IEEE P1588™ D2.2
Draft Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems

Prepared by the

Precise Networked Clock Synchronization Working Group of the

IM/ST Committee

IEEE Standards Activities Department
Standards Licensing and Contracts
445 Hoes Lane, P.O. Box 1331
Piscataway, NJ 08855-1331, USA

Abstract

This standard specifies a protocol enabling precise synchronization of clocks in measurement and control systems implemented with technologies such as network communication, local computing and distributed objects. The protocol is applicable to systems communicating via packet networks. The protocol enables heterogeneous systems that include clocks of various inherent precision, resolution and stability to synchronize. The protocol supports system-wide synchronization accuracy and precision in the sub-microsecond range with minimal network and local clock computing resources. The default behavior of the protocol allows simple systems to be installed and operated without requiring the management attention of users.

Keywords: clock, distributed system, master clock, measurement and control system, real time clock, synchronized clock, boundary clock, transparent clock

Introduction

(This introduction is not part of IEEE P1588 D2.0, Draft Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems.)

This standard defines a protocol enabling precise synchronization of clocks in measurement and control systems implemented with technologies such as network communication, local computing and distributed objects. The clocks communicate with each other over a communication network. The protocol generates a master slave relationship among the clocks in the system. All clocks ultimately derive their time from a clock known as the grandmaster clock. In its basic form, this protocol is intended to be administration free.

History

Measurement and control applications are increasingly using distributed system technologies such as network communication, local computing, and distributed objects. Without a standardized protocol for synchronizing the clocks in these devices, it is unlikely that the benefits will be realized in the multi-vendor system component market. Existing protocols for clock synchronization are not optimum for these applications. For example, NTP, Network Time Protocol, targets large distributed computing systems with millisecond synchronization requirements. The protocol proposed in this standard specifically addresses the needs of measurement and control systems:

- Spatially localized,
- Microsecond to sub-microsecond accuracy and precision,
- Administration free, and most importantly
- Accessible for both high-end devices and low cost low-end devices.

Patent rights

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. A patent holder or patent applicant has filed a statement of assurance that it will grant licenses under these rights without compensation or under reasonable rates, with reasonable terms and conditions that are demonstrably free of any unfair discrimination to applicants desiring to obtain such licenses. Other Essential Patent Claims may exist for which a statement of assurance has not been received. The IEEE is not responsible for identifying Essential Patent Claims for which a license may be required, for conducting inquiries into the legal validity or scope of Patents Claims, or determining whether any licensing terms or conditions are reasonable or non-discriminatory. Further information may be obtained from the IEEE Standards Association.

Participants

At the time this draft standard was completed, the Precise Networked Clock Synchronization Working Group had the following membership:

John Eidson, *Chair*

Hans Weibel, *Vice-chair*

Silvana Rodrigues, *Secretary*

John D MacKay, *Editor*

Galina Antonova
Doug Arnold
Sivaram Balasubramanian
P. Stephan Bedrosian
Stewart Bryant
Chris Calley
George Claseman
Ron Cohen
Robert Cubbage
John C Eidson
Tom Farley
John Fischer

John Fleck
Georg Gaderer
Geoffrey M Garner
Michael Gerstenberger
Franz-Josef Götz
Bruce Hamilton
Kenneth Hann
Ken Harris
Jim Innis
Joel Keller
Jacob Kornerup
Kang Lee
John D MacKay
Dirk S. Mohl
Anatoly Moldovansky
Laurent Montini
Paul Myers
Karen O'Donoghue
Jonathon D. Paul
Stephen Peterson
Antti Pietilainen
Bill Powell
Silvana Rodrigues
David Roe
David Rosselot
Stephan Schüler
Markus Seehofer
Mark Shepard
Veselin Skendzic
Dave Tonks
Richard Tse
Aljosa Vrancic
Hans Weibel
Ludwig Winkel
Taylor Wray
Gabriel Zigelboim

The following members of the balloting committee voted on this standard. Balloters may have voted for approval, disapproval, or abstention.

(to be supplied by IEEE)

Acknowledgements

The working group would like to acknowledge Sara Bitan, William Burr, Ray Perlner, Albert Treytl, Steve Singer and Yahel Shlomi for their contributions to the security experimental extension.

CONTENTS

TABLES

FIGURES

Draft Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems

1. Overview

1.1 Scope

This standard defines a protocol enabling precise synchronization of clocks in measurement and control systems implemented with technologies such as network communication, local computing and distributed objects. The protocol is applicable to systems communicating by local area networks supporting multicast messaging including but not limited to Ethernet. The protocol enables heterogeneous systems that include clocks of various inherent precision, resolution, and stability to synchronize to a grandmaster clock. The protocol supports system-wide synchronization accuracy in the sub-microsecond range with minimal network and local clock computing resources. The default behavior of the protocol allows simple systems to be installed and operated without requiring the administrative attention of users. The standard includes mappings to UDP/IP, DeviceNet and a layer-2 Ethernet implementation. It includes formal mechanisms for message extensions, higher sampling rates, correction for asymmetry, a clock type to reduce error accumulation in large topologies, and specifications on how to incorporate the resulting additional data into the synchronization protocol. The standard permits synchronization accuracies better than 1 nanosecond. The protocol has features to address applications where redundancy and security are a requirement. The standard defines conformance and management capability. There is provision to support unicast as well as multicast messaging. The standard includes an annex on recommended practices. Annexes defining communication-medium-specific implementation details for additional network implementations are expected to be provided in future versions of this standard.

1.2 Purpose

Measurement and control applications are increasingly employing distributed system technologies such as network communication, local computing, and distributed objects. Many of these applications will be enhanced by having an accurate system-wide sense of time achieved by having local clocks in each sensor, actuator, or other system device. Without a standardized protocol for synchronizing these clocks, it is unlikely that the benefits will be realized in the multi-vendor system component market. Existing protocols for clock synchronization are not optimum for these applications. For example, Network Time Protocol (NTP), targets large distributed computing systems with millisecond synchronization requirements. The protocol in this standard specifically addresses the needs of measurement and control and operational systems in the fields of test and measurement, industrial automation, military systems, manufacturing systems, power utility systems and certain telecommunications applications. These applications need:

— Spatially localized systems with options for larger systems,

— Microsecond to sub-microsecond accuracy

— Administration free operation

— Applicability for both high-end devices and low-cost, low-end devices

— Provisions for the management of redundant and fault tolerant systems.

A number of different application areas such as industrial automation, telecommunication, semiconductor manufacturing, military systems, and utility power generation have emerged that require the standard to be revised.

1.3 Layout of the document

This standard, which defines the Precision Time Protocol (PTP), is divided into 19 clauses:

Clause	**Purpose**
1	Provides the scope and benefits of this standard
2	Lists references to other standards
3	Provides definitions that are either not found in other standards or have been modified for use with this standard
4	Provides conventions for the notation used in this standard
5	Defines the data types used in this standard
6	Provides an overview of PTP
7	Defines characteristics of PTP entities
8	Defines PTP data sets
9	Defines PTP for ordinary and boundary clocks
10	Defines PTP for transparent clocks
11	Specifies PTP time computations and corrections
12	Specifies how to syntonize and synchronize clocks
13	Defines the format of messages passed between participating clocks
14	Specifies type,length,value (TLV) formats
15	Specifies management TLVs
16	Defines general optional features of this standard
17	Defines state configuration options of this standard
18	Defines forward and backward compatibility between versions
19	Defines requirements for conformance

Annexes are provided as follows:

Annex	**Purpose**
A	Using PTP
B	Defines timescales and epochs in PTP
C	Examples of timing computations and message fields
D	Defines mappings of PTP to User Datagram Protocol (UDP) over Internet Protocol version 4 (IPv4)
E	Defines mappings of PTP to UDP over Internet Protocol version 6 (IPv6)
F	Defines mappings of PTP over IEEE 802.3
G	Defines mappings of PTP to DeviceNet™ [1]
H	Defines mappings of PTP to ControlNet™ [2]

[1] DeviceNet™ is a trade name of Open DeviceNet Vendor Association, Inc. This information is given for the convenience of users of this standard and does not constitute an endorsement by the IEEE or IEC of the trademark holder or any of its products. Compliance to this standard does not require use of the trade name DeviceNet™. Use of the trade name DeviceNet™ requires permission of Open DeviceNet Vendor Association, Inc.

I	Defines mappings of PTP to PROFINET™ [3]
J	Default PTP Profile
K	Defines an experimental security option
L	Defines an experimental cumulative frequency TLV
M	Bibliography

[2] ControlNet™ is a trade name of ControlNet International, Ltd. This information is given for the convenience of users of this International standard and does not constitute an endorsement by the IEEE or IEC of the trademark holder or any of its products. Compliance to this profile does not require use of the trade name ControlNet™. Use of the trade name ControlNet™ requires permission of ControlNet International, Ltd.

[3] PROFIBUS and PROFINET are the trade names of the non-profit organization PROFIBUS Nutzerorganisation e.V. (PNO). This information is given for the convenience of users of this standard and does not constitute an endorsement by the IEEE or IEC of the trade names holder or any of its products. Compliance to this standard does not require use of the registered logos. Use of the logos requires permission of the trade name holder

2. Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

IEEE Std 802®, IEEE Standard for Local and Metropolitan Area Networks: Overview and Architecture[4]

ISO/IEC 10646:2003 Information technology — Universal Multiple-Octet Coded Character Set (UCS) — Part 1: Architecture and Basic Multilingual Plane[5]

IEC 61158-3-2:2007, Industrial communication networks – Fieldbus specifications – Part 3-2 (Ed.1.0): Data-link layer service definition – Type 2 elements

IEC 61158-4-4:2007, Industrial communication networks – Fieldbus specifications – Part 4-2 (Ed.1.0): Data-link layer protocol specification – Type 2 elements

IEC 61158-5-2:2007, Industrial communication networks – Fieldbus specifications – Part 5-2 (Ed.1.0): Application layer service definition – Type 2 elements

IEC 61158-6-2:2007, Industrial communication networks – Fieldbus specifications – Part 6-2 (Ed.1.0): Application layer protocol specification – Type 2 elements

IEC 62026-3:2007, Low-voltage switchgear and controlgear - Controller-device interfaces (CDIs) - Part 3: DeviceNet

IEC 61158-5-10:2007, Industrial communication networks – Fieldbus specifications — Part 5-10: Application layer service definition – Type 10 elements

IEC 61158-6-10:2007, Industrial communication networks – Fieldbus specifications — Part 6-10: Application layer protocol specification – Type 10 elements

IEC 61784-1:2007, Industrial communication networks – Profiles – Part 1: Fieldbus profiles

IEC 61784-2:2007, Industrial communications networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC 8802-3

IEEE 802.3-2005, IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and Physical Layer specifications

IEEE 802.1Q-2005, IEEE Standards for Information technology – Telecommunications and information exchange between systems – IEEE standard for Local and metropolitan area networks – Part 1Q: Virtual bridged local area networks

[4] IEEE Publications are available from the Institute of Electrical and Electronics Engineers. 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA and < http://standards.ieee.org/getieee802/802.html >.

[5] IEC publications are available from the Sales Department of the International Electrotechnical Commission, Case Postale 131, 3, rue de Varembé, CH-1211, Genève 20, Switzerland/Suisse (http://www.iec.ch/). IEC publi-cations are also available in the United States from the Sales Department, American National Standards Institute, 11 West 42nd Street, 13th Floor, New York, NY 10036, USA.

IEEE 802.1AB IEEE Standards for Information technology – Telecommunications and information exchange between systems - IEEE standard for Local and metropolitan area networks - Part 1AB: Station and Media Access Control Connectivity Discovery

3. Definitions, acronyms, and abbreviations

3.1 Definitions

For the purposes of this document, the following terms and definitions apply.

