOCK_SERVE.request service interface

NOTE—The details of the CLOCK_SERVE interface are highly preliminary and subject to change.

7.4.4.1 Function

Triggers the ClockSlave entity to provide the current *grandTime* value.

7.4.4.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
CLOCK_SERVE.request {
    interval            // Interval between returned indications
}
```

7.4.4.3 When generated

The CLOCK_SERVE.request service primitive is invoked by a client-resident ClockTarget entity. The intent is to trigger the ClockSlave's invocation of a following CLOCK_SERVE.indication primitive, thus providing the ClockTarget entity with periodic samples of the current *grandTime* value.

7.4.4.4 Effect of receipt

Upon receipt by a ClockSlave entity, a copy of the current *grandTime* value is returned.

7.4.5 CLOCK_SERVE.indication service interface

7.4.5.1 Function

Provides the ClockTarget entity with periodic *grandTime* samples derived from the reference clock.

NOTE—Details of the CLOCK_SERVE interface are under discussion and subject to change. On possibility is the ClockTarget could requests samples-per-second or seconds-per-sample to avoid arithmetic approximations, wherein decimal/binary fractions do not have exact numerical equivalents.

7.4.5.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
CLOCK_SERVE.indication {
    interval,          // Nominal ClockSync transmission interval
    flags,             // Control flags
    utcOffset,         // Difference between UTC and TAI timescales
    grandTime          // The grandmaster time, when the indication is sent.
}
```

The parameters of the CLOCK_SLAVE.indication service-interface primitive are described as follows:

7.4.5.2.1 *grandTime*: An 80-bit field that specifies the grandmaster synchronized time within the ClockSlave entity, when the previous CLOCK_SLAVE.request service-interface was invoked.

7.4.5.3 When generated

The invocation of the CLOCK_SLAVE.indication service primitive is invoked by the receipt of a ClockTarget supplied CLOCK_SLAVE.request PDU. The intent is to provide the ClockTarget entity with a current *grandTime* value.

7.4.5.4 Effect of receipt

Upon receipt by a ClockTarget entity, the {*grandTime*,*localTime*} affiliation is expected to be saved and (along with previously saved copies) used to adjust the rate of the grandmaster synchronized ClockTarget-resident clock.

7.5 ClockSlave state machine

7.5.1 Function

To provide time…TBD.

7.5.2 State machine definitions

7.5.2.1 NULL: A constant that (by design) cannot be confused with a valid value.

7.5.2.2 Q_CS_IND: The queue identifier associated with periodic PDUs sent from the ClockSlave.

7.5.2.3 Q_CS_REQ: The queue identifier associated with PDUs sent to the ClockSlave.

7.5.2.4 Q_CS_RES: The queue identifier associated with per-request PDUs sent from the ClockSlave.

7.5.2.5 Q_GS_TX: The queue identifier associated with PDUs sent from the GrandSync.

7.5.3 State machine variables

7.5.3.1 *ePtr*: A pointer to entity-dependent information, including the following:
rxSaved—A copy of the GrandSync supplied MA_DATAUNIT.request value.
interval—The expected service rate of CLOCK_SLAVE.request services.
baseTimer—Recently saved time events, each consisting of the following:
index—Index into the *timed*[ ] array, where last times were stored.
range—Number of entries within the *timed*[ ] array
timed[range]—Recently saved time events, each consisting of the following:
grandTime—A previously sampled grandmaster synchronized time.
extraTime—The residual error associated with the sampled *grandTime* value.
localTime—The station-local time affiliated with the sampled *grandTime* value.

7.5.3.2 *cxInfo*: A contents of a higher-level supplied time-synchronization request, including the following:
frameCount—A value that increments on each CLOCK_MASTER.request PDU transfer.

7.5.3.3 *nextTime*: Storage representing *grandTime* and *extraTime* values returned from call to *NextTimed*( ).

7.5.3.4 *rsPtr*: A pointer to the service-data-unit portion of *rxInfo*.

7.5.3.5 *rxInfo:* A contents of a GrandSync supplied CLOCK_SYNC.response (see 6.2.2), including:
destination_address, *source_address*, *service_data_unit*
Where *service_data_unit* comprises:
protocolType, *function*, *version*, *priority*, *clockID*,
hops, *interval*, *flags*, *utcOffset*, *hops*, *grandTime*, *extraTime*, *localTime*

7.5.3.6 *rxPtr*: A pointer to *rxInfo*.

7.5.3.7 *intervalRx*: The synchronization interval of this station's GrandSync-selected clock-slave port.

7.5.3.8 *localTime*: See 6.3.3.

7.5.3.9 *ssPtr*: A pointer to the service-data-unit portion of the *ePtr->rxSaved* storage

7.5.3.10 *sxPtr*: A pointer to the *ePtr->rxSaved* storage

7.5.3.11 *timePtr*: A pointer to the *ePtr->timed*[ ] array storage

7.5.3.12 *txInfo*: A contents of a ClockSlave supplied CLOCK_SLAVE.indication (see 6.2.2), comprising:
frameCount—The saved value of the like named field from the previous CLOCK_SLAVE.request PDU.
grandTime—The grandmaster synchronized time sampled during the CLOCK_SLAVE.request transfer.

7.5.3.13 *txPtr*: A pointer to *txInfo* storage.

7.5.3.14 *intervalTx*: The synchronization interval of this ClockSlave entity.

7.5.4 State machine routines

7.5.4.1 *Dequeue*(*queue*): See 6.3.4.

7.5.4.2 *Enqueue*(*queue, info*): See 6.3.4.

7.5.4.3 *NextSaved*(*btPtr, intervalRate, grandTime, extraTime, thisRxTime*):
Saves *grandTime*, *extraTime* values associated with a snapshot taken at *thisRxTime*, with the saved values spanning a *intervalRate* specified interval.

7.5.4.4 *NextTimed*(*btPtr, localTime, intervalBack*):
Returns *grandTime* and *extraTime* values associated with a snapshot taken at *localTime*, back-interpolated

by a *intervalBack* time, based on previous received-time information saved in the *btPtr* referenced data structure.

7.5.4.5 *StationTime*(*entity*): See 6.3.4.

7.5.4.6 *ClockSyncSdu*(*info*): See 7.3.3.

7.5.5 ClockSlave state table

NOTE—Details of CLOCK_SERVE interface are under discussion and thus this state machine is subject to change.

The ClockSlave state machine includes a media-dependent timeout, which effectively disconnects a clock-slave port in the absence of received ClockSync frames, as illustrated in Table 7.2.

Table 7.2—ClockSlave state table

Current		Row	Next	
state	**condition**		**action**	**state**
START	(rxInfo = Dequeue(Q_GS_TX)) != NULL	1	—	TEST

	((cxInfo = Dequeue(Q_CS_RX)) != NULL	2	intervalRx = ssPtr->interval;
			intervalTx = ePtr->interval;
			intervalBack = (3 * intervalRx + intervalTx) / 2;
			nextTimes = NextTimed(btPtr, localTime, intervalBack);
			txPtr->count = cxInfo.count;
			txPtr->grandTime = nextTimes.grandTime + nextTimes.extraTime;
			Enqueue(Q_CS_IND, txInfo);
				START

	—	3	localTime = StationTime(ePtr);	
TEST	ClockSyncSdu(rsPtr)	4	*sxPtr = *rxPtr; NextSaved(btPtr, intervalRate, rsPtr->grandTime; rsPtr->extraTime, rsPtr->localTime);	START
	—	5	—	

Row 7.2-1: The received CLOCK_SYNC.response parameters are dequeued for checking.
Row 7.2-2: A clock-slave request generates an affiliated information-providing indication.
The affiliated indication has the sequence-count information provided by the request.
The delivered end-point *grandTime* value is the sum of delivered *grandTime* and *extraTime* values.
The requested content is queued for delivery to the higher-level client.
Row 7.2-3: Wait for the next change-of-conditions.

Row 7.2-4: Validated GrandSync entity requests are accepted; its time parameters are saved.
The back-interpolation time is estimated from the *interval* times of the source and clock slave.
(This back-interpolation time is used by *NextTimed*( ), which provides transmission-time estimates.)
Row 7.2-5: Wait for the next change-of-conditions.

8. Ethernet full duplex (EFDX) state machines

8.1 Overview

This clause specifies the state machines that support 802.3 Ethernet full duplex (EFDX) bridges. The operations are described in an abstract way and do not imply any particular implementations or any exposed interfaces. There is not necessarily a one-to-one correspondence between the formal specification and the interfaces in any particular implementation.

8.1.1 EFDX link indications

The duplex-link TimeSyncRxEfdx state machines are provided with snapshots of ClockSync-frame reception and transmission times, as illustrated by the ports within Figure 8.1. These link-dependent indications can be different for bridge ports attached to alternative media.

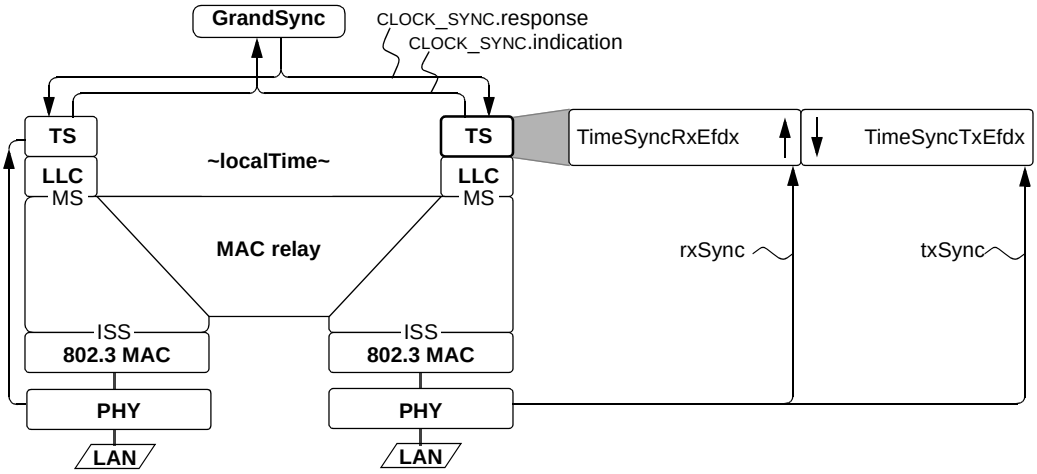

Figure 8.1—EFDX-link interface model

8.1.2 Link-delay compensation

Synchronization accuracies are affected by the transmission delays associated with transmissions over links between bridges. To compensate for these transmission delays, the receive port is responsible for compensating {*grandTime*,*extraTime*,*localTime*} affiliations by the (assumed to be constant) frame-transmission delay.

8.1.3 Clock-synchronization intervals

Receive timestamp hardware is assumed to affiliate a *thisRxTime* (a *localTime* snapshot, sampled on frame receipt) with an incoming frame, before that frame is passed to the receive-TS entity. In a similar fashion, transmit timestamp hardware is assumed to save a *thisTxTime* value in the transmit-TS entity, after each frame transmission.

The details of services interfaces that support these timestamp activities are under discussion and therefore not included in this draft. Until these discussions converge, the protocols simply assume the availability of the *thisRxTime* and *thisTxTime* values, without specifying how these values are updated.

8.1.4 Clock-synchronization intervals

Clock synchronization involves synchronizing the clock-slave clocks to the reference provided by the grand clock master. Tight accuracy is possible with matched-length duplex links, since bidirectional frame transmissions can cancel the cable-delay effects.

Clock synchronization involves the processing of periodic events. Multiple time periods are involved, as listed in Table 8.1. The clock-period events trigger the update of free-running timer values; the period affects the timer-synchronization accuracy and is therefore constrained to be small.

Table 8.1—Clock-synchronization intervals

Name	Time	Description
clock-period	< 50 ns	Resolution of timer-register value updates
send-period	10 ms	Time between sending of periodic ClockSync frames between adjacent stations
slow-period	100 ms	Time between computation of clock-master/clock-slave rate differences

The send-period events trigger the interchange of ClockSync frames between adjacent stations. While a smaller period (1 ms or 100 μs) could improve accuracies, the larger value is intended to reduce costs by allowing computations to be executed by inexpensive (but possibly slow) bridge-resident firmware.

The slow-period events trigger the computation of timer-rate differences. The timer-rate differences are computed over two slow-period intervals, but recomputed every slow-period interval. The larger 100 ms (as opposed to 10 ms) computation interval is intended to reduce errors associated with sampling of clock-period-quantized slow-period-sized time intervals.

8.2 efdxClockSync frame format

8.2.1 efdxClockSync fields

EFDX clock-synchronization (efdxClockSync) frames facilitate the synchronization of neighboring clock-master and clock-slave stations. The frame, which is normally sent at 10ms intervals, includes time-snapshot information and the identity of the network's clock master, as illustrated in Figure 8.2. The gray boxes represent physical layer encapsulation fields that are common across Ethernet frames.