3.1.1 accuracy: The mean of the time or frequency error between the clock under test and a perfect reference clock, over an ensemble of measurements. Stability is a measure of how the mean varies with respect to variables such as time, temperature, etc. The precision is a measure of the deviation of the error from the mean.

3.1.2 atomic process: A process is atomic if the values of all inputs to the process are not permitted to change until all of the results of the process are instantiated, and the outputs of the process are not visible to other processes until the processing of each output is complete.

3.1.3 clock: A node participating in the PTP protocol that is capable of providing a measurement of the passage of time since a defined epoch

NOTE — In the case of PTP ordinary and boundary clocks that are properly synchronized, the epoch is the epoch of the timescale in use. In the case of PTP transparent clocks, the epoch is locally defined and not necessarily aligned with the timescale.

3.1.4 default: In this document the word default when applied to attribute values and options means the configuration of a PTP device as it is delivered from the manufacturer.

direct communication: A communication of PTP information between two ordinary or boundary clocks with no intervening boundary clock.

3.1.5 domain: A logical grouping of clocks that synchronize to each other using the protocol, but that are not necessarily synchronized to clocks in another domain

3.1.6 end-to-end transparent clock: A transparent clock that supports the use of end-to-end delay measurement mechanism between slave clocks and the master clock

3.1.7 epoch: The origin of a timescale.

3.1.8 event: An abstraction of the mechanism by which signals or conditions are generated and represented.

3.1.9 foreign master: An ordinary or boundary clock sending Announce messages to another clock that is not the current master recognized by the other clock.

3.1.10 fractional frequency offset: The fractional frequency offset FFO between a measured frequency and a reference frequency is defined as follows:

$$FFO = \frac{(FO - FR)}{FR}$$

Where FO is the measured frequency and FR is the reference frequency..

3.1.11 grandmaster clock: Within a domain, a clock that is the ultimate source of time for clock synchronization using the protocol.

3.1.12 holdover: A clock previously synchronized/syntonized to another clock (normally a primary reference or a master clock) but now free-running based on its own internal oscillator, whose frequency is being adjusted using data acquired while it had been synchronized/syntonized to the other clock, is said to be in holdover or in the holdover mode, as long as it is within its accuracy requirements.

3.1.13 link: A network segment between two PTP ports supporting the peer delay mechanism of this standard. The peer delay mechanism is designed to measure the propagation time over such a link.

3.1.14 management node: A device that configures and monitors clocks.

3.1.15 master clock: In the context of a single PTP communication path, a clock that is the source of time to which all other clocks on that path synchronize.

3.1.16 message timestamp point: A point within a PTP event message serving as a reference point in the message. A timestamp is defined by the instant a message timestamp point passes the reference plane of a clock.

3.1.17 multicast communication: A communication model in which each PTP message sent from any PTP port is capable of being received and processed by all PTP ports on the same PTP communication path.

3.1.18 node: A device that can issue or receive PTP communications on a network.

3.1.19 one-step clock: A clock that provides time information using a single event message.

3.1.20 ordinary clock: A clock that has a single PTP port in a domain and maintains the timescale used in the domain. It may serve as a source of time, i.e. be a master clock, or may synchronize to another clock, i.e. be a slave clock.' 'boundary clock: A clock that has multiple PTP ports in a domain and maintains the timescale used in the domain. It may serve as the source of time, i.e. be a master clock, and may synchronize to another clock, i.e. be a slave clock.

3.1.21 parent clock: Parent clock is synonymous with master clock. The master clock to which a clock is synchronized.

3.1.22 peer-to-peer transparent clock: A transparent clock that, in addition to providing PTP event transit time information, also provides corrections for the propagation delay of the link connected to the port receiving the PTP event message. In the presence of peer-to-peer transparent clocks, delay measurements between slave clocks and the master clock are performed using the peer-to-peer delay measurement mechanism.

3.1.23 phase change rate: The observed rate of change in the measured time with respect to the reference time. The phase change rate is equal to the fractional frequency offset between the measured frequency and the reference frequency, see 3.1.11.

3.1.24 portNumber: An index identifying a specific PTP port on a PTP node.

3.1.25 precision: See **accuracy**.

3.1.26 Precision Time Protocol (PTP): The protocol defined by this standard. As an adjective, it indicates that the modified noun is specified in or interpreted in the context of this standard.

3.1.27 primary reference: A source of time and or frequency that is traceable to international standards, see **traceable**

3.1.28 profile: The set of allowed PTP features applicable to a device.

3.1.29 PTP communication: Information used in the operation of the protocol, transmitted in a PTP message over a PTP communication path.

3.1.30 PTP communication path: The signaling path portion of a particular network enabling direct communication among ordinary and boundary clocks.

3.1.31 PTP message: One of the message types defined in this standard.

3.1.32 PTP node: A PTP ordinary, boundary, or transparent clock or a device that generates or parses PTP messages.

3.1.33 PTP port: A logical access point of a clock for PTP communications to the communications network.

3.1.34 recognized standard time source: A recognized standard time source is a source external to PTP that provides time and or frequency as appropriate that is traceable to the international standards laboratories maintaining clocks that form the basis for the **International Atomic Time** (TAI) and Universal Coordinated Time (UTC) timescales. Examples of these are GPS, NTP, and NIST timeservers.

3.1.35 requestor: The port implementing the peer-to-peer delay mechanism that initiates the mechanism by sending a Pdelay_Req message.

3.1.36 responder: The port responding to the receipt of a Pdelay_Req message as part of the operation of the peer-to-peer delay mechanism

3.1.37 stability: See **accuracy.**

3.1.38 synchronized clocks: Two clocks are synchronized to a specified uncertainty if they have the same epoch and their measurements of the time of a single event at an arbitrary time differ by no more than that uncertainty.

3.1.39 syntonized clocks: Two clocks are syntonized if the duration of the second is the same on both, which means the time as measured by each advances at the same rate. They may or may not share the same epoch.

3.1.40 timeout: A mechanism for terminating requested activity that, at least from the requester's perspective, does not complete within the specified time.

3.1.41 timescale: A linear measure of time from an epoch.

3.1.42 traceability: A property of the result of a measurement or the value of a standard whereby it can be related to stated references, usually national or international standards, through an unbroken chain of comparisons all having stated uncertainties." [M26]

3.1.43 translation device: A boundary clock or in some cases a transparent clock that translates the protocol messages between regions implementing different transport and messaging protocols, between different versions of this standard, or different PTP profiles.

3.1.44 transparent clock: A device that measures the time taken for a PTP event message to transit the device and provides this information to clocks receiving this PTP event message, see end-to-end transparent clock and peer-to-peer transparent clock

3.1.45 two-step-clock: A clock that provides time information using the combination of an event message and a subsequent general message. See: **one-step clock**.

3.2 Acronyms and abbreviations

ARB	arbitrary
BMC	best master clock
CAN	Controller Area Network
CP	Communication Profile [according to IEC 61784-1]
CPF	Communication Profile Family [according to IEC 61784-1]
DS	differentiated service
E2E	end-to-end
GPS	Global Positioning System
IANA	Internet Assigned Numbers Authority
ICV	integrity check value
ID	identification
IPv4, IPv6	Internet Protocol version 4 / 6
JD	Julian Date
JDN	Julian Day Number
MAC	media access controller [according to IEEE 802.3]
MJD	Modified Julian Day
NIST	National Institute of Standards and Technology (see www.nist.gov)
NTP	Network Time Protocol[M6]
OUI	organizational unique identifier (allocated by IEEE) see note
P2P	peer-to-peer
PHY	physical layer [according to IEEE 802.3]
POSIX	Portable Operating System Interface (see ISO/IEC 9945:2003)
PPS	pulse per second
PTP	Precision Time Protocol
SA	security associations
SNTP	Simple Network Time Protocol
TAI	International Atomic Time
TC	traffic class
TLV	type, length, value [according to IEEE 802.1AB]
ToS	type of service
UCMM	UnConnect Message Manager
UDP/IP	User Datagram Protocol (see RFC 768 [M14]) / Internet Protocol (see RFC 791 [M15])
UTC	Coordinated Universal Time

NOTE – the organizational unique identifier (OUI) is typically used in specifications or the implementation of devices for the purpose of identification. It identifies the organization that owns the OUI-depdendent subidentifier and may not necessarily be the organization that defines the specification or provides the hardware. The IEEE OUI listing can be obtained at: http://standards.ieee.org/regauth/oui/index.shtml

4. Conventions

4.1 Descriptive lexical form syntax

4.1.1 Lexical form syntax

A lexical form refers to:

— A name

— A data type.

The conventions illustrated in the following list regarding lexical forms are used in this standard:

- Type names: e.g. ClockQuality (no word separation, initial letter of each word capitalized)
- Enumeration members and global constants: e.g. ATOMIC_CLOCK (underscore word separation, all letters capitalized)
- Fields within messages, instances of structures, and variables: e.g. secondsField, clockQuality, clockIdentity (two word field names at a minimum, no word separation, initial word not capitalized, initial letter capitalization on subsequent words)
- Members of a structure: e.g. clockQuality.clockClass (structure name followed by a period followed by the member name)
- Data set name: e.g. defaultDS, parentDS, portDS, currentDS, timePropertiesDS (no word separation, initial letter of each word not capitalized followed by the letters DS)
- Data set members: e.g. defaultDS.clockQuality.clockClass (Data set name followed by a period followed a type name followed by a period followed by the variable name)
- <localNameForSomething>: text enclosed in angle brackets, < >, is used where the standard needs to refer to something whose syntax or lexical form is dependent on the local implementation and language.

When a lexical form appears in text, as opposed to in a type, or a format definition, the form is to be interpreted as singular, plural or possessive as appropriate to the context of the text.

4.2 Word usage

4.2.1 Shall

The word shall, equivalent to “is required to,” is used to indicate mandatory requirements, strictly to be followed in order to conform to the standard and from which no deviation is permitted.

4.2.2 Recommended

The word recommended is used to indicate flexibility of choice with a strong preference alternative.

4.2.3 Must

The word must indicates an unavoidable situation.

4.2.4 Should

The word should, equivalent to “is recommended that,” is used to indicate

- Among several possibilities one is recommended as particularly suitable, without mentioning or excluding others.
- That a certain course of action is preferred but not required.
- That (in the negative form) a certain course of action is deprecated but not prohibited.

4.2.5 May

The word may, equivalent to “is permitted,” is used to indicate a course of action permissible within the limits of the standard.

4.2.6 Can

The word can, equivalent to “is able to,” is used to indicate possibility and capability, whether material or physical.

4.2.7 Optional

Clauses and text marked optional are not required to be implemented. If the option is implemented, then all specifications of the clause or text respectively shall be implemented according to this standard.

NOTE— This definition is recursive, which means that options within options obey these same rules.

4.2.8 Reserved

The word reserved indicates:

- If used in an assignment of values to an enumeration or an attribute that the values indicated are reserved for use in future editions of this standard and shall not be used for any other purpose.
- If used in a field of a message that the field is reserved for use in future editions of this standard. The field shall be present in the message with the size specified. No interpretation of reserved fields is to be made for this edition of this standard and the fields shall not be used for any other purpose.

4.3 Behavioral specification notation

State transition diagrams are used to specify behavioral characteristics as illustrated in Figure 1. Each state transition diagram is composed of the following components:

- Named boxes, representing states
- Directed arrows, indicating transitions from one state to the next.

Each transition is labeled with:

- The enabling event or predicate label for a transition and
- The transition action label for a transition.

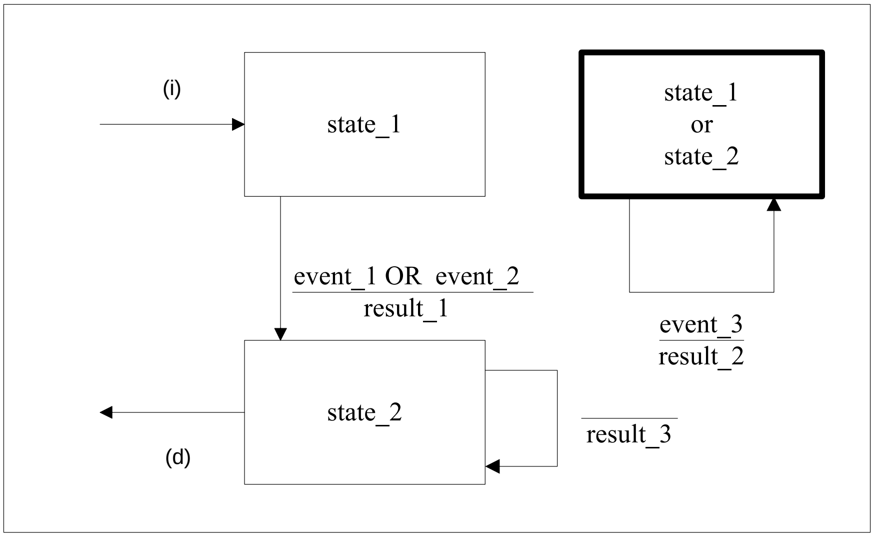


Figure 1: Mealy state transition diagram

The notation used describes state transition diagrams using the Mealy style, where actions are associated with the transition from one state to another.

Events, for example “event_1,” “event_2” and “event_3,” identify the inputs to the state machine. They can be operation requests and responses, or internal occurrences such as timer expirations.

Predicates, for example “event_1 OR event_2,” identify enabling conditions for transitions. The first predicate encountered, evaluated from left to right, which is TRUE, selects the transition to execute and therefore the next state.

Transition actions, for example “result_1,” are the actions that are executed before transitioning to the next state.

The next state identifies the state for the state machine after the selected transition action completes. The value of the current state changes as the transition to the next state occurs.

A bold line for a state box indicates that the box represents multiple states. Any transition shown that begins and terminates in such a state box indicates that there has been no change in state.

Transitions, for example the transition resulting in result_3, which have no indicated enabling conditions, occur via unspecified mechanisms. Unless otherwise stated, in PTP the events giving rise to these mechanisms are implementation-specific and outside the scope of the standard.

A transition into a state machine, for example “(i),” is indicated by a transition arrow that has no source state. A transition out of a state machine, for example “(d),” is indicated by a transition arrow with no destination state.

NOTE—For example: As a result of either event_1 or event_2 becoming TRUE, state_1 is replaced with the value of the next state. In this example the next state is state_2, which is specified as the name of the state box that is the target of the transition arrow. Before the transition, result_1 occurs. “event_3” can occur in either state_1 or state_2. The state is unchanged but an action, result_2, occurs as the result of event_3.

5. Data types and on-the-wire formats in a PTP system

5.1 General

The data types specified for the various PTP variables and message fields define logical properties that are necessary for correct operation of the protocol or interpretation of PTP message content.

Implementations are free to use any internal representation of PTP data types; however the internal representation shall not change the semantics of any quantity visible via PTP communications or the semantics of any specified operation of the protocol.

5.2 Primitive data types specifications

All non-primitive PTP data types are derived from the primitive types listed in Table 1. These types are not tied to any specific programming language. The essential properties of each type shall be as follows:

— Integer: All integer types are of finite length as indicated by the number associated with each, e.g. UInteger48, and as signed or unsigned as indicated by the absence or presence of a leading U. Numbers with these data types obey the laws of arithmetic within the range represented by the length. Arithmetic operations are treated as modulo the capacity of the data type, e.g. the sum of two UInteger48 values is computed modulo 2^{+48}. Signed integers are represented in two's complement form.

— Enumeration: All enumerations are of finite field length as indicated by the number associated with each, e.g. Enumeration4. Unless otherwise stated in this standard the only interpretations of the bit pattern in the enumeration field are the association between the bit patterns and the assigned meanings of the enumeration.

— Boolean: The only interpretation is as logical values within the context of Boolean algebra.

Nibble and Octet: These are 4 and 8 bit fields respectively. The only interpretations are those explicitly defined within this standard.

Table 1: Primitive PTP data types

Data type	Definition
Boolean	TRUE or FALSE.
Enumeration4	4-bit enumerated value
Enumeration8	8-bit enumerated value
Enumeration16	16-bit enumerated value
UInteger4	4-bit unsigned integer
Integer8	8-bit signed integer
UInteger8	8-bit unsigned integer
Integer16	16-bit signed integer
UInteger16	16-bit unsigned integer
Integer32	32-bit signed integer
UInteger32	32-bit unsigned integer
UInteger48	48-bit unsigned integer
Integer64	64-bit signed integer
Nibble	4-bit field not interpreted as a number
Octet	8-bit field not interpreted as a number

5.3 Derived data type specifications

5.3.1 General

Arrays of any of the primitive data types are represented in the format <data type>[length] <label>, where <length> indicates the number of instances of the data type in the array and <label> is the lexical name for the array data type so defined.

Structures consisting of an ordered list of members is indicated with the syntax:

```
struct <StructureName>
{
    <DataType1> <memberName1>;

    <DataType2> <memberName2>;


    …
};
```

where <StructureName> is the lexical name for the data type so defined, <DataType1> is the data type of the first member and <memberName1> the lexical name of the first member and so forth.

The syntax typedef <DataType> <TypeName> is interpreted as defining a derived data type with the same properties as the data type defined by <DataType> but with a new name given by <TypeName>.

The syntax typedef <DataType> [length] <TypeName> is interpreted as defining a derived data type consisting of an array of elements of type <DataType>, but with a new name given by <TypeName>.

5.3.2 TimeInterval

The TimeInterval type represents time intervals.

```
struct TimeInterval
{
                Integer64 scaledNanoseconds;
};
```

The scaledNanoseconds member is the time interval expressed in units of nanoseconds and multiplied by 2^{+16}.

Positive or negative time intervals outside the maximum range of this data type shall be encoded as the largest positive and negative values of the data type respectively.

For example: 2.5 ns is expressed as: 0000 0000 0002 8000_{16}

5.3.3 Timestamp

The Timestamp type represents a positive time with respect to the epoch.

```
struct Timestamp
{
                UInteger48 seconds;
                UInteger32 nanoseconds;
};
```

The seconds member is the integer portion of the timestamp in units of seconds.

The nanoseconds member is the fractional portion of the timestamp in units of nanoseconds. The nanoseconds member is always less than 10^9.

For example:

+2.000000001 seconds is represented by seconds = 0000 0000 0002_{16} and nanoseconds= 0000 0001_{16}

5.3.4 ClockIdentity

The ClockIdentity type identifies a clock.

```
typedef Octet[8]  ClockIdentity;
```

5.3.5 PortIdentity

The PortIdentity type identifies a PTP port.

```
struct PortIdentity
{
		ClockIdentity clockIdentity;
		UInteger16 portNumber;
};
```

5.3.6 PortAddress

The PortAddress type represents the protocol address of a PTP port.

```
struct PortAddress
{
		Enumeration16 networkProtocol;
		UInteger16 addressLength;
		Octet[addressLength] address;
};
```

The value of the networkProtocol member shall be taken from the networkProtocol enumeration, see 7.4.1.

The addressLength is the length in octets of the address. The range shall be 1 to 16 octets.

The address member holds the protocol address of a port in the format defined by the mapping annex of the protocol as identified by the networkProtocol member. The most significant octet of the address is mapped into the octet of the address member with index 0.

5.3.7 ClockQuality

The ClockQuality represents the quality of a clock.

```
struct ClockQuality
{
		UInteger8 clockClass;
		Enumeration8 clockAccuracy;
		UInteger16 offsetScaledLogVariance;
};
```

5.3.8 TLV

The TLV type represents TLV extension fields.

```
struct TLV
{
                Enumeration16 tlvType;
                UInteger16 lengthField;
                Octet[lengthField ] value;
};
```

The length of all TLVs shall be an even number of octets.

5.3.9 PTPText

The PTPText data type is used to represent textual material in PTP messages.

```
struct PTPText
{
                UInteger8 lengthField;
                Octet[lengthField] text;
};
```

The text field shall be encoded as UTF-8 symbols as specified by ISO/IEC 10646. The most significant octet of the leading text symbol shall be the element of the array with index 0.

NOTE— A single UTF-8 symbol can be 1– 4 octets long. Therefore the lengthField value can be larger than the number of symbols.

5.3.10 FaultRecord

The FaultRecord type is used to construct fault logs.

```
struct FaultRecord
{
     UInteger16 faultRecordLength;
     Timestamp  faultTime;
     Enumeration8 severity;
     PTPText faultName;
     PTPText faultValue;
     PTPText faultDescription;
};
```

5.4 On-the-wire formats

5.4.1 General

PTP protocol data units consist of the PTP messages defined in clauses 13, 14, 15, 16, and 17 based on the data types defined in clauses 5 and 17.2. The internal ordering of the fields of the PTP protocol data units is specified in subclauses 5.4.2 to 5.4.4.

5.4.2 Primitive data types

Numeric primitive data types defined in 5.2 shall be formatted with the most significant octet nearest the beginning of the protocol data unit followed in order by octets of decreasing significance.

The Boolean data type TRUE shall be formatted as a single bit equal to 1 and FALSE as a single bit equal to 0.
Enumerations of whatever length shall be formatted as though the assigned values are unsigned integers of the same length, e.g. Enumeration16 shall be formatted as though the value had type UInteger16.

5.4.3 Arrays of primitive types

All arrays shall be formatted with the member having the lowest numerical index closest to the start of the protocol data unit followed by successively higher numbered members. In octet arrays, the octet with the lowest numerical index is termed the most significant octet.

When a field containing more than one octet is used to represent a numeric value, the most significant octet shall be nearest the start of the protocol data unit, followed by successively less significant octets.

When a single octet contains multiple fields of primitive data types, the bit positions within the octet of each of the primitive types as defined in the message field specification shall be preserved. For example, the first field of the header of the PTP messages is a single octet composed of two fields, one of type nibble bits 4-7, and one of type Enumeration4 bits 0-3, see 13.3.1.

5.4.4 Derived data types

Derived data types defined as structs shall be formatted with the first member of the struct closest to the beginning of the protocol data unit followed by each succeeding member. Each member shall be formatted according to its data type.

Derived data types defined as typedefs shall be formatted according to its referenced data type.

5.4.5 Mapping of PTP protocol data units into their on-the-wire formats

Unless otherwise specified, PTP protocol data units shall be mapped to and from their on-the-wire format based on the rules of the underlying physical layer transport. Any exceptions are specified in one of the transport specific Annexes of this standard or in an applicable PTP profile.

NOTE – PTP protocol mechanisms operate in the upper layers of the protocol stack (i.e., PTP is an "application" that uses the services of the network or link layer). The physical layer transport dictates the on-the-wire format.

6. Clock synchronization model

6.1 General

Clause 6 provides a model for understanding the operation of the Precision Time Protocol. The exact specifications of these interactions are found in subsequent clauses.

The PTP standard specifies a clock synchronization protocol. This protocol is applicable to distributed systems consisting of one or more nodes, communicating over a network. Nodes are modeled as containing a real-time clock that may be used by applications within the node for various purposes such as generating timestamps for data or ordering events managed by the node. The protocol provides a mechanism for synchronizing the clocks of participating nodes to a high degree of accuracy and precision. This standard specifies:

a) The Precision Time Protocol, and

b) The node, system, and communication properties necessary to support PTP.