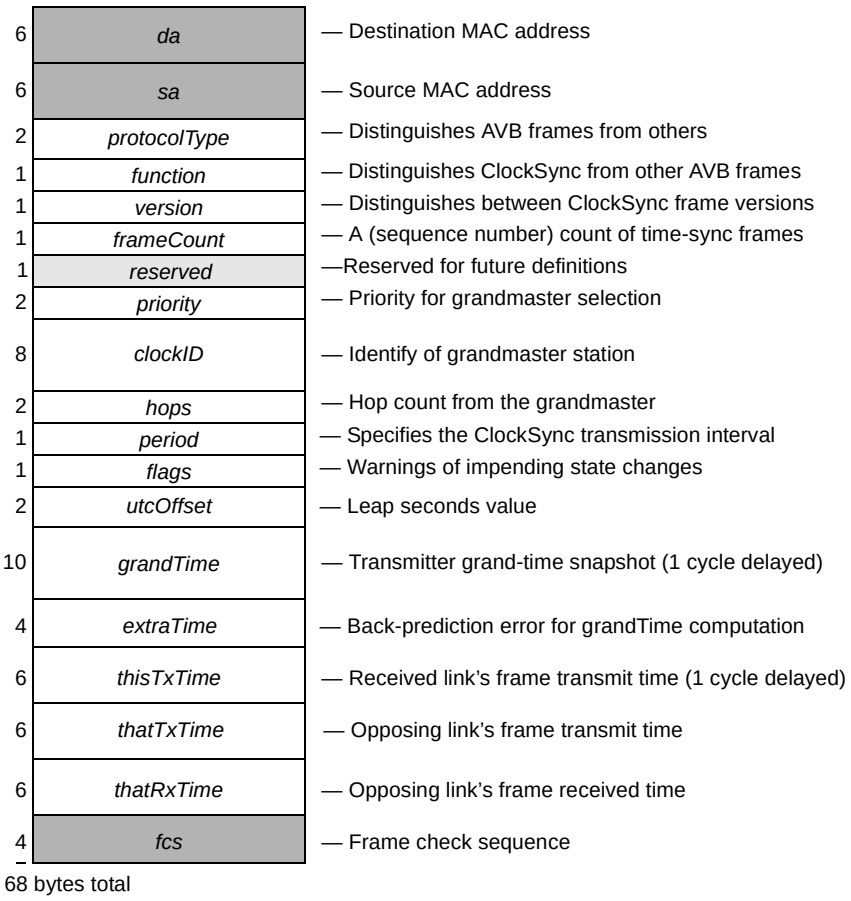

Figure 8.2—efdxClockSync frame format

NOTE— Existing 1588 time-snapshot hardware captures the values between byte-offset 34 and 45 (inclusive). The location of the *frameCount* field (byte-offset 44) has been adjusted to ensure this field can be similarly captured for the purpose of unambiguously associating ClockSync-packet snapshots (that bypass the MAC) and ClockSync-packet contents (that pass through the MAC).

The 48-bit ***da*** (destination address), 48-bit ***sa*** (source address) field, 16-bit ***protocolType,*** 8-bit ***function***, 8-bit ***version***, 2-byte ***priority***, 8-byte ***clockID***, 2-byte ***hops***, 1-byte ***priority***, 1-byte ***flags***, 2-byte ***utcOffset***, 80-bit ***grandTime***, and 32-bit ***extraTime*** field are specified in 6.2.1.2.

8.2.1.1 ***frameCount*****:** An 8-bit field that is incremented by one between successive ClockSync frame transmission.

8.2.1.2 ***thisTxTime:*** A 48-bit field that specifies the local free-running time within the neighbor station, when the previous ClockSync frame was transmitted on the incoming link (see 6.2.1.9).

8.2.1.3 *thatTxTime*: A 48-bit field that specifies the local free-running time within the source station, when the previous ClockSync frame was transmitted on the opposing link (see 6.2.1.9).

8.2.1.4 *thatRxTime*: A 48-bit field that specifies the local free-running time within the target station, when the previous ClockSync frame was received on the opposing link (see 6.2.1.9).

8.2.1.5 *fcs*: A 32-bit (frame check sequence) field that is a cyclic redundancy check (CRC) of the frame.

8.3 EFDX TimeSync service interfaces

8.3.1 Shared service interfaces

The per-port EFDX per-port interfaces are between the TS and frame-generation entities.

8.3.2 ES_UNITDATA .request service interface

8.3.2.1 Function

Provides the TS entity with grandmaster-selection and clock-synchronization parameters derived from a received ClockSync frame.