6.2 Principle assumptions about the network and implementation recommendations

The following are the principle assumptions and recommendations that should be followed in order to ensure correct operation of the protocol. These are discussed in greater detail in later clauses.

a) PTP assumes that the network eliminates cyclic forwarding of PTP messages within each communication path (e.g. by using spanning tree protocol). PTP eliminates cyclic forwarding of PTP messages between communication paths.

b) PTP is tolerant of occasional missed message, duplicated message, or message that arrived out of order. However, PTP assumes that such impairments are relatively rare.

c) PTP was designed assuming a multicast communication model. PTP also supports a unicast communication model as long as the behavior of the protocol is preserved. PTP assumes that Announce messages are periodically sent by one port and delivered to all other ports of ordinary or boundary clocks within a communication path. If the communication path consists of more than two ports, the assumption is that Announce messages are either sent in multicast or the Announce information is replicated to all ports in the communication path using unicast messages. PTP ports discover other ports within a communication path through the receipt of multicast Announce messages. When multicast communication is not available another form of discovery (e.g. by configuration) is required, see for example 17.5. PTP management messages sent to all ports also require either multicast messaging or replication of the management message to all ports within the communication path.

d) Like all message-based time transfer protocols, PTP time accuracy is degraded by asymmetry in the paths taken by event messages, see 7.4.2. Specifically the time offset error is 1/2 of the asymmetry. Asymmetry is not detectable by PTP; however if known, PTP corrects for asymmetry. Asymmetry can be introduced in the physical layer, e.g. via transmission media asymmetry, by bridges and routers, and in large systems by the forward and reverse paths traversed by event messages taking different routes through the network. Systems should be configured and components selected to minimize these effects guided by the required timing accuracy. In single subnet systems with distances of a few meters, asymmetry is not usually a concern for time accuracies above a few 10s of ns.

e) If two-step clocks are used, then the network has to be designed such that the general message takes the same path as the event message through a transparent clock. Failure to do this will result in a condition where the transparent clock does not calculate path delay properly. This is a condition that is undetectable and may introduce additional jitter and wander, but it will not break the protocol.

f) PTP assumes that the number of boundary clocks forming the master-slave synchronization hierarchy from the grandmaster clock to any slave clock is less than 255, see 9.3.2.5.

g) Network components, for example bridges, introduce timing jitter and wander that if uncorrected can degrade time transfer accuracy. Since the jitter and wander are often traffic dependent, the network traffic patterns should be designed to minimize the traffic and to minimize the variation in the traffic load. It is also recommended that PTP event messages be sent in high priority compared with other data, see A.5.3.3. Whenever possible such devices should be replaced by PTP boundary or transparent clocks.

h) The network's protocol is structured such that a message timestamp point can be defined.

6.3 PTP systems

A PTP system is a distributed, networked system consisting of a combination of PTP and non-PTP devices. PTP devices include ordinary clocks, boundary clocks, end-to-end transparent clocks, peer-to-peer transparent clocks, and management nodes. Non-PTP devices include bridges, routers and other infrastructure devices, and possibly devices such as computers, printers, and other application devices.

The protocol is a distributed protocol that specifies how the real-time clocks in the system synchronize with each other. These clocks are organized into a master-slave synchronization hierarchy with the clock at the top of the hierarchy — the grandmaster clock — determining the reference time for the entire system. The synchronization is achieved by exchanging PTP timing messages, with the slaves using the timing information to adjust their clocks to the time of their master in the hierarchy.

Devices in a PTP system communicate with each other via a communication network. The network may include translation devices between segments implementing different network communication protocols.

The protocol executes within a logical scope called a domain. Unless otherwise specified, all PTP messages, data sets, state machines and all other PTP entities are always associated with a particular domain. A given physical network and individual devices connected to the network can be associated with multiple domains. Within this standard the time established within one domain by the protocol is independent of the time in other domains.

6.4 PTP message classes

The protocol defines event and general PTP messages. Event messages are timed messages in that an accurate timestamp is generated both at transmission and receipt as specified in 6.6.5. General messages do not require accurate timestamps.

The set of event messages consists of:

a) Sync (see 13.6),

b) Delay_Req (see 13.6),

c) Pdelay_Req (see 13.9), and

d) Pdelay_Resp (see 13.10).

The set of general messages consists of:

a) Announce (see 13.5),

b) Follow_Up (see 13.7),

c) Delay_Resp (see 13.8),

d) Pdelay_Resp_Follow_Up (see 13.11),

e) Management (see Clause 15), and

f) Signaling (see 13.12).

The Sync, Delay_Req, Follow_Up, and Delay_Resp messages are used to generate and communicate the timing information needed to synchronize ordinary and boundary clocks.

The Pdelay_Req, Pdelay_Resp, and Pdelay_Resp_Follow_Up are used to measure the link delay between two clock ports implementing the link delay measurement mechanism, i.e. the peer delay mechanism. The link delay is used to correct timing information in Sync and Follow_Up messages in systems composed of peer-to-peer transparent clocks. Ordinary and boundary clocks that implement the peer delay mechanism can synchronize using the measured link delays and the information in the Sync and Follow_Up messages.

The Announce message is used to establish the synchronization hierarchy.

The management messages are used to query and update the PTP data sets maintained by clocks. These messages are also used to customize a PTP system and for initialization and fault management. Management messages are used between PTP management nodes and PTP Clocks.

The signaling messages are used for communication between clocks for all other purposes. For example, signaling messages can be used for negotiation of the rate of unicast messages between a master and its slaves.

All messages can be extended by means of a standard type, length, value (TLV) extension mechanism. For example, the PATH_TRACE message extensions can be used to detect rogue frames, see 16.2.1 for more detail on rogue frames.

6.5 PTP device types

6.5.1 General

There are five basic types of PTP devices:

a) Ordinary clock,

b) Boundary clock,

c) End-to-end transparent clock,

d) Peer-to-peer transparent clock, and

e) Management node.

All five types implement one or more aspects of the protocol.
There are two mechanisms used in PTP to measure the propagation delay between PTP ports. The first, the delay request-response mechanism, uses the messages Sync, Delay_Req, Delay_Resp, and, if required, Follow_Up, see 11.3. The second, the peer delay mechanism, uses the messages Pdelay_Req, Pdelay_Resp, and, if required, Pdelay_Resp_Follow_Up, see 11.4. Ports on ordinary and boundary clocks can be implemented using either mechanism. Ports on end-to-end transparent clocks are independent of these mechanisms. Ports on peer-to-peer transparent clocks use the peer delay mechanism. These two mechanisms do not interwork on the same communication path. In addition the peer delay mechanism is

restricted to topologies where each peer-to-peer port communicates PTP messages with at most one other such port, see 11.4.4.

As a result the use of the various clock types is limited as follows:

- Ordinary and boundary clocks with ports implementing the peer delay mechanism, and peer-to-peer transparent clocks can only be connected in topologies where ports implementing the peer delay mechanism communicate PTP messages to and from a single port also implementing the peer delay mechanism, see 11.4.4. Except in carefully designed networks, this will preclude the use of end-to-end transparent clock and bridges that do not support PTP using the peer delay mechanism, e.g. a conventional bridge.
- Ordinary and boundary clock ports implementing the delay request-response mechanism, and end-to-end transparent clocks can be connected in any topology that excludes ports using the peer delay mechanism. This precludes the use of peer-to-peer transparent clocks in such as system.
- A boundary clock with ports supporting each of the two mechanisms may be used to bridge between regions supporting the different mechanisms.

6.5.2 Ordinary clocks

The model of an ordinary clock is illustrated in Figure 2.

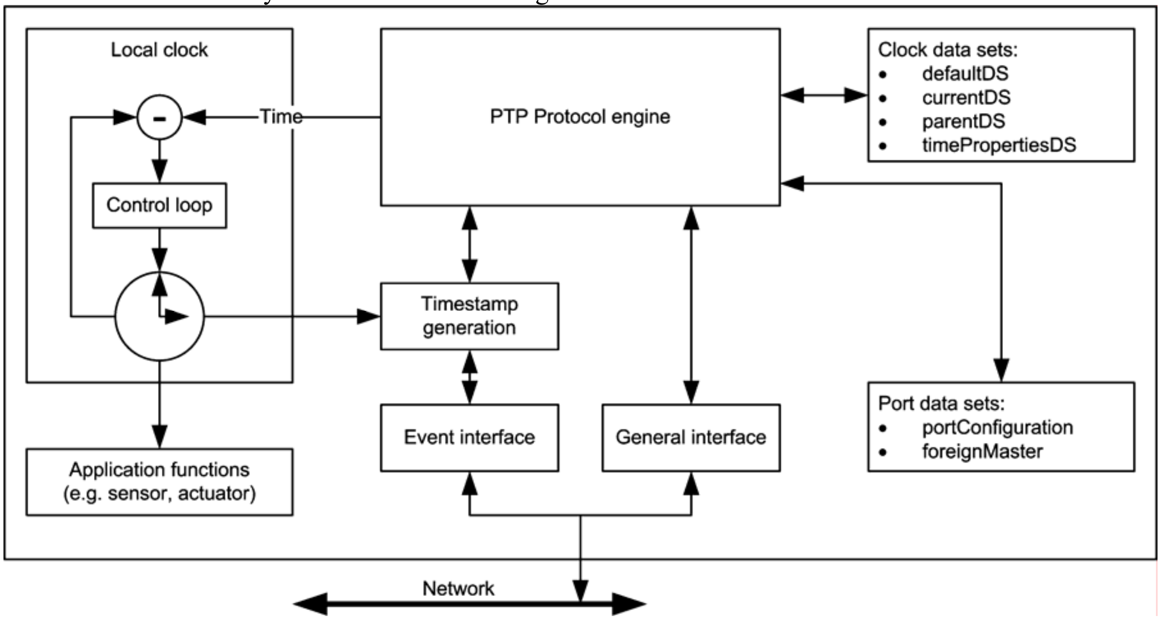


Figure 2: Model of an ordinary clock

An ordinary clock communicates with the network via two logical interfaces based on a single physical port. The event interface is used to send and receive event messages, which are timestamped by the timestamp generation block based on the value of the local clock. The general interface is used to send and receive general messages. An ordinary clock in a domain supports a single copy of the protocol and has a single PTP state. The ordinary clock can be the grandmaster clock in a system or it can be a slave clock in the master-slave hierarchy.

An ordinary clock maintains two types of data sets, referred to as clock data sets and port data sets respectively.

The clock data sets are:

a) defaultDS: Attributes describing the ordinary clock.

b) currentDS: Attributes related to synchronization.

c) parentDS: Attributes describing the parent (the clock to which the ordinary clock synchronizes) and the grandmaster (the clock at the root of the master-slave hierarchy).

d) timePropertiesDS: Attributes of the timescale.

The port data sets are attributes of the port including the PTP state.

The protocol engine:

a) Sends and receives PTP messages.

b) Maintains the data sets.

c) Executes the state machine associated with the port.

d) If the port is in the slave state (synchronizes to a master), it computes the master’s time based on the received PTP timing messages and timestamps that were generated.

The control loop in the local clock adjusts the clock to agree with the time of its master if the ordinary clock port is in the slave state. If the port is in the master state, the local clock is free running or possibly synchronized to an external source of time such as the Global Positioning System (GPS). If the port is in the master state and the ordinary clock is the grandmaster clock of the domain, then the local clock is typically synchronized to an external source of time traceable to TAI and UTC such as the GPS system.

In some applications such as industrial automation, an ordinary clock may also be associated with an application device such as a sensor or actuator. In a telecommunications application an ordinary clock may be associated with a timing demarcation device. The local clock provides time support to such a device as illustrated in Figure 2.

6.5.3 Boundary clocks

The model of a boundary clock is illustrated in Figure 3.