8.3.2.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
ES_UNITDATA.indication {
    destination_address,      // Destination address
    source_address,           // Optional
    priority,                 // Forwarding priority
    this_rx_time,             // A localTime snapshot, sampled on frame arrival
    service_data_unit,        // Delivered content
    {                         // Contents of the service_data_unit
        protocolType,         // Distinguishes AVB frames from others
        function,             // Distinguishes between ClockSync and other AVB frames
        version,              // Distinguishes between ClockSync frame versions
        priority,             // Precedence for grandmaster selection
        clockID,              // Precedence for grandmaster selection
        hops,                 // Distance from the grandmaster station
        interval,             // Nominal ClockSync transmission interval
        flags,                // Control flags
        utcOffset,            // Difference between UTC and TAI timescales
        grandTime,            // Global-time snapshot (1-cycle delayed)
        extraTime,            // Accumulated grandTime error
        thisTxTime,           // Opposing link's frame transmit time
        thatRxTime,           // Opposing link's frame received time
        thatTxTime            // Received link's frame transmit time (1-cycle delayed)
    }
}
```

The parameters of the ES_UNITDATA.indication are described as follows:

The 48-bit ***destination_address***, 48-bit ***source_address***, and 8-bit ***priority*** fields are specified in 6.2.1.2.

The 48-bit ***this_rx_time*** field is a copy of the localTime value, sampled upon frame arrival.

The service_data_unit consists of subfields; for content exchanged with the GrandTime protocol entity, these fields include the following.

The 16-bit ***protocolType***, 8-bit ***function***, 8-bit ***version***, 2-byte ***priority***, 8-byte ***clockID***, 2-byte ***hops***, 1-byte ***interval***, 1-byte ***flags***, 2-byte ***utcOffset***, 10-byte ***grandTime***, and 4-byte ***extraTime*** fields are specified in 6.2.1.2.

The 6-byte ***thisTxTime***, 6-byte ***thatRxTime***, and 6-byte ***thatTxTime*** fields are specified in 8.2.1.

8.3.2.3 When generated

The ES_UNITDATA.indication service primitive is invoked by the receipt for a ClockSync frame.

8.3.2.4 Effect of receipt

Upon receipt by the TimeSync entity, the media-dependent fields (*thisTxTime*, *this_rx_time*, *thatTxTime* and *thatRxTime*) are utilized to compute the link delay associated with the attached span. These are then stripped from the PDU and replaced by a *localTime* value, providing the GrandSync entity with the link-delay compensated {*grandTime*,*extraTime*,*localTime*} triad necessary for performing clock-clock synchronization at other ports.

8.3.3 ES_UNITDATA.response service interface

8.3.3.1 Function

Provides the per-port transmit entity with grand-master selection and clock-synchronization parameters derived from the GrandSync-echoed ClockSync PDU.

8.3.3.2 Semantics of the service primitive

The semantics of the ES_UNITDATA.response and the ES_UNITDATA.indications are the same (see 8.3.2.2).

8.3.3.3 When generated

The ES_UNITDATA.response service primitive is periodically invoked by the per-port TimeSync entity for the purpose of transmitting a ClockSync frame to the adjacent station.

8.3.3.4 Effect of receipt

Upon receipt by the port-transmit entity, the contents are encapsulated into a GrandSync frame that is sent to the neighbor station.

8.4 TimeSyncRxEfdx state machine

8.4.1 Function

The TimeSyncRxEfdx state machine is responsible for monitoring its port's received MAC-supplied frames and sending GrandSync PDUs. The sequencing of this state machine is specified by Table 8.2; details of the computations are specified by the C-code of Annex G.

8.4.2 State machine definitions

8.4.2.1 DC_DELAY: The default cable-length value, used before length-calibration completes.
value—10 ns (corresponds to an approximate 2 meter cable length)

8.4.2.2 MAX_HOPS: See 6.3.2.2.

8.4.2.3 MOD_FRAME: A constant representing the number of possible *frame*.*frameCount* values.
value—256.

8.4.2.4 MOD_RATE: A constant representing the number of possible time-array values.
default value—16.

8.4.2.5 NULL: A constant that (by design) cannot be confused with a valid value.

8.4.2.6 Q_ES_RX: The queue identifier associated with the received MAC frames.

8.4.2.7 Q_GS_RX: The queue identifier associated with MAC frames sent into GrandSync.

8.4.2.8 RX_PHASE: The number of received initialization phases.
value—2

8.4.3 State machine variables

8.4.3.1 *count*: A transient value representing the expected value of the next *rxInfo*.*frameCount* value.

8.4.3.2 *cxInfo*: A contents of a lower-level supplied time-synchronization poke indication, including:
frameCount—The value of the like-named field within the last ClockSync packet arrival.
localTime—The value of *localTime* associated with the last ClockSync packet arrival.

8.4.3.3 *cxPtr*: A pointer to *cxInfo* storage.

8.4.3.4 *delay*: Values (possibly scaled integers) representing cable-delay times.

8.4.3.5 *ePtr*: A pointer to a data structure that contains port-specific information comprising:
past—A storage-array index for the past saved value.
phase—A saturating integer that counts received GrandSync frames since initialization.
last—A storage-array index for the last saved value.
frameCount—The value of *frameCount* within the last received frame.
times[*N*]—An array of time groups, where each array elements consists of:
rxTime—The receive time associated with received time-sync frames.
txTime—The transmit time associated with received time-sync frames.

8.4.3.6 *localTime*: See 6.3.3.

8.4.3.7 *ratio0, ratio*: Variables representing the ratio of this station's timer to this port's neighbor timer.

8.4.3.8 *roundTrip*: The time between transmit-to-neighbor and receive-from-neighbor events.

8.4.3.9 *rsPtr*: A pointer to the service-data-unit portion of *rxInfo* storage.

8.4.3.10 *rxInfo*: Storage for received time-sync PDUs, comprising:
destination_address, *source_address*, *service_data_unit*
Where *service_data_unit* comprises:
protocolType, *function*, *version*, *frameCount*, *priority*, *clockID*,
hops, *interval*, *flags*, *utcOffset*, *grandTime*, *extraTime*, *localTime*, *thatTxTime*, *thatRxTime*

8.4.3.11 *rxPtr*: A pointer to the *rxInfo* storage.

8.4.3.12 *timePtrA, timePtrB*: Pointers to the past and last *times*[ ] array locations.

8.4.3.13 *timeRxA, timeRxB, timeRxA, timeRxB*: Intermediate receive/transmit time variables.

8.4.3.14 *tsPtr*: A pointer to service-data-unit portion of *txInfo* storage.

8.4.3.15 *turnRound*: The time between receive-at-neighbor and transmit-from-neighbor events.

8.4.3.16 *txInfo*: Storage for information sent to the GrandSync entity, comprising:
destination_address, *source_address*, *service_data_unit*
Where *service_data_unit* comprises:
protocolType, *function*, *version*, *priority*, *clockID*,
hops, *interval*, *flags*, *utcOffset*, *grandTime*, *extraTime*, *localTime*

8.4.3.17 *txPtr*: A pointer to *txInfo* storage.

8.4.4 State machine routines

8.4.4.1 *ClipRatio*(*ratio, deviation*): Clips the *ratio* value to within 1.0+*deviation* and 1.0–*deviation*.

8.4.4.2 *Dequeue*(*queue*): See 6.3.4.

8.4.4.3 *Enqueue*(*queue, info*): See 6.3.4.

8.4.4.4 *Min*(*x, y*): Returns the minimum of *x* and *y* values.

8.4.4.5 *StationTime*(*entity*): See 7.3.3.

8.4.4.6 *ClockSyncSdu*(*info*): See 6.3.4.

8.4.5 TimeSyncRxEfdx state machine table

The TimeSyncRxEfdx state machine processes efdxClockSync PDUs and forwards revised ClockSync PDUs to the GrandSync entity, as illustrated in Table 8.2.

Table 8.2—TimeSyncRxEfdx state machine table

Current		Row	Next	
state	**condition**		**action**	**state**
START	—	1	ePtr->phase = 0	FIRST
FIRST	(rxInfo=Dequeue(Q_ES_RX)) != NULL	2	count = (ePtr->frameCount + 1) % MOD_FRAME; ePtr->frameCount = rxPtr->frameCount;	NEXT
	—	3	localTime = StationTime(ePtr);	FIRST
NEXT	!ClockSyncSdu(rsPtr)	4	Enqueue(Q_ES_TX, rxPtr);	FIRST
	count != rxPtr->frameCount	5	—	
	ePtr->phase < RX_PHASE	6	ePtr->last = ePtr->past = 0;	SAVE
	—	7	—	

SAVE	—	8	thisRxTime = ePtr->thisRxTime0;
			ePtr->thisRxTime0 = rxPtr->thisRxTime;
			timePtrA = &(ePtr->times[ePtr->last]);
			timePtrB = &(ePtr->times[ePtr->past]);
			timeRxB = timePtrB->txTime;
			timeRxB = timePtrB->rxTime;
			timePtrA->rxTime = timeRxA = thisRxTime;
			timePtrA->txTime = timeTxA = rsPtr->thisTxTime;
				PEEK

Table 8.2—TimeSyncRxEfdx state machine table

Current		Row	Next	
state	**condition**		**action**	**state**
PEEK	ePtr->phase < RX_PHASE	9	ePtr->phase = ePtr->phase +1;	FIRST
	rxPtr->hops >= MAX_HOPS	10	—	
	ePtr->past==ePtr->last	11	ePtr->past = (ePtr->past + 1) % MOD_RATE;	SEND
	—	12	—	

SEND	—	13	ePtr->last = (ePtr->last + 1) % MOD_RATE;
			ratio0 = (timeTxA – timeTxB) / (timeRxA – timeRxB);
			ratio = ClipRatio(ratio0, 200PPM);
			roundTrip = thisRxTime – rsPtr->thatTxTime;
			turnRound = rsPtr->thisTxTime – rsPtr->thatRxTime;
			delay = Min(0, roundTrip – (turnRound * ratio));
			txPtr->destination_address=rxPtr->destination_address;
			txPtr->source_address = rxPtr->source_address;
			tsPtr->protocolType = rsPtr->protocolType;
			tsPtr->function = rsPtr->function;
			tsPtr->version = rsPtr->version;
			tsPtr->grandTime = rsPtr->grandTime;
			tsPtr->extraTime = rsPtr->extraTime;
			tsPtr->localTime = thisRxTime – delay;
			tsPtr->hops = rsPtr->hops;
			tsPtr->interval = ePtr->interval;
			tsPtr->flags = ePtr->flags;
			Enqueue(Q_GS_RX, txPtr);
				FIRST

Row 8.2-1: Initialization clears the initialization *phase* to zero, allowing unitialized times to be ignored.
Row 8.2-2: Initiate inspection of frames received from the lower-level MAC.
Row 8.2-3: Update times while waiting for the next change-of-state.

Row 8.2-4: The non-ClockSync frames are passed through.
Row 8.2-5: Non-sequential ClockSync frames are ignored.
Row 8.2-6: Until initialized, clear the circular-buffer index values.
Row 8.2-7: Otherwise, use previous circular-buffer index values.
Row 8.2-8: Save incoming clockSync information.

Row 8.2-9: Step through the initialization phases, based on received-frame count.
Row 8.2-10: Discard over-aged ClockSync frames.
Row 8.2-11: Update tail-index, to avoid circular-buffer overflow.
Row 8.2-12: Otherwise, allow the circular-buffer depth to grow.
Row 8.2-13: Send the revised PDU.

8.5 TimeSyncTxEfdx state machine

8.5.1 Function

The TimeSyncTxEfdx state machine is responsible for saving time parameters from GrandSync-echoed ClockSync PDUs and forming efdxClockSync PDUs for transmission through the MAC and over the attached link.

8.5.2 State machine definitions

8.5.2.1 MAX_HOPS: See 6.3.2.2.

8.5.2.2 NULL: A constant that (by design) cannot be confused with a valid value.

8.5.2.3 Q_ES_TX: The queue identifier associated with frames sent to the MAC.

8.5.2.4 Q_GS_TX: The queue identifier associated with frames sent from the GrandSync.

8.5.2.5 T10ms: A constant the represents a 10 ms value.

8.5.3 State machine variables

8.5.3.1 *active*: A variable that indicates when initialization has completed.

8.5.3.2 *intervalBack, intervalRate, intervalRx, intervalTx*:
Internal variables that represents the intervals between sampled GransSync events.

8.5.3.3 *dPtr*: A pointer this port's associated TimeSyncRxEfdx-entity storage.

8.5.3.4 *ePtr*: A pointer to a data structure that contains port-specific information comprising:
destination_address, *source_address*, *protocolID*, *function*, *version*, *hops*—
Copies of like-named fields from the GrandSync provided ClockSync PDU.