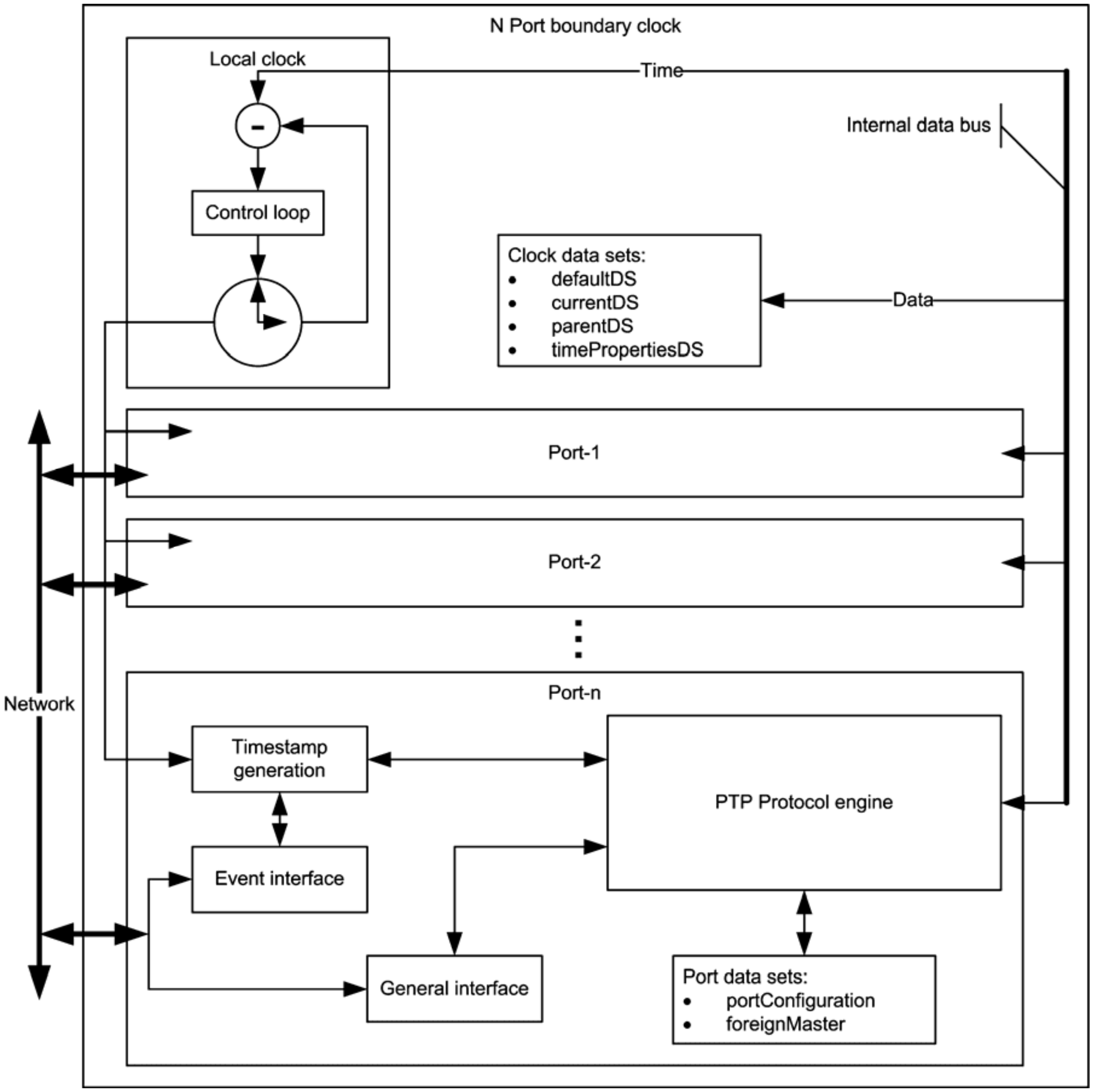


Figure 3: Model of a boundary clock

The boundary clock typically has several physical ports with each physical port communicating with the network via two logical interfaces- event and general. Each port of a boundary clock is like the port of an ordinary clock with the following exceptions:

a) The clock data sets are common to all ports of the boundary clock,

b) The local clock is common to all ports of the boundary clock,

c) Each protocol engine has the additional function of resolving the states of all ports to determine which port provides the time signal used to synchronize the local clock.

The messages related to synchronization, establishing the master-slave hierarchy, and signaling, see 6.4, terminate in the protocol engine of a boundary clock and are not forwarded. Management messages are forwarded to other ports on the boundary clock subject to restrictions to limit the propagation of these messages within the system.

The boundary clock model of Figure 3 is applicable only to PTP messages. For all non-PTP messages the boundary clock behaves as a normal network component, e.g. bridge, repeater, or router.

A boundary clock is typically used only as a network element and is not normally associated with application devices such as sensors or actuators.

6.5.4 End-to-end transparent clocks

The model of an end-to-end transparent clock is illustrated in Figure 4.

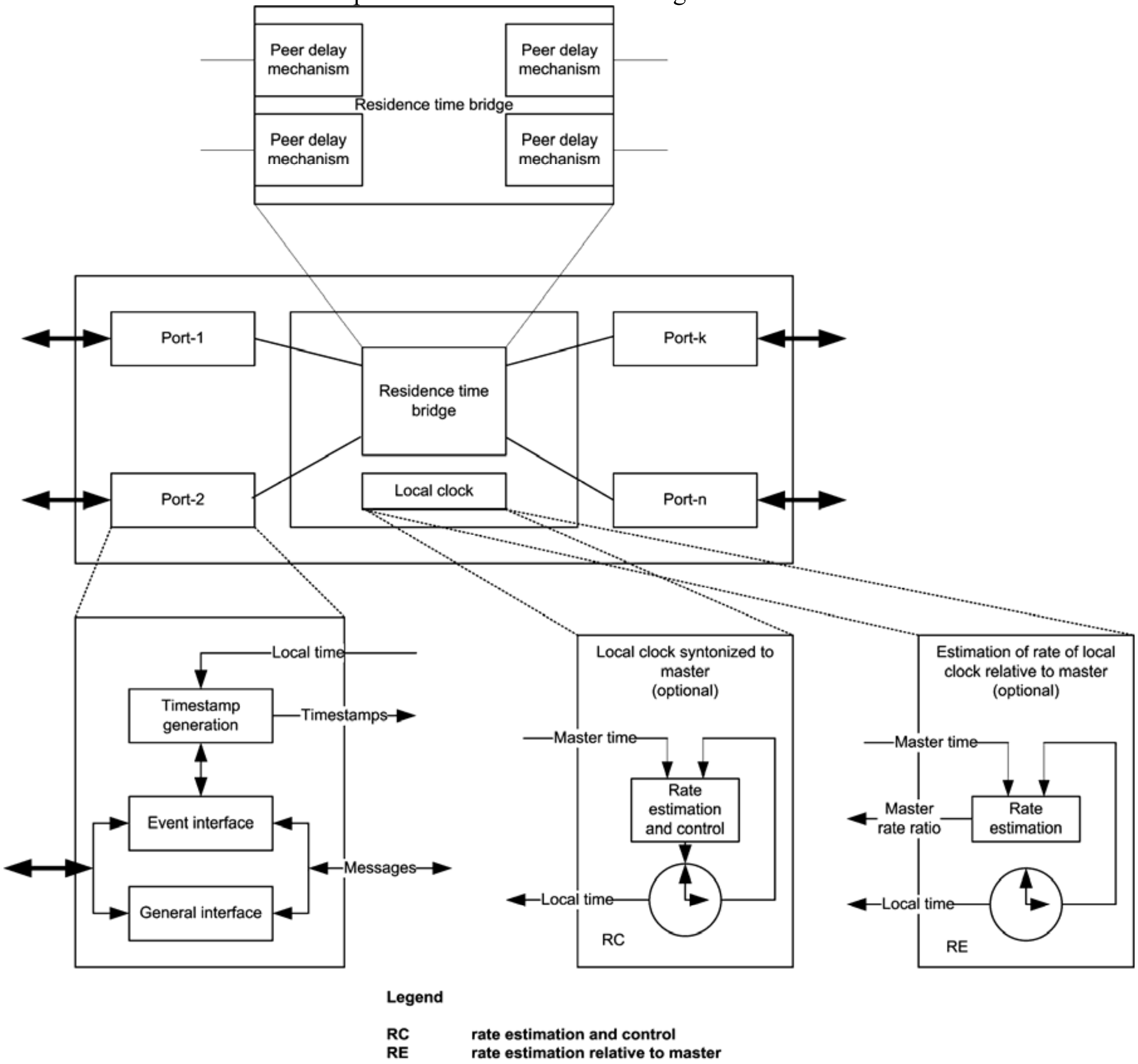


Figure 4: Model of an end-to-end transparent clock

The end-to-end transparent clock forwards all messages just as a normal bridge, router, or repeater. However for PTP event messages, the residence time bridge, shown in Figure 4, measures the residence time of PTP event messages (the time the message takes to traverse the transparent clock). These residence times are accumulated in a special field, the correction field, of the PTP event message or the associated follow up message (Follow_Up or Pdelay_Resp_Follow_Up). This correction is based on the difference in the timestamp generated when the event message enters and leaves the transparent clock. Any updates to checksums required by the network protocol are made. Note that the value of the correction update and checksums are specific to each output port and message since the residence times are not necessarily the same for all paths through the transparent clock or for successive messages on the same path. The correction process is illustrated for an arbitrary pair of ports in Figure 5.

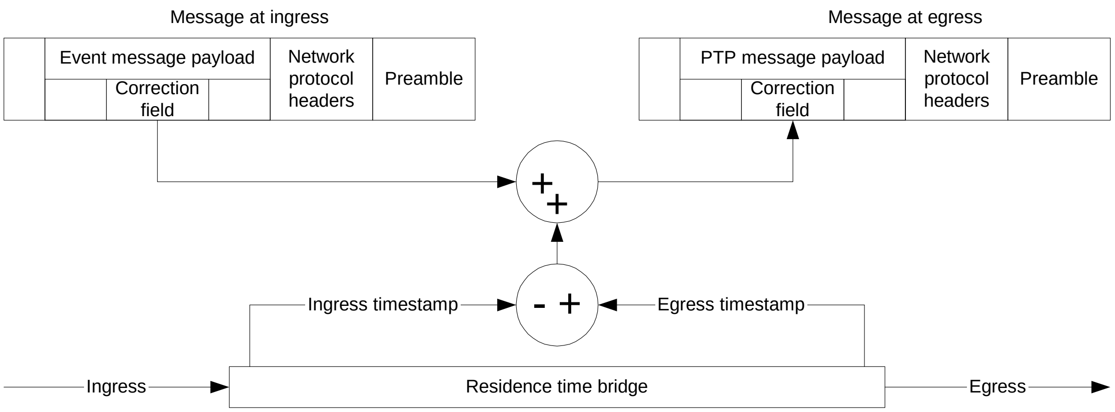


Figure 5: End-to-end residence time correction model

The timestamps used in computing the residence time are based on timestamps generated from the local clock. Since these accumulated residence times are used by a slave to adjust the time provided by a master, it is important that any errors resulting from differences in the rates of the master and the transparent clocks be negligible for the accuracy required by the application. Since it is possible that the rates of the master and local clocks (essentially the definition of the second on the two clocks) can differ by 0.02 %, the error introduced can be 0.02 % of the measured residence time. Thus for a residence time of 1 ms the maximum error is 200 ns. This error may be unacceptably large, and there are several possible ways of reducing it. One way is to make the rate of the local clock equal to that of the master, i.e., syntonize the local clock to the master, by observing the timing information in received Sync and, if present, Follow_Up messages, corrected for any upstream residence times. This is illustrated in Figure 4 in block RC (rate control). The master time, corrected for any upstream residence times, is input to the Rate estimation and control block in RC (shown by the arrow labeled 'Master time''). The corresponding local time is also input to this block. The Rate estimation and control block uses the sequences of master and corresponding local times to estimate the ratio of the master and local clock rates. The Rate estimation and control block then uses this estimated rate ratio to adjust (i.e., control) the local clock rate. Note that the adjustment of the local clock rate need not be a physical changing of the oscillator frequency (i.e., an analog implementation); it equally acceptable to use a fixed frequency local oscillator, compute the ratio of the master and local rates, and multiply the timestamps taken with the local clock by this ratio (i.e., a digital implementation). The key aspect of the scheme illustrated by RC is that it operates closed loop, i.e., the local times used in the rate estimation and residence time measurement are relative to the rate-adjusted local oscillator. This means that rate adjustments to an oscillator at one node influence the adjustments at downstream nodes, because the residence time is used to adjust the master times at downstream nodes.

A second way to reduce the effect of residence time error is to, as before, use the sequence of master times, corrected for upstream residence times, and local times to estimate the ratio of the master and local clock rates. This is illustrated in the block RE (rate estimation) in Figure 4. However, in this case the frequency of the local oscillator is not adjusted, and therefore the residence times accumulated in the Sync or Follow_Up correction fields, which are used in obtaining the corrected master times, are based on the free-running oscillator frequencies. The ratio of the master to local clock rate is used to compute a residence time correction, which must be accumulated separately. This accumulated residence time correction is used by a slave clock as a correction when the slave computes its offset from the master; however, it is not used in computing the ratio of the master to local clock rate. The key aspect of the scheme illustrated by RE is that it operates open loop, i.e., the local times used in the rate estimation are local times relative to the free-running oscillator. The computed rate ratio at a node does not influence the rate ratios computed at downstream nodes, because the residence time relative to the free-running oscillator is used to obtain the corrected master times at downstream nodes. The residence time correction, based on the rate ratio, is used only to obtain offset at a slave clock.

A master connected to several slaves via end-to-end transparent clocks is required to process the Delay_Req messages from all such slaves since these messages are forwarded by the end-to-end transparent clock.

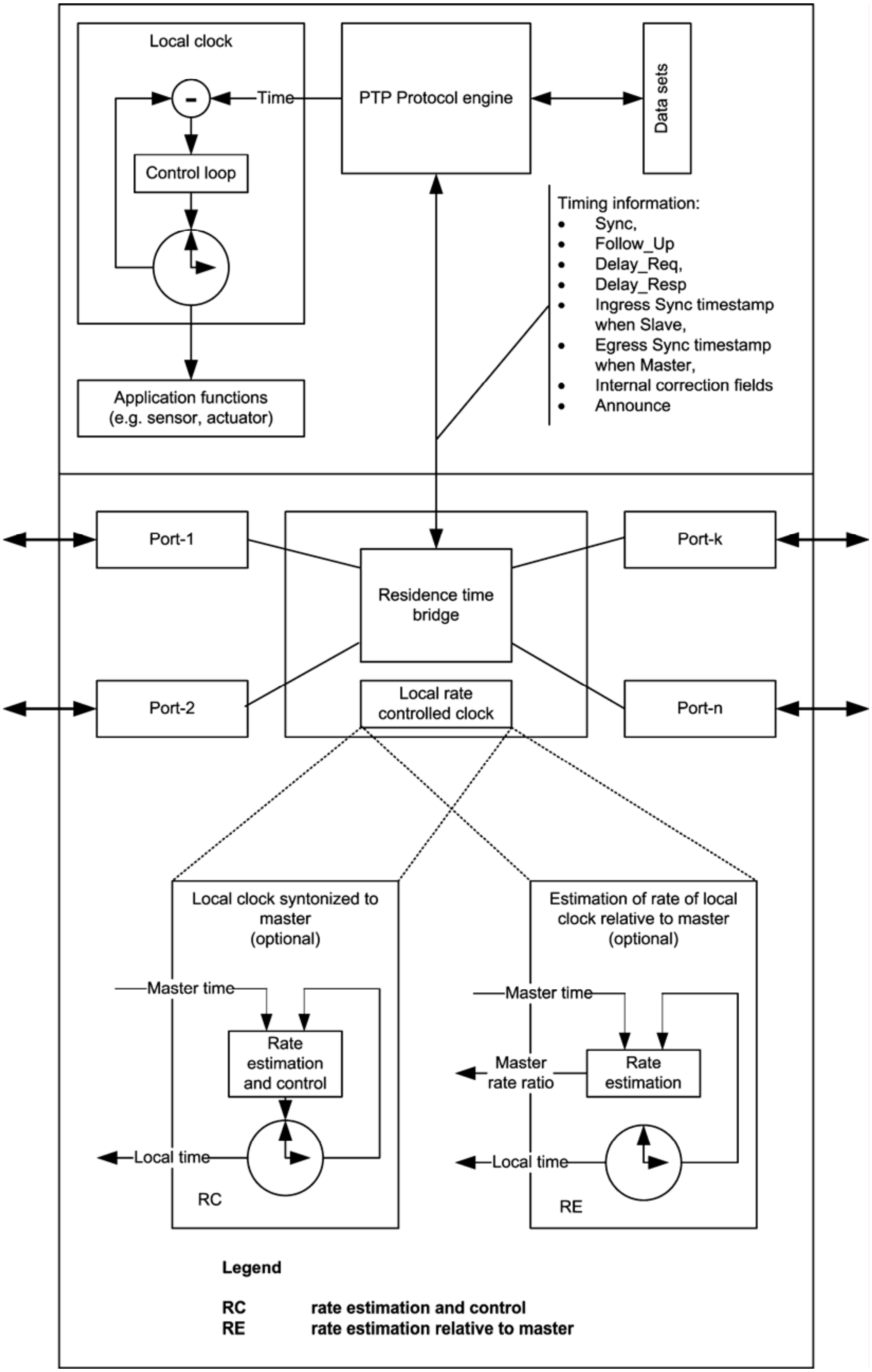

Local clock
-
Time
PTP Protocol engine
Data sets
Control loop
Timing information:
• Sync,
• Follow_Up
• Delay_Req,
• Delay_Resp
• Ingress Sync timestamp when Slave,
• Egress Sync timestamp when Master,
• Internal correction fields
• Announce
Application functions (e.g. sensor, actuator)
Port-1
Port-k
Residence time bridge
Port-2
Local rate controlled clock
Port-n
Local clock syntonized to master (optional)
Estimation of rate of local clock relative to master (optional)
Master time
Rate estimation and control
Local time
RC
Master time
Master rate ratio
Rate estimation
Local time
RE
Legend
RC rate estimation and control
RE rate estimation relative to master

Figure 6: Combined ordinary and end-to-end transparent clock

An end-to-end transparent clock may be used as a network element or it may be associated with application devices such as sensors or actuators. In the latter case, an ordinary clock is combined with the transparent clock to provide real-time support for the application device. A model of such a combined device is illustrated in Figure 6.

In Figure 6, when the ordinary clock is a slave, the timing information includes the ingress timestamp. When it is a master it includes the egress timestamp. The residence time bridge delivers the incoming PTP timing messages, the Announce messages, the ingress timestamp generated by the incoming Sync message, and any internal timing corrections to the protocol engine of the ordinary clock. The protocol engine computes the correct time based on this information and sends it as input to the local clock. If the ordinary clock were a master then it would originate Sync and Follow_Up messages with the sending timestamps referenced to the local clock of the ordinary clock and based on internal timing corrections and the egress timestamp. In practice the end-to-end transparent and ordinary clock functions would share a common physical clock.

6.5.5 Peer-to-peer transparent clocks

The model of a peer-to-peer transparent clock is illustrated in Figure 7.

The peer-to-peer transparent clock differs from the end-to-end transparent clock in the way it corrects and handles the PTP timing messages. In all other respects it is identical with the end-to-end transparent clock.

The peer-to-peer transparent clock has an additional per port block as indicated in Figure 7. This block is used to compute the link delay between each port and a similarly equipped port on another node sharing the link, i.e. the link peer. The link peer exists in another clock supporting the peer delay mechanism since non-peer-to-peer devices are not expected between peer-to-peer transparent clocks. The computation of link delay is based on an exchange of Pdelay_Req, Pdelay_Resp, and possibly Pdelay_Resp_Follow_Up messages with the link peer. As a result of these exchanges the link delay is known for each port of the peer-to-peer transparent clock. Unlike the end-to-end transparent clock, which corrects and forwards all PTP timing messages, the peer-to-peer transparent clock only corrects and forwards Sync and Follow_Up messages. The appropriate correction field in these messages is updated for both the residence time of the Sync message within the peer-to-peer transparent clock and for the link delay on the port receiving the Sync message.

This correction process is illustrated in Figure 8. The egress PTP timing message contains the residence time and link delay corrections for the actual ingress link traversed by a Sync packet. A change in Sync packet routing through a system of peer-to-peer clocks may result in the Sync packet entering the peer-to-peer transparent clock from a different link. Since the peer-to-peer transparent clock corrects the residence and path times based on the actual link and internal path taken by each Sync packet, the egress timing information is always correct to within the accuracy of the measurements. Therefore the timing information provided to the slave always reflects the actual path through the network of peer-to-peer transparent clocks. By contrast, in the case of the end-to-end corrections, the slave waits for a new path delay value based on the combined Sync and Delay_Resp messages, which takes more time since the messages need to traverse the entire new path before correct values are present at the slave.

The measurement of path delay using the peer delay mechanism does not interwork with path delay measurements based on the delay request-response mechanism. As a result a peer-to-peer transparent clock can only work with clock ports supporting the peer delay mechanism. With peer-to-peer transparent clocks the master only needs to issue Sync and Follow_Up messages and respond to Pdelay_Req messages. It does not receive Delay_Req messages and therefore the processing workload is independent of the number of peer-to-peer clocks served by a given port. Likewise, end-to-end transparent clocks support only the delay request-response mechanism and not the peer delay mechanism. If a network contains Peer-to-Peer transparent clocks in one region and end-to-end transparent clocks in another region, the regions can be connected only through a boundary clock.