baseTimer—Recently saved time events, each consisting of the following:
index—Index into the *timed*[ ] array, where last times were stored.
range—Number of entries within the *timed*[ ] array
timed[range]—Recently saved time events, each consisting of the following:
grandTime—A previously sampled grandmaster synchronized time.
extraTime—The residual error associated with the sampled *grandTime* value.
localTime—The station-local time affiliated with the sampled *grandTime* value.
frameCount—A consistency-check identifier that is incremented on each transmission.
intervalRx—The expected receive-port interval between successive time-sync transmissions.
intervalTx—The configured transmit-port interval between successive time-sync transmissions.
lastTime—The last transmit time, saved for timeout purposes.
rxSaved—A copy of the last received GrandSync parameters.
thisTxTime—The *localTime* value associated with the last transmission.

8.5.3.5 *localTime*: See 6.3.3.

8.5.3.6 *rxInfo*: Storage for received GrandSync PDUs, comprising:
destination_address, *source_address*, *service_data_unit*
Where *service_data_unit* comprises:
protocolType, *function*, *version*, *hops*, *clockID*,
interval, *flags*, *utcOffset*, *grandTime*, *extraTime*, *localTime*

8.5.3.7 *rsPtr*: A pointer to service-data-unit portion of *rxInfo* storage.

8.5.3.8 *rxPtr*: A pointer to the *rxInfo* storage.

8.5.3.9 *intervalRx*: Represents the sync-interval associated with this station's clock-slave port.

8.5.3.10 *thisTxTime*: A variable that indicates when the last GrandSync frame was transmitted.

8.5.3.11 *times*: A variable pair consisting of the following components:
grandTIme—A time that is referenced to the grandmaster clock.
extraTime—A incremental time that should be added (at the endstation) to the grandmaster clock.

8.5.3.12 *tsPtr*: A pointer to service-data-unit portion of *txInfo* storage.

8.5.3.13 *txInfo*: Storage for to-be-transmitted time-sync PDUs, comprising:
destination_address, source_address, service_data_unit
Where *service_data_unit* comprises:
protocolType, function, version, hops, clockID,
interval, flags, utcOffset, grandTime, extraTime, thisTxTime,
thatRxTime, thatTxTime

8.5.3.14 *txPtr*: A pointer to *txInfo* storage.

8.5.3.15 *intervalTx*: A variable that represents the sync-interval associated with this clock-master port.

8.5.4 State machine routines

8.5.4.1 *Dequeue*(*queue*): See 6.3.4.

8.5.4.2 *Enqueue*(*queue, info*): See 6.3.4.

8.5.4.3 *Enqueue*(*interval*): Expand the compacted interval value to a scaled binary-integer seconds format.

8.5.4.4 *NextSaved*(*btPtr, intervalRate, grandTime, extaTime, thisRxTime*): See 7.5.4.

8.5.4.5 *NextTimed*(*btPtr, localTime, intervalBack*): See 7.5.4.

8.5.4.6 *StationTime*(*entity*): See 7.3.3.

8.5.4.7 *ClockSyncSdu*(*info*): See 6.3.4.

8.5.5 TimeSyncTxEfdx state machine table

The TimeSyncTxEfdx state machine includes a media-dependent timeout, which effectively disconnects a clock-slave port in the absence of received ClockSync frames, as illustrated in Table 8.3.

Table 8.3—TimeSyncTxEfdx state machine table

Current		Row	Next	
state	**condition**		**action**	**state**
START	(localTime–ePtr->lastTime) > T10ms	1	ePtr->lastTime = localTime;	SEND
	(rxInfo=Dequeue(Q_GS_TX)) != NULL	2	—	SINK

	—	3	intervalRx = ePtr->intervalRx;
			intervalTx = ePtr->intervalTx;
			intervalBack = (3 * intervalRx + intervalTx) / 2;
			intervalRate = intervalBack + (3 * intervalTx) / 2;
				START

Table 8.3—TimeSyncTxEfdx state machine table

Current		Row	Next	
state	**condition**		**action**	**state**

SINK	ClockSyncSdu(rsPtr)	4	ePtr->destination_address=rxPtr->destination_address;
			ePtr->source_address = rxPtr->source_address;
			ePtr->protocolID = rsPtr->protocolID;
			ePtr->function = rsPtr->function;
			ePtr->version = rsPtr->version;
			ePtr->clockID = rsPtr->clockID;
			ePtr->hops = rsPtr->hops;
			ePtr->intervalRx = Expand(rsPtr->interval);
			ePtr->flags = rsPtr->flags;
			ePtr->utcOffset = rsPtr->utcOffset;
			NextSaved(btPtr, intervalRate, rsPtr->grandTime,
			rsPtr->extraTime,rsPtr->localTime);
				START

	—	5	Enqueue(Q_ES_TX, rxPtr);	
SEND	sPtr->hops>=MAX_HOPS	6	—	START

	—	7	dPtr = PortPair(ePtr);
			active = (dPtr->phase < RX_PHASE);
			thisTxTime = ePtr->thisTxTime;
			times = NextTimed(btPtr, thisTxTime, intervalBack);
			ePtr->txFrameCount += 1;
			txPtr->destination_address=ePtr->destination_address;
			txPtr->source_address = ePtr->source_address;
			tsPtr->protocolID = ePtr->protocolID;
			tsPtr->function = ePtr->function;
			tsPtr->version = ePtr->version;
			tsPtr->clockID = ePtr->clockID;
			tsPtr->hops = active ? (ePtr->hops+1) : MAX_HOPS;
			tsPtr->flags = ePtr->flags;
			tsPtr->flags.disruption |= !active;
			tsPtr->utcOffset = ePtr->utcOffset;
			tsPtr->frameCount = (ePtr->frameCount%COUNT);
			tsPtr->grandTime = times.grandTime;
			tsPtr->extraTime = times.extraTime;
			tsPtr->thisTxTime = thisTxTime;
			tsPtr->thatTxTime = dPtr->thisTxTime;
			tsPtr->thatRxTime = dPtr->thisRxTime;
			Enqueue(Q_ES_TX, txPtr);	

Row 8.3-1: Transmit periodic ClockSync PDUs.
Row 8.3-2: Receive the GrandSync-echoed ClockSync PDUs.
Row 8.3-3: Update values while waiting for the next change-of-state.

Row 8.3-4: The ClockSync PDUs are checked further.
Row 8.3-5: The non-ClockSync PDUs are passed through.

Row 8.3-6: Overly aged ClockSync frames are discarded.
Row 8.3-7: ClockSync frames are formulated and transmitted.

9. Wireless state machines

EDITOR DVJ NOTE—This clause is has not tracked recent modifications/improvements to 802.1as edited by Kevin Stanton. The reviewer is referred to the most-recent revision of 802.1as, with the cautionary note that the GrandSync service interface considerations are quite different and subject to change as the document content is harmonized.

10. Ethernet passive optical network (EPON) state machines

NOTE—This clause is based on indirect knowledge of the Ethernet-PON (EPON) specifications, as interpreted by the author, and have not been reviewed by the 802.1 or 802.3 WGs. The intent was to provide a forum for evaluation of the GrandSync interfaces, while also triggering discussion of EPON design details. As such, the contents are highly preliminary and subject to change.

10.1 Overview

This clause specifies the state machines that support Ethernet passive optical network (EPON) based bridges. The operations are described in an abstract way and do not imply any particular implementations or any exposed interfaces. There is not necessarily a one-to-one correspondence between the formal specification and the interfaces in any particular implementation.

A (simplified) EPON topology consists of a single optical line terminal (OLT) attached to multiple optical network units (ONUs), as illustrated in Figure 10.1.

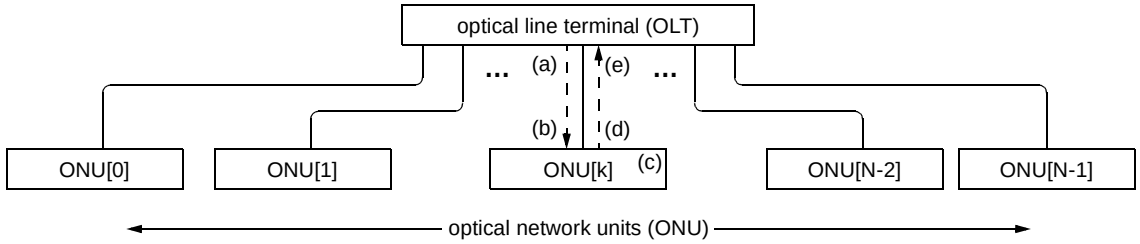

Figure 10.1—EPON topology

Time calibration (see 802.3, 64.2.1.1) involves operations performed at the transmission and reception of an OLT-to-ONU request (illustrated at points (a) and (b)), as well as operations performed at the transmission and reception of ONU-to-OLT response (illustrated at points (d) and (e)).

Both the OLT and the ONU have 32-bit counters that increment every 16 ns. At critical times, the current value of these counters is saved in a timestamp register and/or frame location, as listed below. When completed with each OLU, the central OLT is aware of link delays associated with each of the attached ONUs.

a) OLT request transmit; the frame timestamp is set:
   frame.timestamp = *olt.counter*
b) OLU request receipt; the local counter is set:
   olu.counter = *timestamp*
c) The *olu.counter* continues to increments every 16 ns period.
d) OLU response transmit; frame timestamp is set:
   frame.timeStamp = *olu.counter*
e) OLT response receipt; round-trip and per-link delays are then computed at the OLT, as follows:
   roundTripTime = *olt.counter* – *frame.timeStamp*
   linkDelay = *roundTripTime* / 2

The time-synchronization (TS) client uses this RTT for the link-delay compensation purposes.

10.1.1 Link-dependent indications

The TimeSyncEpon state machines have knowledge of network-local synchronized *ticksTime* timers. With this knowledge, the TimeSyncEpon state machines can operated on frames received from the LLC, as illustrated in Figure 10.2. Link-dependent indications could be required for bridge ports attached to alternative media (not illustrated).

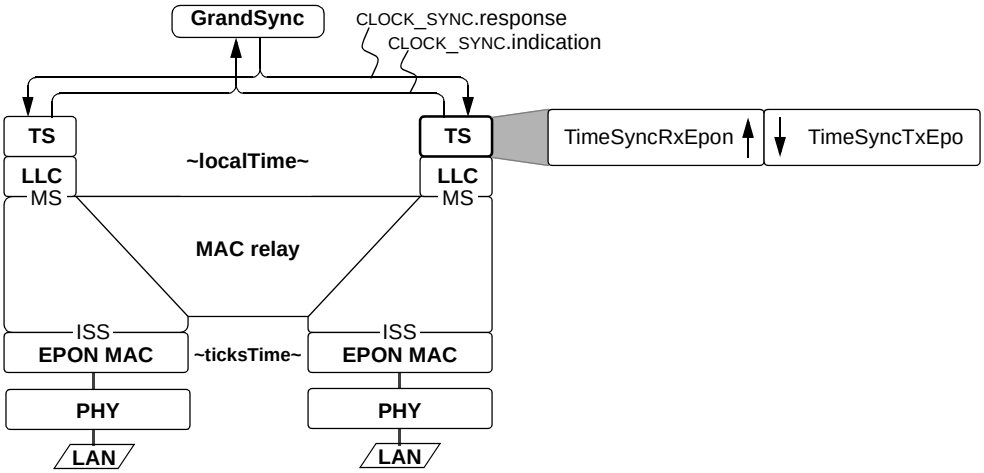

Figure 10.2—EPON interface model

The *ticksTime* values are represented as timers that are incremented once every 16 ns interval, as illustrated on the left side of Figure 10.3. Each synchronized local timer is roughly equivalent to a 6-bit *sec* (seconds) field and a 26-bit *fraction* (fractions of second) field timer, as illustrated on the right side of Figure 10.3.

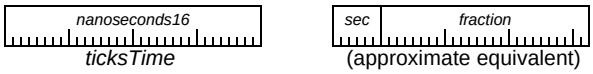

Figure 10.3—Format of EPON-dependent times

The EPON MAC is supplied with frame transmit/receive snapshots, but these are transparent-to and not-used-by the clock-synchronization state machine. Instead, these are used to synchronize the *ticksTime* values in associated MACs and the TimeSyncEpon state machines have access to these synchronized *ticksTime* values.

10.2 timeSyncEpon frame format

The timeSyncEpon frames facilitate the synchronization of neighboring clock-master and clock-slave stations. The frame, which is normally sent at 10 ms intervals, includes time-snapshot information and the identity of the network's clock master, as illustrated in Figure 10.4. The gray boxes represent physical layer encapsulation fields that are common across Ethernet frames.

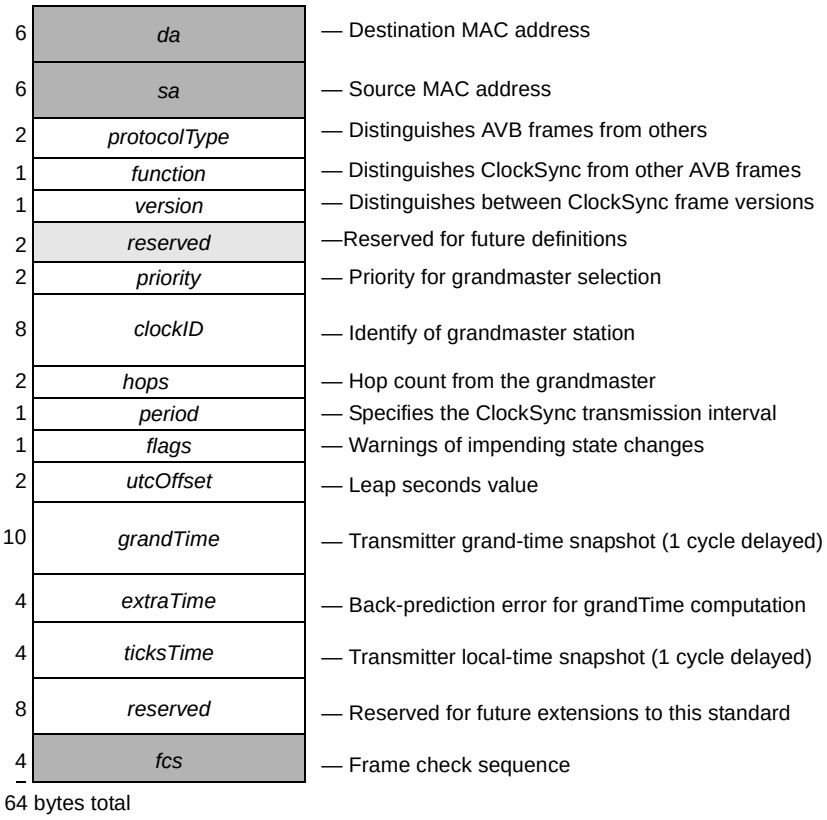

Figure 10.4—timeSyncEpon frame format

The 48-bit ***da*** (destination address), 48-bit ***sa*** (source address) field, 16-bit ***protocolType,*** 8-bit ***function***, 8-bit ***version***, 2-byte ***priority***, 8-byte ***clockID***, 2-byte ***hops***, 1-byte ***priority***, 1-byte ***flags***, 2-byte ***utcOffset***, 80-bit ***grandTime***, and 32-bit ***extraTime*** field are specified in 6.2.1.2.

10.2.1 *ticksTime*: A value representing local time in units of a 16 ns timer ticks, as illustrated in Figure 10.5.

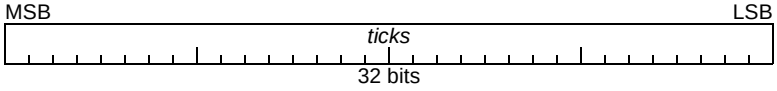

Figure 10.5—tickTime format

10.3 TimeSyncRxEpon service interface primitives

10.3.1 ES_UNITDATA.indication

10.3.1.1 Function

Provides the TimeSyncRxEpon entity with clock-synchronization parameters derived from arriving time-sync frames.

10.3.1.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
ES_UNITDATA.indication {
    destination_address,      // Destination address
    source_address,           // Optional
    priority,                 // Forwarding priority
    service_data_unit,        // Delivered content
    {                         // Contents of the service_data_unit
        protocolType,         // Distinguishes AVB frames from others
        function,             // Distinguishes between ClockSync and other AVB frames
        version,              // Distinguishes between ClockSync frame versions
        priority,             // Precedence for grandmaster selection
        clockID,              // Precedence for grandmaster selection
        hops,                 // Distance from the grandmaster station
        interval,             // Nominal ClockSync transmission interval
        flags,                // Control flags
        utcOffset,            // Difference between UTC and TAI timescales
        grandTime,            // Global-time snapshot (1-cycle delayed)
        extraTime,            // Accumulated grandTime error
        ticksTime             // Local-time snapshot (1-cycle delayed)
    }
}
```