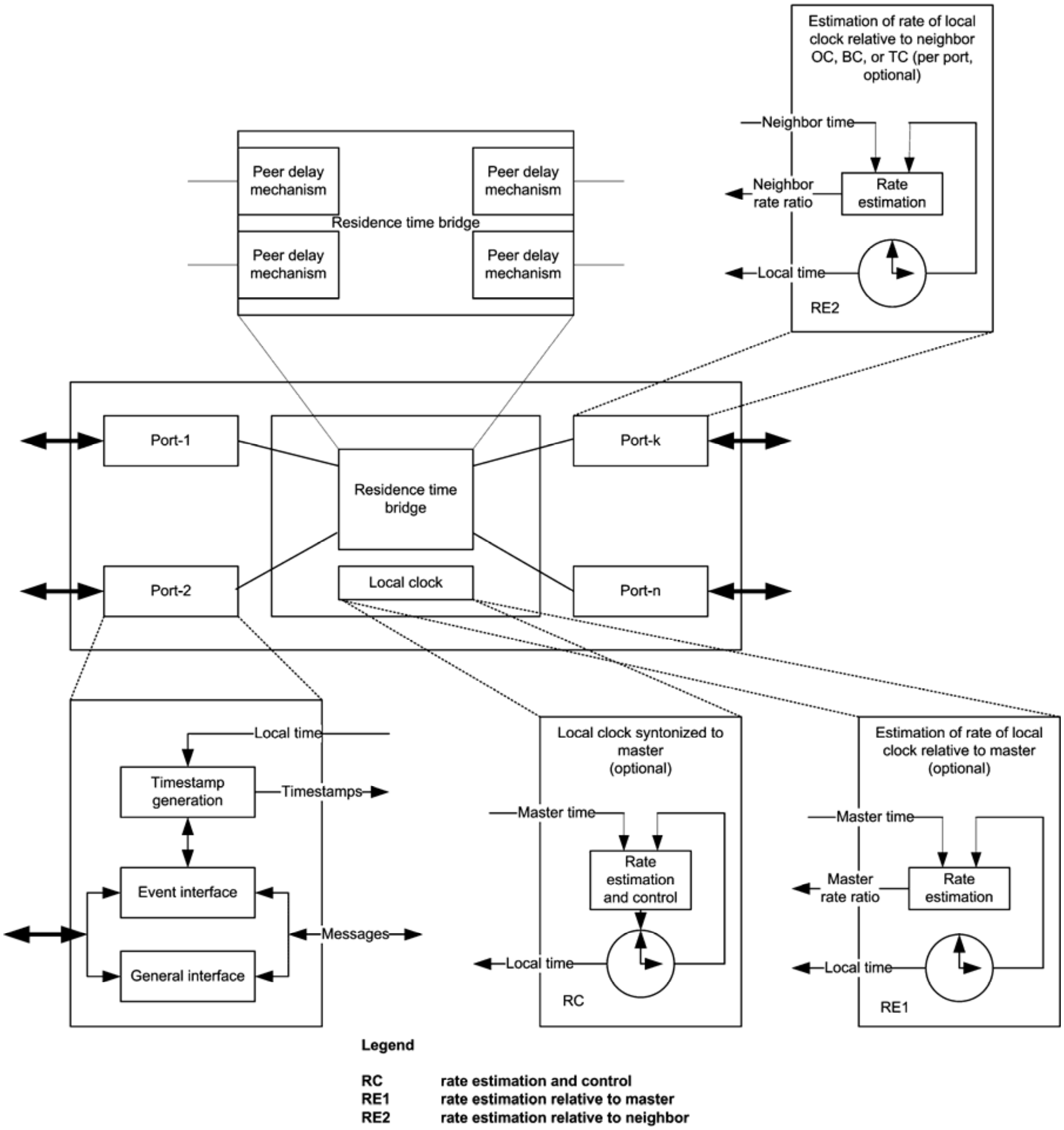


Figure 7: Model of a peer-to-peer transparent clock

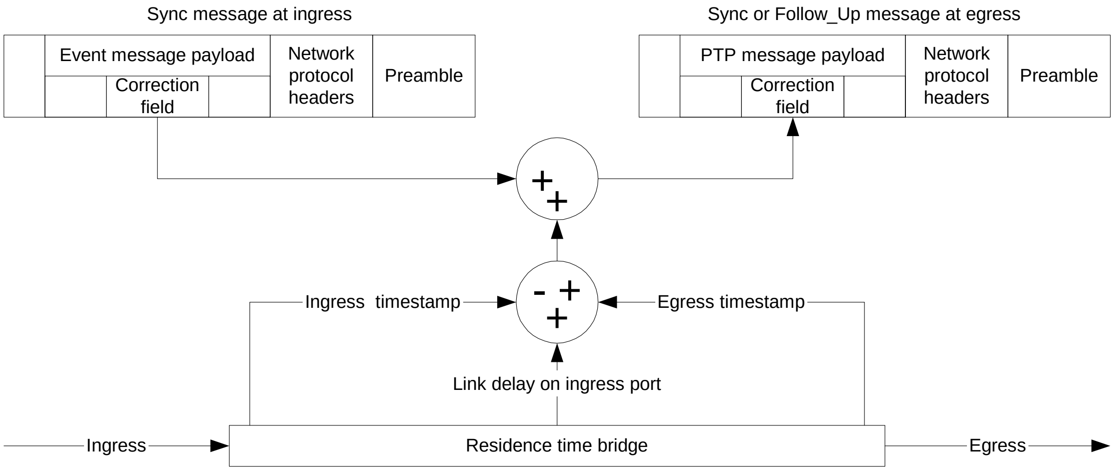


Figure 8: Peer-to-peer residence time and link delay correction model

The timestamps used in computing the residence and link delay times are based on timestamps generated from the local clocks. Since these accumulated residence and link delay times are used by a slave to adjust the time provided by a master, it is important that any errors resulting from differences in the rates of the master and the transparent clocks be negligible for the accuracy required by the application. Since it is possible that the rates of the master and local clocks (essentially the definition of the second on the two clocks) can differ by 0.02 %, the error introduced can be 0.02 % of the measured residence time. Thus for a residence time of 1 ms the maximum error is 200 ns. This error may be unacceptably large, and there are several possible ways of reducing it. One way is to make the rate of the local clock equal to that of the master, i.e., syntonize the local clock to the master, by observing the timing information in received Sync and, if present, Follow_Up messages, corrected for any upstream residence and link delay times. This is illustrated in Figure 4 in block RC (rate control). The master time, corrected for any upstream residence and link delay times, is input to the Rate estimation and control block in RC (shown by the arrow labeled ‘Master time’’). The corresponding local time is also input to this block. The Rate estimation and control block uses the sequences of master and corresponding local times to estimate the ratio of the master and local clock rates. The Rate estimation and control block then uses this estimated rate ratio to adjust (i.e., control) the local clock rate. Note that the adjustment of the local clock rate need not be a physical changing of the oscillator frequency (i.e., an analog implementation); it equally acceptable to use a fixed frequency local oscillator, compute the ratio of the master and local rates, and multiply the timestamps taken with the local clock by this ratio (i.e., a digital implementation). The key aspect of the scheme illustrated by RC is that it operates closed loop, i.e., the local times used in the rate estimation and residence time measurement are relative to the rate-adjusted local oscillator. This means that rate adjustments to an oscillator at one node influence the adjustments at downstream nodes, because the residence time is used to adjust the master times at downstream nodes.

A second way to reduce the effect of residence time error is to, as before, use the sequence of master times, corrected for upstream residence and link delay times, and local times to estimate the ratio of the master and local clock rates. This is illustrated in the block RE1 (master rate ratio estimation) in Figure 7. However, in this case the frequency of the local oscillator is not adjusted, and therefore the residence times accumulated in the Sync or Follow_Up correction fields, which are used in obtaining the corrected master times, are based on the free-running oscillator frequencies. The ratio of the master to local clock rate is used to compute a residence time correction, which must be accumulated separately. This accumulated residence time correction is used by a slave clock as a correction when the slave computes its offset from the master; however, it is not used in computing the ratio of the master to local clock rate. The key aspect of the scheme illustrated by RE is that it operates open loop, i.e., the local times used in the rate estimation are local times relative to the free-running oscillator. The computed rate ratio at a node does not influence the rate ratios computed at downstream nodes, because the residence time relative to the free-running oscillator is used to obtain the corrected master times at downstream nodes. The residence time correction, based on the rate ratio, is used only to obtain offset at a slave clock.

The correction of master times by a peer-to-peer transparent clock requires the link delay times as well as the residence times. The link delays are measured using the peer delay mechanism, (see 6.6.4). If the peer-to-peer transparent clocks are syntonized to the master as showin in block RC, the peer delay measurement is also done using syntonized clocks. However, if the rate of the transparent clock relative to the master is measured and used as in block RE1, the peer delay measurement is done using the free-running local clocks. In this case, there is an error in the link delay equal to the elapsed time between the receipt of the Pdelay_Req message and the sending of the Pdelay_Resp message, t3–t2 (see 6.6.4 and Figure 13) multiplied by the fractional frequency offset between the transparent clock and its neighbor. To eliminate this component of error in link delay, the frequency offset between the transparent clock and its neighbor can be measured using the timestamp information contained in successive Pdelay_Resp, and possibly Pdelay_Resp_Follow_Up messages. This is illustrated in the block RE2 (neighbor rate ratio estimation) in Figure 7. Note that to make this measurement, it is necessary to transmit the Pdelay_Req receipt time (t2) and Pdelay_Resp transmit time (t3) separately (see 11.4.2).

A peer-to-peer transparent clock may be used as a network element or it may be associated with application devices such as sensors or actuators. In the latter case, an ordinary clock is combined with the transparent clock to provide real-time support for the application device. A model of such a combined device is illustrated in Figure 9.

In Figure 9 when the ordinary clock is a slave the timing information includes the ingress timestamp. When it is a master it includes the egress timestamp. When the clock is a slave, the residence time bridge delivers the incoming PTP timing messages, the Announce messages, and the ingress timestamp generated by the incoming Sync message, and any internal timing corrections to the protocol engine of the ordinary clock. The protocol engine computes the correct time based on these messages and sends it as input to the local clock. If the ordinary clock were a master then it would originate Sync and Follow_Up messages with the sending timestamps referenced to the local clock of the ordinary clock based on internal timing corrections and the egress timestamp. In practice the peer-to-peer transparent and ordinary clock functions would share a common physical clock.

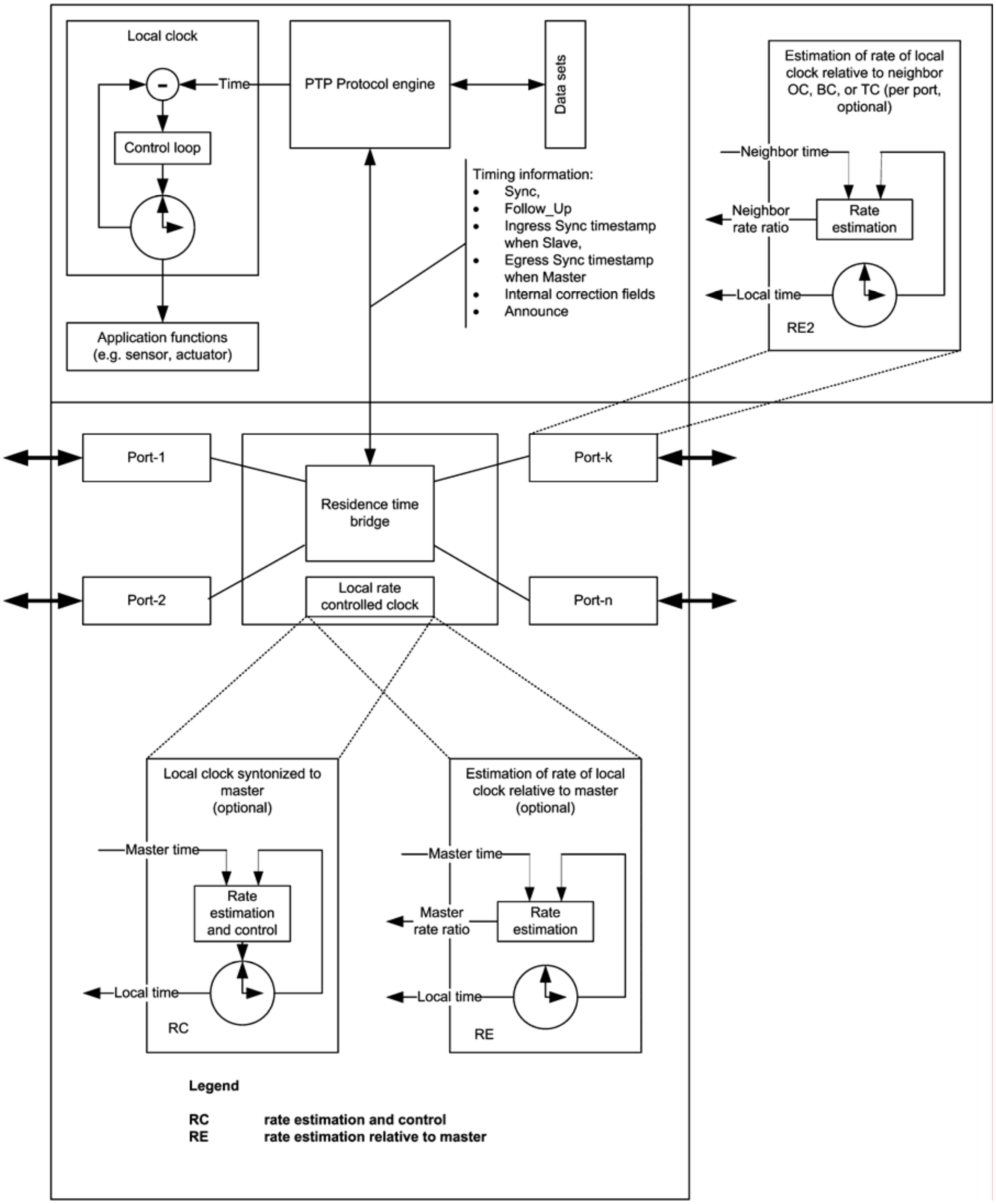

Local clock
-
Time
Control loop
PTP Protocol engine
Data sets
Timing information:
• Sync,
• Follow_Up
• Ingress Sync timestamp when Slave,
• Egress Sync timestamp when Master
• Internal correction fields
• Announce
Application functions (e.g. sensor, actuator)
Estimation of rate of local clock relative to neighbor OC, BC, or TC (per port, optional)
Neighbor time
Neighbor rate ratio
Rate estimation
Local time
RE2
Port-1
Port-2
Residence time bridge
Local rate controlled clock
Port-k
Port-n
Local clock syntonized to master (optional)
Master time
Rate estimation and control
Local time
RC
Estimation of rate of local clock relative to master (optional)
Master time
Master rate ratio
Rate estimation
Local time
RE
Legend
RC rate estimation and control
RE rate estimation relative to master

Figure 9: Combined ordinary and peer-to-peer transparent clock.