The parameters of the ES_UNITDATA.indication are described as follows:

The 48-bit **destination_address**, 48-bit **source_address**, and 8-bit **priority** field are specified in 6.2.1.2.

The service_data_unit consists of subfields; for content exchanged with the GrandTime protocol entity, these fields include the following.

The 16-bit ***protocolType***, 8-bit ***function***, 8-bit ***version***, 2-byte ***priority***, 8-byte ***clockID***, 2-byte ***hops***, 1-byte ***interval***, 1-byte ***flags***, 2-byte ***utcOffset***, 10-byte ***grandTime***, and 4-byte ***extraTime*** fields are specified in 6.2.1.2.

10.3.1.2.1 *ticksTime*: A 32-bit field that specifies the local free-running time within this subnet, when the previous ClockSync frame was received (see 10.2.1).

10.3.1.3 When generated

The service primitive is generated upon the receipt of a time-sync related frame delivered from the MAC. The intent is to facilitate reformatting and snapshot-time adjustment before the content of that frame is delivered to the ClockMaster and TS entities.

10.3.1.4 Effect of receipt

The service primitive invokes processing of time-sync related content and forwarding of unrelated content. For time-sync related content, the processing included reformatting and compensation for receive-link transmission delays.

10.3.2 ES_UNITDATA.response

10.3.2.1 Function

Provides the TimeSyncRxEpon entity with clock-synchronization parameters derived from arriving time-sync frames.

10.3.2.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
ES_UNITDATA.indication {
    destination_address,      // Destination address
    source_address,           // Optional
    priority,                 // Forwarding priority
    service_data_unit,        // Delivered content
    {                         // Contents of the service_data_unit
        protocolType,         // Distinguishes AVB frames from others
        function,             // Distinguishes between ClockSync and other AVB frames
        version,              // Distinguishes between ClockSync frame versions
        priority,             // Precedence for grandmaster selection
        clockID,              // Precedence for grandmaster selection
        hops,                 // Distance from the grandmaster station
        interval,             // Nominal ClockSync transmission interval
        flags,                // Control flags
        utcOffset,            // Difference between UTC and TAI timescales
        grandTime,            // Global-time snapshot (1-cycle delayed)
        extraTime,            // Accumulated grandTime error
        ticksTime             // Local-time snapshot (1-cycle delayed)
    }
}
```

The parameter definitions for the ES_UNITDATA.response and the ES_UNITDATA.indication are the same (see 10.3.1.2).

10.3.2.3 When generated

The service primitive is generated upon the receipt of a time-sync related frame delivered from the MAC. The intent is to facilitate reformatting and snapshot-time adjustment before the content of that frame is delivered to the ClockMaster and TS entities.

10.3.2.4 Effect of receipt

The service primitive invokes processing of time-sync related content and forwarding of unrelated content. For time-sync related content, the processing included reformatting and compensation for receive-link transmission delays.

10.4 TimeSyncRxEpon state machine

10.4.1 Function

The TimeSyncRxEpon state machine is responsible for receiving MAC-supplied ClockSyncRxEpon PDUs, converting their media-dependent parameters, and sending normalized GrandSync PDUs. The sequencing of this state machine is specified by Table 10.1; details of the computations are specified by the C-code of Annex G.

10.4.2 State machine definitions

10.4.2.1 MAX_HOPS: See 6.3.2.2.

10.4.2.2 NULL: A value that (by design) cannot be confused with a valid value.

10.4.2.3 Q_GS_RX: The queue identifier associated with PDUs sent to the GrandSync.

10.4.2.4 Q_ES_TX: The queue identifier associated with frames received by the MAC.

10.4.3 State machine variables

10.4.3.1 *ePtr*: A pointer to a entity-specific data structure comprising:
interval—The expected interval between time-sync frame transmissions.

10.4.3.2 *backTime*: Represents the time lapse between transmission of reception of the ClockSync frame.

10.4.3.3 *rsPtr*: A pointer to the service-data-unit portion of the *rxInfo* storage.

10.4.3.4 *rxInfo*: A storage location for received service-interface parameters, comprising:
destination_address, *source_address*, *service_data_unit*
Where *service_data_unit* comprises:
extraTime, *function*, *grandTime*, *hops*,
precedence, *protocolType*, *ticksTime*, *version*

10.4.3.5 *rxPtr*: A pointer to the *rxInfo* storage location.

10.4.3.6 *tsPtr*: A pointer to the service-data-unit portion of the *txInfo* storage.

10.4.3.7 *txInfo*: A storage location for to-be-transmitted CLOCK_SYNC.indication parameters, comprising:
destination_address, *source_address*, *service_data_unit*
Where *service_data_unit* comprises:
extraTime, *function*, *grandTime*, *hops*, *precedence*,
protocolType, *localTime*, *ticksTime*, *version*

10.4.3.8 *txPtr*: A pointer to the *txInfo* storage location.

10.4.4 State machine routines

10.4.4.1 *Dequeue*(*queue*): See 6.3.4.

10.4.4.2 *Enqueue*(*queue, info*): See 6.3.4.

10.4.4.3 *StationTimes*(*entityPointer*): Returns the station's generic *localTime* and media-dependent subnet-synchronized *tickTime* values.

10.4.4.4 *TicksToTime*(*ticks*): Returns the *localTime* duration corresponding to the argument time duration.

10.4.4.5 *ClockSyncSdu*(*info*): See 6.3.4.

10.4.5 TimeSyncRxEpon state machine table

The TimeSyncRxEpon state machine processes arriving media-dependent ClockSyncRxEpon PDUs and sends standard ClockSync PDUs to the GrandSync entity, as illustrated in Table 10.1.

Table 10.1—TimeSyncRxEpon state machine table

Current		Row	Next	
state	**condition**		**action**	**state**
START	(rxInfo = Dequeue(Q_RX_MAC))!=NULL	1	—	TEST
	—	2	times = StationTime(ePtr);	START
TEST	!ClockSyncSdu(rsPtr)	3	Enqueue(Q_ES_RX, rxInfo);	START
	rsPtr->hops >= MAX_HOPS	4	—	START
	ePtr->type == OLT	5	baseTime = –ePtr->roundTripDelay/2;	SEND
	—	6	baseTime = 0;	

SEND	—	7	diffTime = times.ticksTime – rsPtr->ticksTime;
			localTime = times.localTime –
			TicksToTime(baseTime + diffTime);
			txPtr->destination_address =
			rxPtr->destination_address;
			txPtr->source_address = rxPtr->source_address;
			tsPtr->protocolType = rsPtr->protocolType;
			tsPtr->function = rsPtr->function;
			tsPtr->version = rsPtr->version;
			tsPtr->priority = rsPtr->priority;
			tsPtr->clockID = rsPtr->clockID;
			tsPtr->hops = rsPtr->hops;
			tsPtr->interval = rsPtr->interval;
			tsPtr->flags = rsPtr->flags;
			tsPtr->utcOffset = rsPtr->utcOffset;
			tsPtr->grandTime = rsPtr->grandTime;
			tsPtr->extraTime = rsPtr->extraTime;
			tsPtr->localTime = localTime;
			Enqueue(Q_GS_RX, txInfo);
				START

Row 10.1-1: Initiate inspection of frames received from the lower-level MAC.
Row 10.1-2: Wait for the next frame to arrive.

Row 10.1-3: The non-ClockSync frames are passed through.
Row 10.1-4: Overly aged ClockSync frames are discarded.
Row 10.1-5: Received OLT frames are compensated for the link-delay.
Row 10.1-6: Received ONU frames are not link-delay compensated (this is done at the OLT).

Row 10.1-7: Active ClockSync frames are adjusted for transfer delays and passed through.

10.5 TimeSyncTxEpon state machine

10.6 TimeSyncTxEpon service interface primitives

10.6.1 ES_UNITDATA.request

10.6.1.1 Function

Provides the EPON entity with clock-synchronization parameters for constructing departing time-sync frames.

10.6.1.2 Semantics of the service primitive

The semantics of the primitives are as follows:

```
ES_UNITDATA.request
{
    destination_address,    // Destination address
    source_address,         // Optional
    priority,               // Forwarding priority
    service_data_unit,      // Delivered content
    {                       // Contents of the service_data_unit
        protocolType,       // Distinguishes AVB frames from others
        function,           // Distinguishes between ClockSync and other frames
        version,            // Distinguishes between ClockSync frame versions
        priority,           // Bias for grandmaster precedence
        clockID,            // Tie-breaker for grandmaster precedence
        hops,               // Distance from the grandmaster station
        grandTime,          // Global-time snapshot (1-cycle delayed)
        extraTime,          // Accumulated grandTime error
        ticksTime           // Local-time snapshot
    }
}
```

The parameters of the MA_UNITDATA.request are described in 10.3.1.2.

10.6.1.3 When generated

The service primitive is generated at a periodic rate, for the purposes of synchronizing the *grandTime* values resident in other stations.

10.6.1.4 Effect of receipt

The service primitive triggers the transmission of a ClockSync frame on the affiliated port.

10.7 ClockSlave state machine

10.7.1 Function

The TimeSyncTxEpon state machine is responsible for modifying time-sync CLOCK_SYNC.response parameters to form ClockSync frames for transmission over the attached link.

10.7.2 State machine definitions

10.7.2.1 MAX_HOPS: See 6.3.2.2.

10.7.2.2 NULL: A value that (by design) cannot be confused with a valid value.

10.7.2.3 Q_ES_RX: The queue identifier associated with PDUs sent for transmission by the MAC.

10.7.2.4 Q_GS_TX: The queue identifier associated with PDUs sent for transmission from the GrandSync.

10.7.2.5 T10ms: A constant the represents a 10 ms value.

10.7.3 State machine variables

10.7.3.1 *intervalBack*: Represents the back-interpolation interval for transmit-time affiliations.

10.7.3.2 *ePtr*: A pointer to a entity-specific data structure comprising:
baseTimer—Recently saved time events, each consisting of the following:
index—Index into the *timed*[ ] array, where last times were stored.
range—Number of entries within the *timed*[ ] array
timed[range]—Recently saved time events, each consisting of the following:
grandTime—A previously sampled grandmaster synchronized time.
extraTime—The residual error associated with the sampled *grandTime* value.
localTime—The station-local time affiliated with the sampled *grandTime* value.
lastTime—The last PDU-transmit time; used to space periodic transmissions.
rxSaved—A copy of the last received GrandSync parameters.
interval—The expected interval between time-sync frame transmissions.

10.7.3.3 *rsPtr*: A pointer to the service-data-unit portion of *rxInfo* storage.

10.7.3.4 *rxInfo*: Storage for the contents of GrandSync PDUs, comprising:
destination_address, *source_address*, *service_data_unit*
Where *service_data_unit* comprises:
protocolType, *function*, *version*, *priority*, *clockID*, *hops*,
interval, *flags*, *utcOffset*, *grandTime*, *extraTime*

10.7.3.5 *rxPtr*: A pointer to the *rxInfo* storage.

10.7.3.6 *intervalRx*: Represents the sync-interval associated with this station's clock-slave port.

10.7.3.7 *localTime*: See 6.3.3.

10.7.3.8 *ssPtr*: A pointer to the service-data-unit portion of the *ePtr->rxSaved* storage

10.7.3.9 *sxPtr*: A pointer to the *ePtr->rxSaved* storage.

10.7.3.10 *tsPtr*: A pointer to the service-data-unit portion of *txInfo* storage.

10.7.3.11 *txInfo*: Storage for a to-be-transmitted MAC frame, comprising:
destination_address, *source_address*, *service_data_unit*
Where *service_data_unit* comprises:

extraTime, *function*, *grandTime*, *hops*,
protocolType, *precedence*, *ticksTime*, *version*

10.7.3.12 *txPtr*: A pointer to the *txInfo* storage.

10.7.3.13 *ticksTime*: A 32-bit shared EPON media-dependent time value; incremented every 16 ns.

10.7.4 State machine routines

10.7.4.1 *Dequeue*(*queue*): See 6.3.4.

10.7.4.2 *Enqueue*(*queue, info*): See 6.3.4.

10.7.4.3 *NextTimed*(*btPtr, localTime, intervalBack*): See 7.5.4.

10.7.4.4 *StationTime*(*entity*): See 6.3.4.

10.7.4.5 *TicksTime*(*entity*): See 10.4.4.

10.7.4.6 *ClockSyncSdu*(*info*): See 6.3.4.

10.7.5 TimeSyncTxEpon state machine table

The TimeSyncTxEpon state machine includes a media-dependent timeout, which effectively disconnects a clock-slave port in the absence of received timeSyncEpon frames, as illustrated in Table 10.2.

Table 10.2—TimeSyncTxEpon state machine table

Current		Row	Next	
state	**condition**		**action**	**state**
START	(rxInfo = Dequeue(Q_GS_TX)) != NULL	1	—	TEST
	(localTime – ePtr->lastTime) > T10ms	2	ePtr->lastTime = localTime;	SEND
	—	3	times = StationTime(ePtr);	START
TEST	!ClockSyncSdu(rsPtr)	4	Enqueue(Q_ES_TX, rxPtr);	START
	ssPtr->hops >= MAX_HOPS	5	—	
	ePtr->type == OLT	6	baseTime = –ePtr->roundTripDelay/2;	SEND
	—	7	baseTime = 0;	

SEND	—	8	intervalRx = ssPtr->interval;
			intervalTx = ePtr->interval;
			intervalBack =
			(3 * intervalRx + intervalTx) / 2;
			localTime=
			times.localTime + TicksToTime(baseTime);
			nextTimes =
			NextTimed(btPtr, localTime, intervalBack);
			txPtr->destination_address =
			sxPtr->destination_address;
			txPtr->source_address = sxPtr->source_address;
			tsPtr->protocolType = ssPtr->protocolType;
			tsPtr->function = ssPtr->function;
			tsPtr->version = ssPtr->version;
			tsPtr->precedence = ssPtr->precedence;
			tsPtr->hops = ssPtr->hops;
			tsPtr->grandTime = nextTimes.grandTime;
			tsPtr->extraTime = nextTimes.extraTime;
			tsPtr->ticksTime = ticksTime;
			Enqueue(Q_ES_TX, txPtr);
				START

Row 10.2-1: Accepted PDUs are further checked before being processed.
Row 10.2-2: Transmit periodic ClockSync frames.
Row 10.2-3: Wait for the next change-of-state.

Row 10.2-4: Non-ClockSync PDUs are retransmitted in the standard fashion.
Row 10.2-5: Overly aged ClockSync PDUs are not transmitted.
Row 10.2-6: Transmitted OLT frames are compensated for the link-delay.
Row 10.2-7: Transmitted ONU frames are not link-delay compensated (this is done at the OLT).

Row 10.2-8: Format and transmit the media-specific ClockSync frame.

Annexes

Annex A

(informative)

Bibliography

[B1] IEEE 100, The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition.[1]

[B2] IEEE Std 802-2002, IEEE Standards for Local and Metropolitan Area Networks: Overview and Architecture.

[B3] IEEE Std 801-2001, IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture.

[B4] IEEE Std 802.1D-2004, IEEE Standard for Local and Metropolitan Area Networks: Media Access Control (MAC) Bridges.

[B5] IEEE Std 1394-1995, High performance serial bus.

[B6] IEEE Std 1588-2002, IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems.

[B7] IETF RFC 1305: Network Time Protocol (Version 3) Specification, Implementation and Analysis, David L. Mills, March 1992[2]

[B8] IETF RFC 2030: Simple Network Time Protocol (SNTP) Version 4 for IPv4, IPv6 and OSI, D. Mills, October 1996.

[1]IEEE publications are available from the Institute of Electrical and Electronics Engineers, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA (http://standards.ieee.org/).

[2]IETF publications are available via the World Wide Web at http://www.ietf.org.

Annex B

(informative)

Time-scale conversions

B.1 Overview

For historical reasons, time is specified in a variety of ways as listed in Table B.1. GPS, PTP, and TAI times are based on values yielded by atomic clocks and advance on each second. NTP and UTC times are similar, but are occasionally adjusted by one leap-second, to account for differences between the atomic clocks and the rotation time of the earth.

Table B.1—Time-scale parameters

	Time scale				
Parameter	GPS	PTP	TAI	NTP	UTC
approximate epoch	1980-01-06 1999-08-22	1970-01-01	1972-01-01*	1900-01-01	1972-01-01*
representation	weeks.seconds	seconds	YYYY-MM-DD hh:mm:ss	seconds	YYYY-MM-DD hh:mm:ss
rollover (years)	19.7	8,925,513	10,000	136.19	10,000
leapSeconds	no			yes	

Notes:

* The TAI time when TAI and UTC were first specified to deviate by only integer seconds. (There is no true epoch for the TAI and UTC time scales.)

GPS global positioning satellite

NTP Network Time Protocol

PTP Precision Time Protocol (commonly used in POSIX)

TAI International Atomic Time (from the French term *Temps Atomique International*)

UTC Coordinated Universal Time (a compromise between the English and French):
English speakers wanted the initials of their language: CUT for "coordinated universal time"
French speakers wanted the initials of their language: TUC for "temps universel coordonné".

B.2 TAI and UTC

TAI and UTC are international standards for time based on the SI second; both are expressed in days, hours, minutes and seconds. TAI is implemented by a suite of atomic clocks and forms the timekeeping basis for other time scales in common use. The rate at which UTC time advances is normally identical to the rate of TAI. An exception is an occasion when UTC is modified by adding or subtracting leap seconds.

Prior to 1972-01-01, corrections to the offset between UTC and TAI were made in fractions of a second. After 1972-01-01, leap-second corrections are applied to UTC preferably following second 23:59:59 of the last day of June or December. As of 2006-01-01, TAI and UTC times differed by +33 seconds.

In POSIX based computer systems, the common time conversion algorithms can produce the correct ISO 8601-2004 printed representation format “YYYY-MM-DD hh:mm:ss” for both TAI and UTC.

The PTP epoch is set such that a direct application of the POSIX algorithm to a PTP time-scale timestamp yields the ISO 8601-2004 printed representation of TAI. Subtracting the current *leapSeconds* value from a PTP timestamp prior to applying the POSIX algorithm yields the ISO 8601-2004 printed representation of UTC. Conversely, applying the inverse POSIX algorithm and adding *leapSeconds* converts from the ISO 8601-2004 printed form of UTC to the form convenient for generating a PTP timestamp.

Example: The POSIX algorithm applied to a PTP timestamp value of 8 seconds yields 1970-01-01 00:00:08 (eight seconds after midnight on 1970-01-01 TAI). At this time the value of *leapSeconds* was approximately 8 seconds. Subtracting this 8 seconds from this time yields 1970-01-01 00:00:00 UTC.

Example: The POSIX algorithm applied to a PTP timestamp value of 0 seconds yields 1970-01-01 00:00:00 TAI. At this time the value of *leapSeconds* was approximately 8 seconds. Subtracting this 8 seconds from this time yields 1969-12-31 23:59:52 UTC.

B.3 NTP and GPS

Two standard time sources of particular interest in implementing PTP systems: NTP and GPS. Both NTP and GPS systems are expected to provide time references for calibration of the grandmaster supplied PTP time.

NTP represents seconds as a 32 bit unsigned integer that rolls-over every 2^{32} seconds $\approx$ 136 years, with the first such rollover occurring in the year 2036. The precision of NTP systems is usually in the millisecond range.

NTP is a widely used protocol for synchronizing computer systems. NTP is based on sets of servers, to which NTP clients synchronize. These servers themselves are synchronized to time servers that are traceable to international standards.

NTP provides the current time. In NTP version 4, the current *leapSeconds* value and warning flags marking indicating when a *leapSecond* will be inserted at the end of the current UTC day. The NTP clock effectively stops for one second when the leap second is inserted.

GPS time comes from a global positioning satellite system, GPS, maintained by the U.S. Department of Defense. The precision of GPS system is usually in the 10-100 ns range. GPS system transmissions represent the time as {*weeks*, *secondsInWeek*}, the number of weeks since the GPS epoch and the number of seconds since the beginning of the current week.

GPS also provides the current *leapSeconds* value, and warning flags marking the introduction of a leap second correction. UTC and TAI times can be computed solely based the information contained in the GPS transmissions.

GPS timing receivers generally manage the epoch transitions (1024-week rollovers), providing the correct time (YYYY-MM-DD hh:mm:ss) in TAI and/or UTC time scales, and often also local time; in addition to providing the raw GPS week, second of week, and leap second information.

B.4 Time-scale conversions

Previously discussed representations of time can be readily converted to/from PTP *time* based on a constant offsets and the distributed *leapSeconds* value, as specified in Table B.2. Within Table B.2, all variables represent integers; '/' and '%' represent a integer divide and remainder operation, respectively.

Table B.2—Time-scale conversions

ta		**PTP value *tb*:**
name	format	
GPS	weeks:seconds	tb = ta.seconds + 315 964 819 + (gpsRollovers * 1024 + ta.weeks) * (7 * DAYSECS);
		ta.weeks = (tb – 315 964 819) / (7 * DAYSECS); ta.days = (tb – 315 964 819) % (7 * DAYSECS);
TAI	date{YYYY,MM,DD}:time{hh,mm,ss}	tb = DateToDays("1970-01-01", ta.date) * DAYSECS + ((ta.time.hh * 24) + ta.time.mm) *60) + ta.time.ss;

		secs = tb % DAYSECS;
		ta.date = DaysToDate("1970-01-01", tb / DAYSECS);
		ta.time.hh = secs / 3600;
		ta.time.mm = (secs % 3600)/60;
		ta.time.ss = (secs % 60);

NTP	seconds	tb = (ta + leapSeconds) – 2208988800;
		ta = (ta – leapSeconds) + 2208988800;
UTC	date{YYYY,MM,DD}:time{hh,mm,ss}	tb = DateToDays("1970-01-01", ta.date) * DAYSECS + ((ta.time.hh * 24) + ta.time.mm) *60) + ta.time.ss + leapSeconds;

		tc = tb – leapSeconds;
		secs = tc % DAYSECS;
		ta.date = DaysToDate("1970-01-01", tc/DAYSECS);
		ta.time.hh = secs / 3600;
		ta.time.mm = (secs % 3600)/60;
		ta.time.ss = (secs % 60);

Note:

gpsRollovers Currently equals 1; changed from 0 to 1 between 1999-08-15 and 1999-08-22.
DAYSECS The number of seconds within a day: (60*60*24).
leapSeconds Extra seconds to account for variations in the earth-rotation times: 33 on 2006-01-01.
DateToDays For arguments DateToDays(*past*, *present*), returns days between *past* and *present* dates.
DaysToDate For arguments DaysToDate(*past*, *days*), returns the current date, *days* after the *past* date.

B.5 Time zones and GMT

The term Greenwich Mean Time (GMT) once referred to mean solar time at the Royal Observatory in Greenwich, England. GMT now commonly refers to the time scale UTC; or the UK winter time zone (Western European Time, WET). Such GMT references are strictly speaking incorrect; but nevertheless quite common. The following representations correspond to the same instant of time:

18:07:00 (GMT), commonplace usage	13:07:00 (Eastern Standard Time, EST)
18:07:00 (UTC)	1:07 PM (Eastern Standard Time, EST)
18:07:00 (Western European Time, WET)	10:07:00 (Pacific Standard Time, PST)
6:07 PM (Western European Time, WET)	10:07 AM (Pacific Standard Time, PST)

Annex C

(informative)

Reclocked ClockSync requirements

C.1 Cascaded clock considerations

C.1.1 Cascading causes sync-interval bunching

The naive approach towards forwarding time-synchronization information is to quickly propagate time-reference snapshots through successive stations. Unfortunately, relatively small (¼ interval) residence-time delays per station can cause significant bunching, as illustrated in Figure C.1.

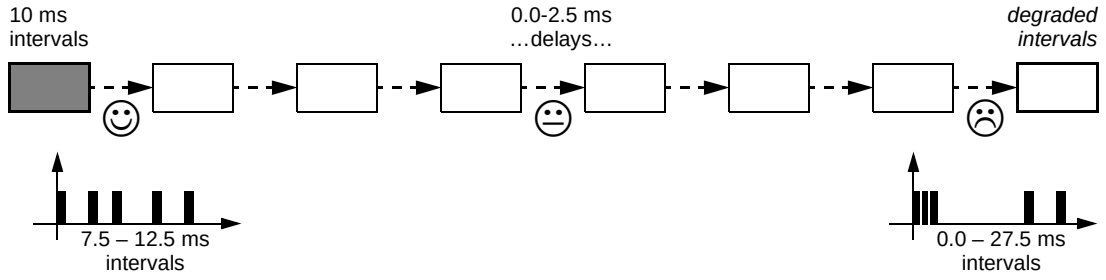

Figure C.1—Cascading causes sync-interval bunching

Techniques for avoiding such bunching are well known and practiced in the form of reclocked synchronous circuits. For example, Ethernet stations accept (baud-rate) information at a closely matched input clock rate, reclock the data with a local reference, and regenerate information without degraded jitter performance.

C.1.2 Reclocking eliminates sync-interval bunching

Applying these techniques to clock-sync transmission is straightforward. Rather than quickly forwarding these frames, their information is saved. That saved information is then forwarded in the same periodic fashion, based on local-station timing, as illustrated in Figure C.2. While such reclocked systems more susceptible to gain-peaking/whiplash effects, inherent design and verification simplicities favor their use.

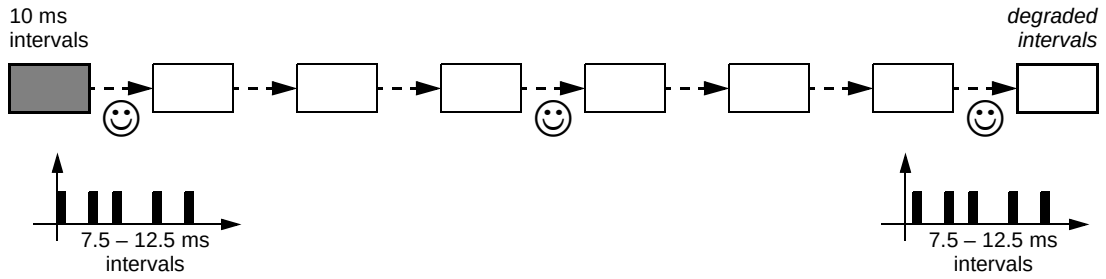

Figure C.2—Reclocking eliminates sync-interval bunching

C.1.3 Reclocking localizes sync-interval properties

The reclocked sync-interval strategy is compatible with bridged mixed-media systems. The persistent or transient sync-interval rate of an intermediate (perhaps longer or more power sensitive) link could be less than the rate assumed for the clock-master, as illustrated in the center of Figure 3.3. Similarly, wireless links could base their timing events on triggers initiated by the clock-slave station, as illustrated in the right side of Figure 3.3.

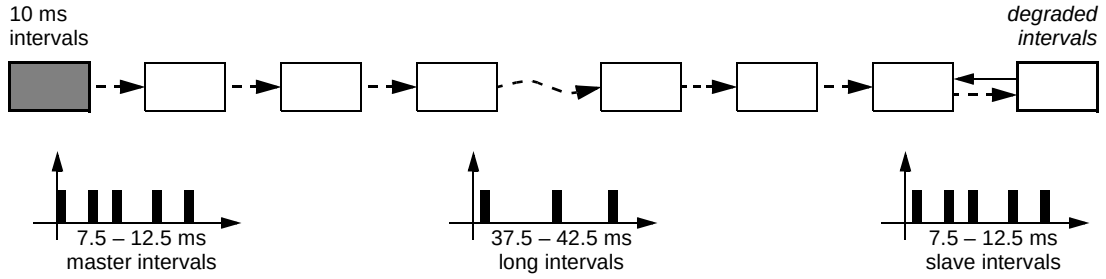

Figure 3.3—Reclocking localizes sync-interval properties

Other flow-through clocking designs would require special “boundary clock” architectures to support such mixed systems. With the interval reclocking strategy, the additional (specification and implementation) complexities of such boundary-clock architectures are easily avoided.

C.2 Sampling offset/rate conversion

Each clock-master port is responsible for using its received {*grandTime*,*extraTime*,*rcTime*} affiliations to derive distinct {*grandTime1*,*extraTime1*,*txTime*} affiliations that are transmitted to its neighbor. Since the values of *rcTime* and *txTime* are (by convention) coupled to the receive and transmit times, this update involves generation of {*grandTime*,*extraTime*,*rcTime*} triads by resampling within the array of previously saved {*grandTime*,*extraTime*,*rcTime*} triads.

C.2.1 Forward extrapolation inaccuracies

A typical design approach (and that used by IEEE Std 1588) views the received {*grandTime*,*rcTime*} affiliations as points on a curve, sampled at received-snapshot times *rc*[*n*]. The objective is to generate the distinct set of {*grandTime1*,*txTime*} affiliations by extrapolating from a distinct set of receive-snapshot times *rc*[*n*], as illustrated in Figure 3.4.

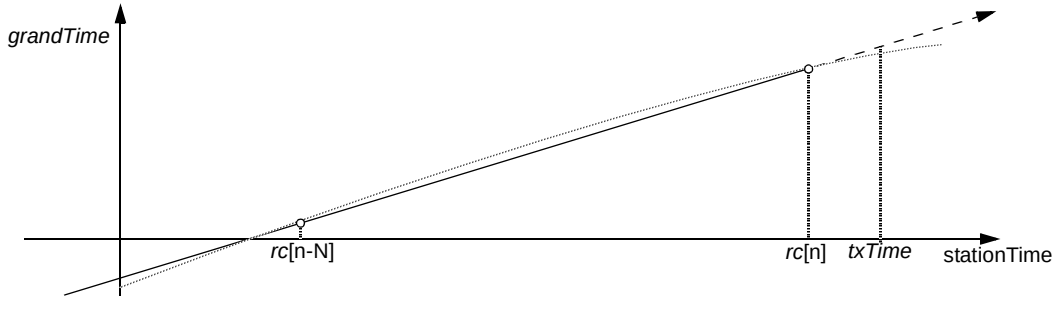

Figure 3.4—Extrapolation for *grandTime*

Extrapolation techniques exhibit gain peaking at frequencies whose wavelength is twice the {*rc*[*n*-N],*rc*[*n*]} slope-averaging interval, because the extrapolated value can exceed what would have been the sampled time value. A cascade of multiple stations emphasizes the gain-peaking inaccuracies, allowing errors to accumulate in an $O(N^2)$ fashion.

C.2.2 Forward interpolation inaccuracies

To reduce gain-peaking effects, the resampling computation can be migrated to a safe-interpolation domain. This involves subtracting a *backTime* constant from *txTime*, yielding a new time *tbTime*, for which a less gain-peaking sensitive interpolation is viable, as illustrated in Figure C.5. In concept, the stale (but not incorrect) {*grandTime*,*rcTime*} affiliations could be passed to the terminal clock-slave stations, wherein a single extrapolation-to-the-future accumulation could be performed. A preferred technique is to compensate the interpolation result on an per-station basis as the time-reference flows towards the clock-slave station, as discussed in the following subclauses.

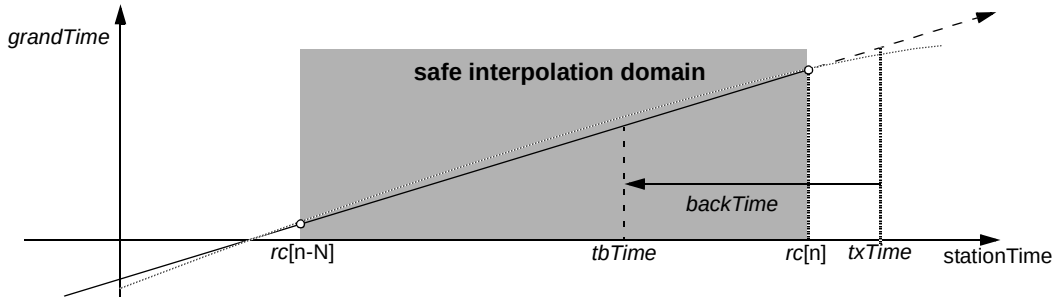

Figure C.5—Extrapolation for *grandTime*

C.2.3 Backward *grandTime* interpolation

A more-scalable backward-interpolation approach also views the received {*grandTime*,*rcTime*} affiliations as points on a curve. The objective is to generate the distinct set of {*grandTime1*,*txTime*} affiliations by interpolating within a distinct set of {*grandTime, rcTime*} points on the curve, as illustrated in Figure C.6.

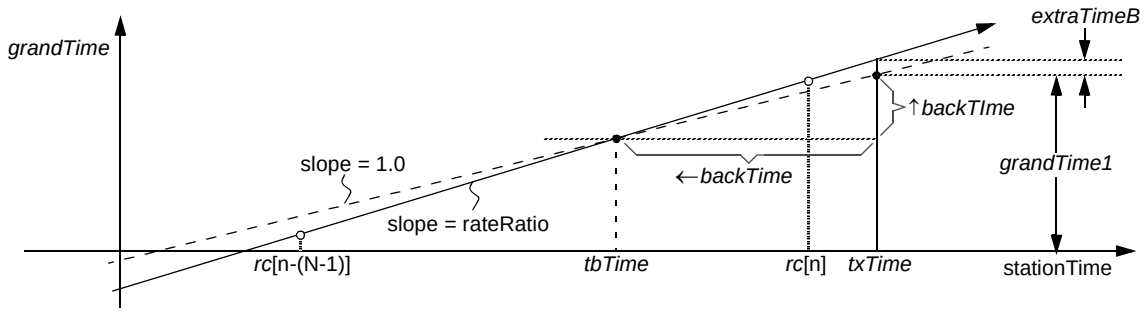

Figure C.6—Interpolation for *grandTimeA*

$$rateRatio = (grandTime[n] - grandTime[n\text{-N}]) / (rc[n] - rc[n\text{-N}]) \quad (3.1)$$

$$grandTime1[m] = grandTime[n] + rateRatio * ((txTime - backTime) - rc[n]) + backTime; \quad (3.2)$$

backTime is a constant (sync-interval dependent) value.

$$extraTime1[m] = (rateRatio - \text{ONE}) * backTime; \quad (3.3)$$

The advantage of this technique is the separation of *grandTime*[*m*] and *extra*[*m*] components. The interpolation process eliminates gain-peaking for the *grandTime*[*m*] value, thus reducing error effects when passing through multiple bridges. The sideband *extraTime* signal remains significant, and is therefore carried through bridges, so that the cumulative *grandTimed*[*m*]+*extraTime*[*m*] value can be passed to the end-point application.

From an intuitive perspective, the whiplash-free nature of the back-in-time interpolation is attributed to the use of interpolation (as opposed to extrapolation) protocols. Interpolation between input values never produces a larger output value, as would be implied by a gain-peaking (larger-than-unity gain) algorithm. A disadvantage of back-in-time interpolation is the requirement for a side-band *extraTime* communication channel, over which the difference between nominal and rate-normalized *backTime* values can be transmitted.

C.2.4 Backward *extraTime* Averaging

An averaging (rather than backward-interpolation) approach is applied to the received {*extraTime*, *rcTime*} affiliations as points on a curve, sampled at received-snapshot times *rc*[*n*]. The {*extraTime*,*tx*[*m*]} affiliations are produced by averaging recently observed *extraTime* values, as illustrated in Figure C.7.

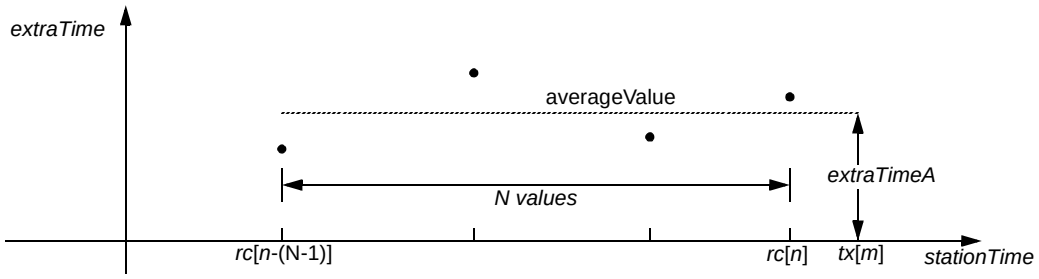

Figure C.7—Interpolation of *extraTimeD*

$$extraTimeA[m] = (extraTime[n–(N–1)] + \ldots extraTime[n]) / N \quad (3.4)$$
$$extraTime1[m] = extraTimeA[m]+ extraTimeB[m]; \quad (3.5)$$

The to-be-transmitted value of *extraTime1*[*m*] consists of a contribution *extraTimeA* (accumulated from previous stations's *grandTime* interpolations) and a contribution *errorTimeB* (coming from this station's *grandTime* interpolation). Note that the averaging of *extraA* values is effectively a low-pass filtering process that removes noise without causing a gain-peaking frequency response.

NOTE—For simplicity and scalability, the computed *extraTime1* time is based on *N*, a fixed number of samples, where *N* is a convenient power-of-two in size.

Annex D

(informative)

Simulation results (preliminary)

D.1 Simulation environment

This annex describes several simulations performed with the intent of comparing time-extrapolation and time-interpolation algorithms. To reduce possibilities of code-conversion errors, the simulation model executes the C code of Annex G. Simulation time is based on a 128-bit *systemTime*, represented by 64-bit seconds and fractions-of-second components, to ensure that precision and range are not constraining factors.

The simulation consists of *bridgeCount* identical super-bridge components, as illustrated in Figure D.1. For generality and uniformity, each bridge includes ClockMaster and ClockSlave entities. The smallest MAC address is assigned to the left-most station; for other stations, the address is incremented for each sequential right-side bridge. The simulations assumed *bridgeCount* values of 8 (the assumed AVB diameter) and 64 (a reasonable IEEE 802.17 ring diameter).

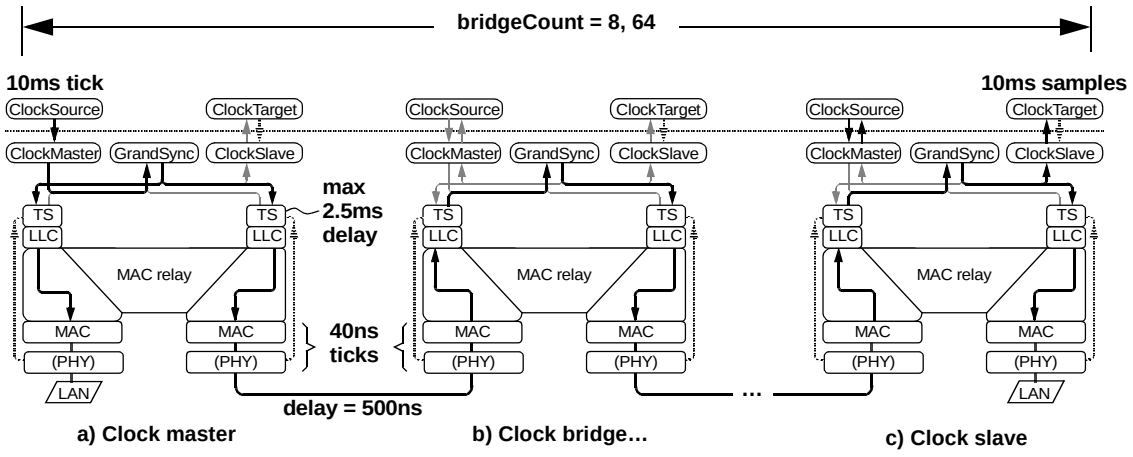

Figure D.1—Time-synchronization flows

The transmit portion of the TS component (emulated by the DuplexTxExec routine) introduces a random delay of no more than 2.5 ms, thus emulating delays consistent with the 10 ms sync-frame transmission rate. A 20 ns sampling clock ambiguity (corresponding to 25 MHz) is incorporated into the MAC component (emulated by the DupMacTxExec routine).

The cable is modeled as a symmetric 500ns delay, corresponding to a cable length of approximately 100 meters.

Station clock accuracies are assigned randomly/uniformly within the range of the allowed ±100 PPM deviation from the simulation's emulated/exact *systemTime* reference.

NOTE—Please be tolerant of the editor of this document, who just downloaded the gnuplot application and fft4 library today. These initial cut-and-paste of plots are primitive (to be improved, when EPS or other formats are understood) and no noise-spectrum plots (to better illustrate gain peaking) are currently available. Improvements expected soon…

D.2 Initialization transients

D.2.1 Cascaded 8 stations

A significant expected initialization transient is observed when all stations simultaneously start operations, as illustrated in Figure D.2. This can be contributed to inaccurate initial estimates of receiver's link-delay and transmitter's rate estimations. The transient delays (although significant) are much less than expected from designs based on many-sample grandmaster rate-syntonization delays within bridges.

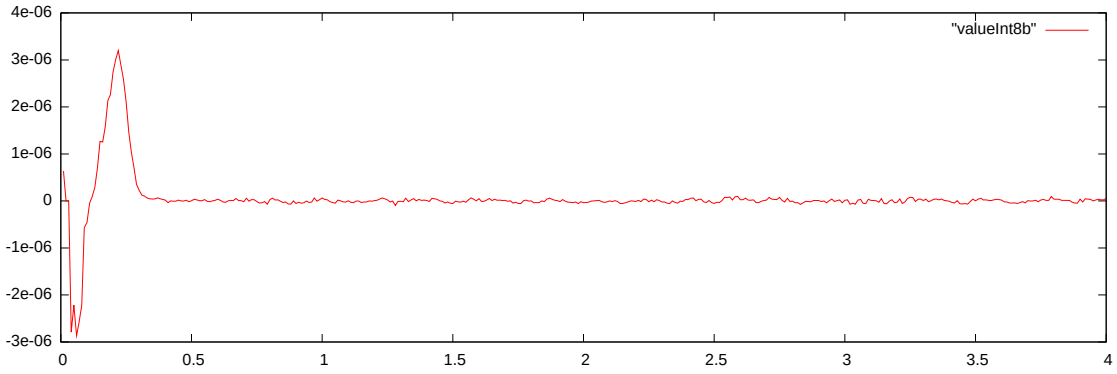

Figure D.2—Startup transients with 8 stations

D.2.2 Cascaded 64 stations

The length of the initialization transient increases when the number of bridges is increased to 64, as illustrated in Figure D.3. The much-longer duration of such transients is perhaps tolerable, but illustrates the desire to avoid extrapolation-based on many-sample grandmaster rate-syntonization delays within bridges.

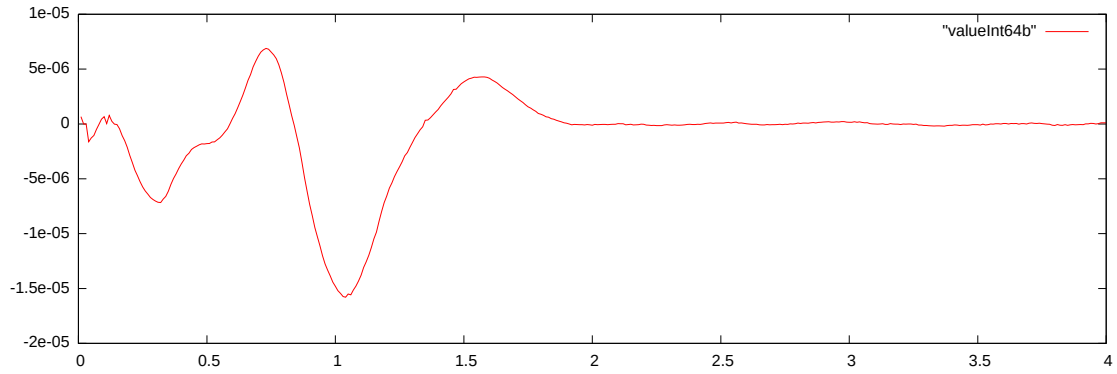

Figure D.3—Startup transients with 64 stations

D.3 Steady-state interpolation errors

D.3.1 Time interpolation with 8 stations

Simulations indicate modest peak-to-peak errors for 8-bridge topologies when interpolation-based protocols are used, as illustrated in Figure D.4.

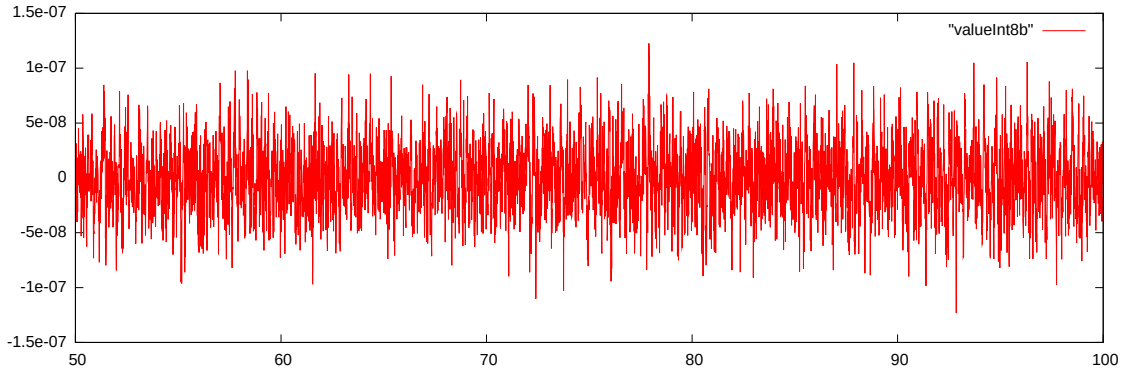

Figure D.4—Time interpolation with 8 stations

D.3.2 Time interpolation with 64 stations

Simulations indicate modest peak-to-peak error increases for 64-bridge topologies (as expected to 8-bridge topologies) when interpolation-based protocols are used, as illustrated in Figure D.5. The data is consistent with less-than-linear expectations, due to statistical averaging and intermediate interpolation filtering.

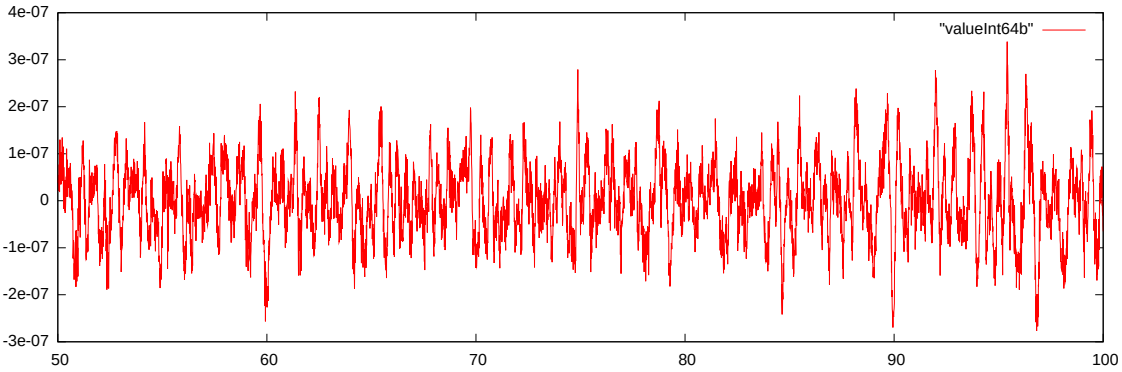

Figure D.5—Time interpolation with 64 stations

D.4 Steady-state extrapolation errors

D.4.1 Time extrapolation with 8 stations

Simulations indicate approximately twice the errors for 8-bridge topologies when extrapolation-based protocols (as opposed to interpolation-based protocols) are used, as illustrated in Figure D.6.

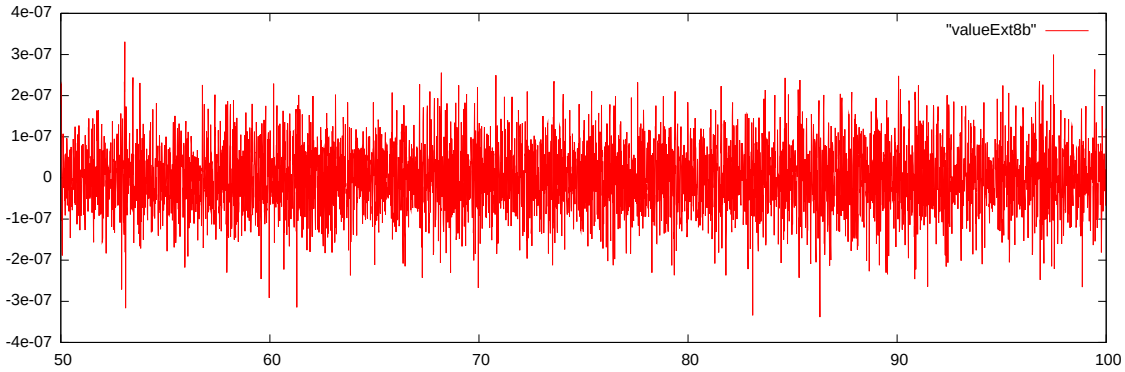

Figure D.6—Time extrapolation with 8 stations

D.4.2 Time extrapolation with 64 stations

Simulations indicate significantly larger peak-to-peak errors for 64-bridge topologies when extrapolation-based protocols (as opposed to interpolation-based protocols) are used, as illustrated in Figure D.7.

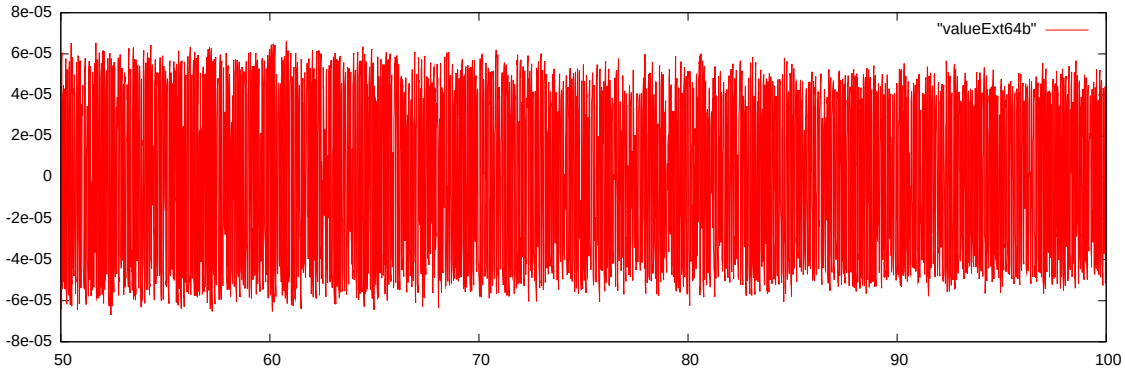

Figure D.7—Time extrapolation with 64 stations

Annex E

(informative)

Bridging to IEEE Std 1394

To illustrate the sufficiency and viability of the AVB time-synchronization services, the transformation of IEEE 1394 packets is illustrated.

E.1 Hybrid network topologies

E.1.1 Supported IEEE 1394 network topologies

This annex focuses on the use of AVB to bridge between IEEE 1394 domains, as illustrated in Figure E.1. The boundary between domains is illustrated by a dotted line, which passes through a SerialBus adapter station.

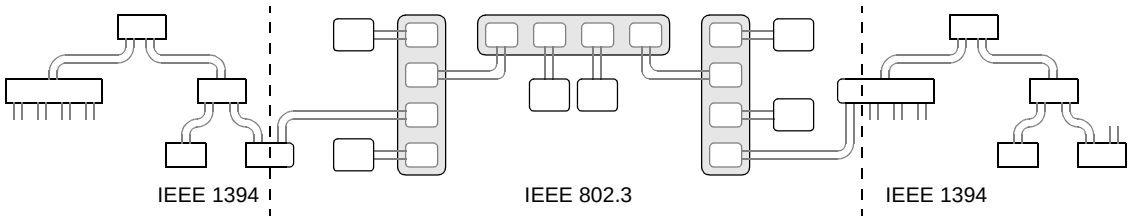

Figure E.1—IEEE 1394 leaf domains

E.1.2 Unsupported IEEE 1394 network topologies

Another approach would be to use IEEE 1394 to bridge between IEEE 802.3 domains, as illustrated in Figure E.2. While not explicitly prohibited, architectural features of such topologies are beyond the scope of this working paper.

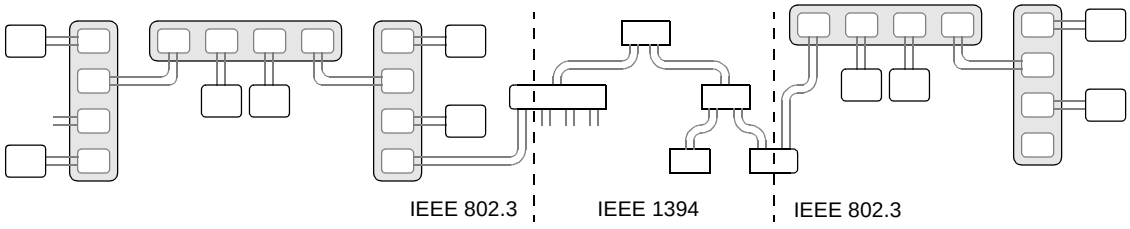

Figure E.2—IEEE 802.3 leaf domains

E.1.3 Time-of-day format conversions

The difference between AVB and IEEE 1394 time-of-day formats is expected to require conversions within the AVB-to-1394 adapter. Although multiplies are involved in such conversions, multiplications by constants are simpler than multiplications by variables. For example, a conversion between AVB and IEEE 1394 involves no more than two 32-bit additions and one 16-bit addition, as illustrated in Figure E.3.

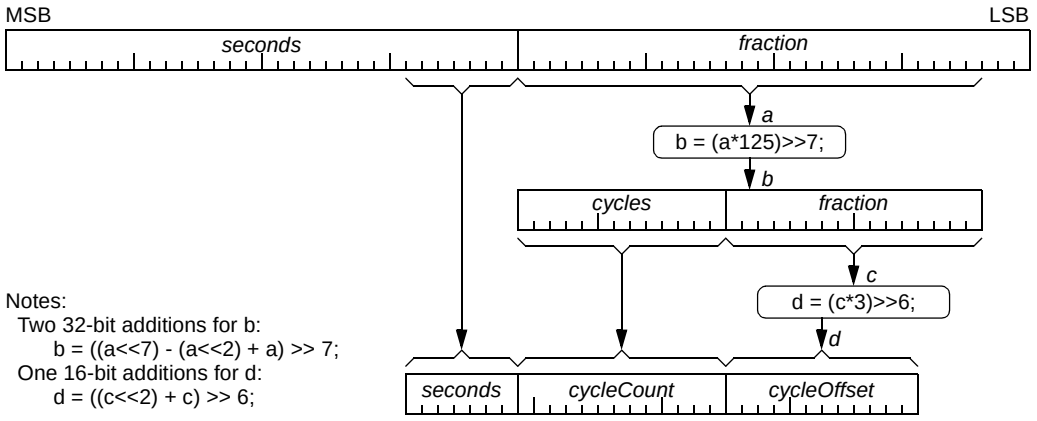

Figure E.3—Time-of-day format conversions

E.1.4 Grandmaster precedence mappings

Compatible formats allow either an IEEE 1394 or IEEE 802.3 stations to become the network's grandmaster station. While difference in format are present, each format can be readily mapped to the other, as illustrated in Figure E.4:

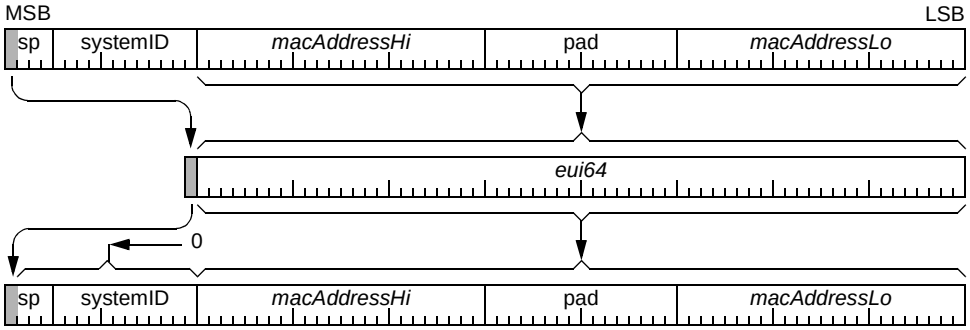

Figure E.4—Grandmaster precedence mapping

Annex F

(informative)

Time-of-day format considerations

To better understand the rationale behind the ‘extended binary’ timer format, various possible formats are described within this annex.

F.1 Possible time-of-day formats

F.1.1 Extended binary timer formats

The extended-binary timer format is used within this working paper and summarized herein. The 64-bit timer value consist of two components: a 40-bit *seconds* and 40-bit *fraction* fields, as illustrated in Figure F.1.

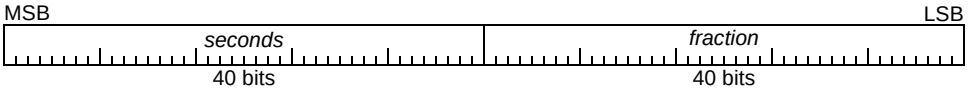

Figure F.1—Global-time subfield format

The concatenation of 40-bit *seconds* and 40-bit *fraction* field specifies an 80-bit *time* value, as specified by Equation F.1.

$$time = seconds + (fraction / 2^{40}) \quad \text{(F.1)}$$

Where:

seconds is the most significant component of the time value.
fraction is the less significant component of the time value.

F.1.2 IEEE 1394 timer format

An alternate “1394 timer” format consists of *secondCount*, *cycleCount*, and *cycleOffset* fields, as illustrated in Figure F.2. For such fields, the 12-bit *cycleOffset* field is updated at a 24.576MHz rate. The *cycleOffset* field goes to zero after 3071 is reached, thus cycling at an 8kHz rate. The 13-bit *cycleCount* field is incremented whenever *cycleOffset* goes to zero. The *cycleCount* field goes to zero after 7999 is reached, thus restarting at a 1Hz rate. The remaining 7-bit *secondCount* field is incremented whenever *cycleCount* goes to zero.

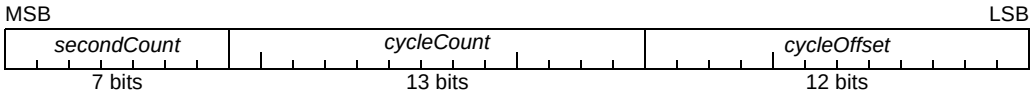

Figure F.2—IEEE 1394 timer format

F.1.3 IEEE 1588 timer format

IEEE Std 1588-2002 timer format consists of seconds and nanoseconds fields components, as illustrated in Figure F.3. The nanoseconds field must be less than 10^9; a distinct *sign* bit indicates whether the time represents before or after the epoch duration.

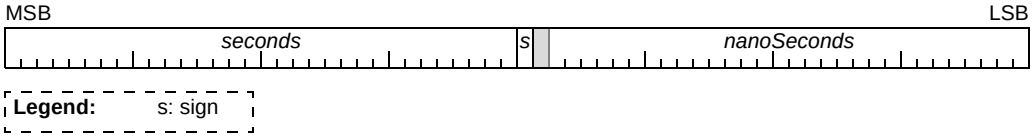

Figure F.3—IEEE 1588 timer format

F.1.4 EPON timer format

The IEEE 802.3 EPON timer format consists of a 32-bit scaled nanosecond value, as illustrated in Figure F.4. This clock is logically incremented once each 16 ns interval.

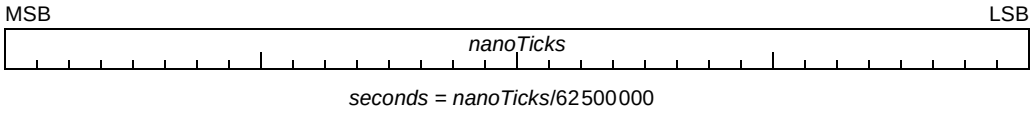

Figure F.4—EPON timer format