6.5.6 Management nodes

A management node is a PTP device that:

a) Has one or more physical connections to the network,

b) Serves as an human or programmatic interface to PTP management messages,

c) May be combined with any of the clock types.

6.6 Synchronization overview

6.6.1 General

There are two phases in the normal execution of the protocol:

a) Establishing the master-slave hierarchy, and

b) Synchronizing the clocks.

6.6.2 Establishing the master-slave hierarchy

6.6.2.1 General

Within a domain, each port of an ordinary and boundary clock executes an independent copy of the protocol state machine. For “state decision events” each port examines the contents of all Announce messages received on the port. Using the best master clock algorithm, the Announce message contents and the contents of the data sets associated with the clock are analyzed to determine the state of each port in a boundary clock or ordinary clock.

6.6.2.2 PTP state machine

Each port of an ordinary and boundary clock maintains a separate copy of the PTP state machine. This state machine defines the allowed states of the port and the transition rules between states. The principal “state decision events” in determining the master-slave hierarchy are the receipt of an Announce message and the end of an announceInterval (the interval between Announce messages). The port states determining the master-slave hierarchy are:

a) MASTER: The port is the source of time on the path served by the port.

b) SLAVE: The port synchronizes to the device on the path with the port that is in the MASTER state.

c) PASSIVE: The port is not the master on the path nor does it synchronize to a master.

6.6.2.3 Best master clock algorithm

The best master clock algorithm compares data describing two clocks to determine which data describes the better clock. This algorithm is used to determine which of the clocks described in several Announce messages received by a local clock port is the best clock. It is also used to determine whether a newly discovered clock — a foreign master — is better than the local clock itself. The data describing a foreign master is contained in grandmaster fields of an Announce message. The data describing the local clock is contained in the defaultDS data set of the clock.

The best master clock algorithm is composed of two separate algorithms:

a) Data set comparison algorithm.

b) State decision algorithm.

The data set comparison algorithm is based on pair wise comparisons of attributes with the following precedence:

a) priority1: This is a user configurable designation that a clock belongs to an ordered set of clocks from which a master is selected.

b) clockClass: An attribute defining a clock’s TAI traceability.

c) clockAccuracy: An attribute defining the accuracy of a clock.

d) offsetScaledLogVariance: An attribute defining the stability of a clock.

e) priority2: This is a user configurable designation that provides finer grained ordering among otherwise equivalent clocks.

f) clockIdentity: A tie-breaker based on unique identifiers.

In addition to this precedence order, the “distance” measured by the number of boundary clocks between the local clock and the foreign master is used when two Announce messages reflect the same foreign master. The distance is indicated in the stepsRemoved field of Announce messages. This reflection can occur in PTP systems with cyclic paths not removed by a protocol outside of PTP. The data set comparison algorithm unambiguously selects one of the two clocks as “better” or as “topologically better. ”

The state decision algorithm determines whether the next state of the port — the recommended state — will be MASTER, SLAVE, or PASSIVE based on the results of the data set comparison algorithm and whether the local clock’s class is less than 128, see 7.6.2.4. This recommended state is then evaluated by the port protocol engine based on the current state of the protocol state machine to determine the actual next port state.

6.6.2.4 Simple master-slave hierarchy

The process establishing a master-slave hierarchy among the ordinary and boundary clocks in a domain as illustrated in Figure 10.

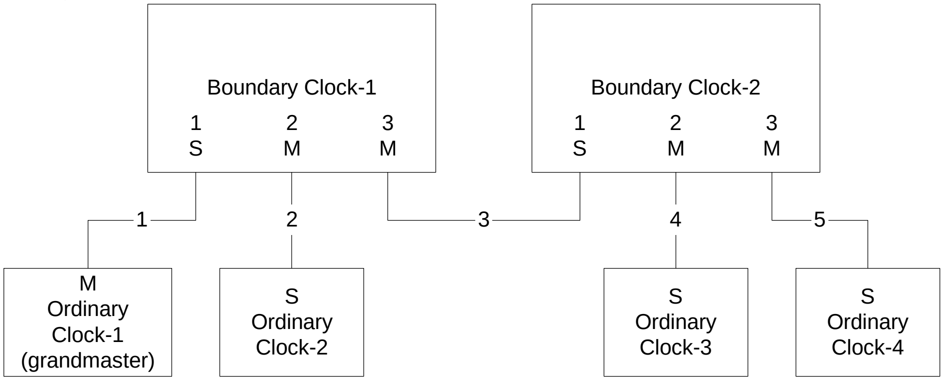


Figure 10: Simple master-slave clock hierarchy

In this example, ordinary clock-1 is at the root of the hierarchy and is called the grandmaster clock. Port-1 of boundary clock-1 is a slave (as indicated by the S) to the grandmaster clock. All other ports on boundary clock-1 are masters to the clocks connected to them. Thus port-1 of boundary clock-2 is a slave to boundary clock-1 and so forth. Only ordinary and boundary clocks maintain this form of state and only boundary clocks establish the branch points in the master-slave hierarchy (i.e., paths 1, 2, 3, 4, and 5 may contain transparent clocks, but these clocks do not participate in the master/slave hierarchy and do not maintain this form of state).

6.6.2.5 Pruning mesh topologies

Figure 11 illustrates the case where a mesh network is reduced to a tree structured master-slave hierarchy by the protocol. This occurs when the underlying bridging or routing protocols do not eliminate cyclic paths. In Figure 11 ordinary clock-1 is assumed to have been selected as the grandmaster clock by the best master clock algorithm. In the boundary clocks the port states have been selected by the best master clock algorithm as shown to construct the tree. The pruned paths are shown in dashed lines. For each boundary clock one port is selected by the best master clock algorithm as the slave port. The other ports are set to either the master or passive state. The best master clock algorithm ensures that a single master port is selected on each segment.

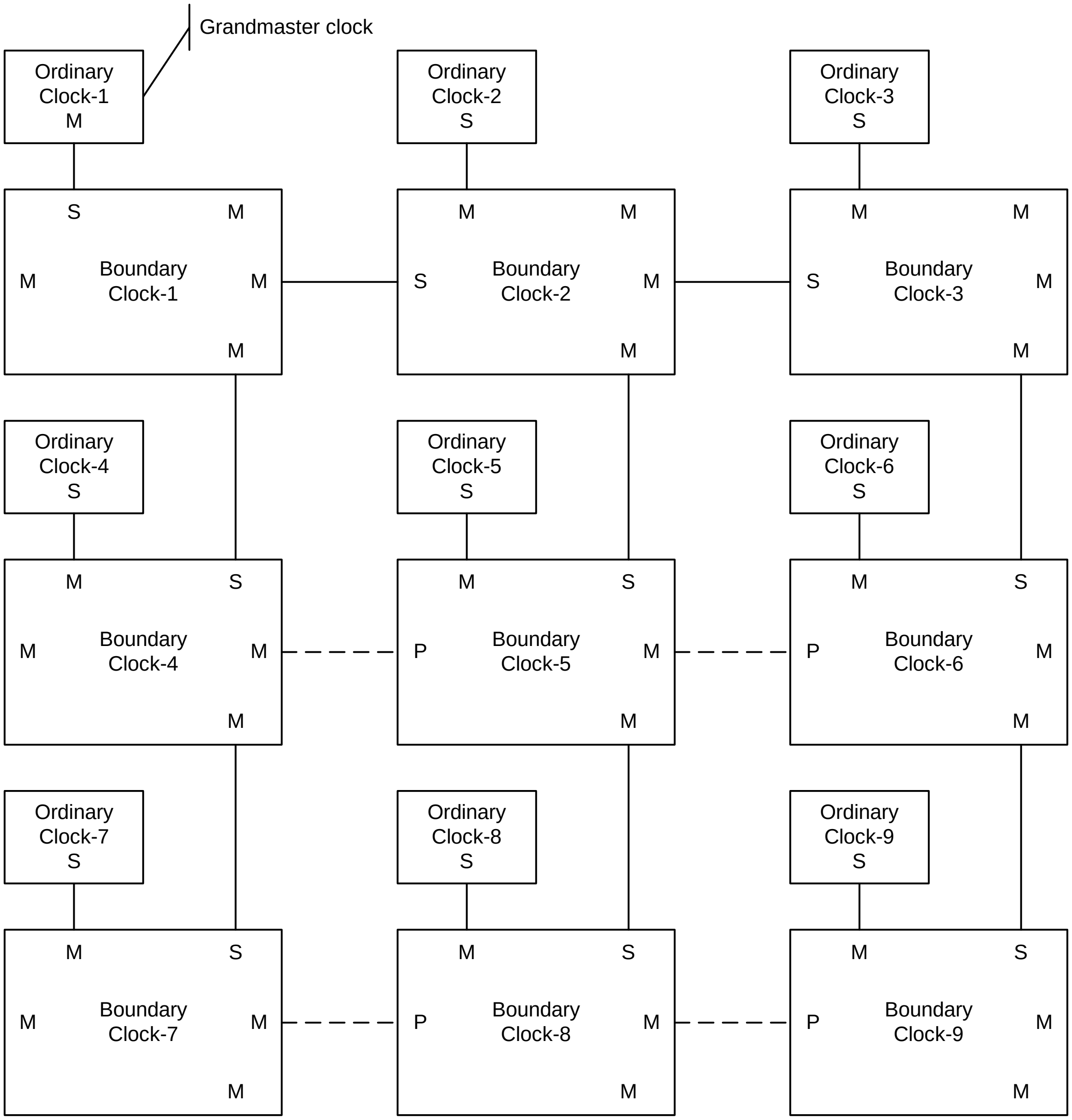

Grandmaster clock
Ordinary Clock-1 M
Ordinary Clock-2 S
Ordinary Clock-3 S
S M M Boundary Clock-1 M M
M M S Boundary Clock-2 M M
M M S Boundary Clock-3 M M
Ordinary Clock-4 S
Ordinary Clock-5 S
Ordinary Clock-6 S
M S M Boundary Clock-4 M M
M S P Boundary Clock-5 M M
M S P Boundary Clock-6 M M
Ordinary Clock-7 S
Ordinary Clock-8 S
Ordinary Clock-9 S
M S M Boundary Clock-7 M M
M S P Boundary Clock-8 M M
M S P Boundary Clock-9 M M

Figure 11: Pruned mesh topology

6.6.3 Synchronizing ordinary and boundary clocks

In a PTP system, the real-time clocks in ordinary or boundary clocks synchronize by exchanging PTP timing messages on the communication path linking the two clocks. For example, in Figure 10 the real-time clock in boundary clock-1 synchronizes to the real-time clock in ordinary clock-1 by exchanging messages on communication path-1.

The basic pattern of synchronization message exchange is illustrated in Figure 12.

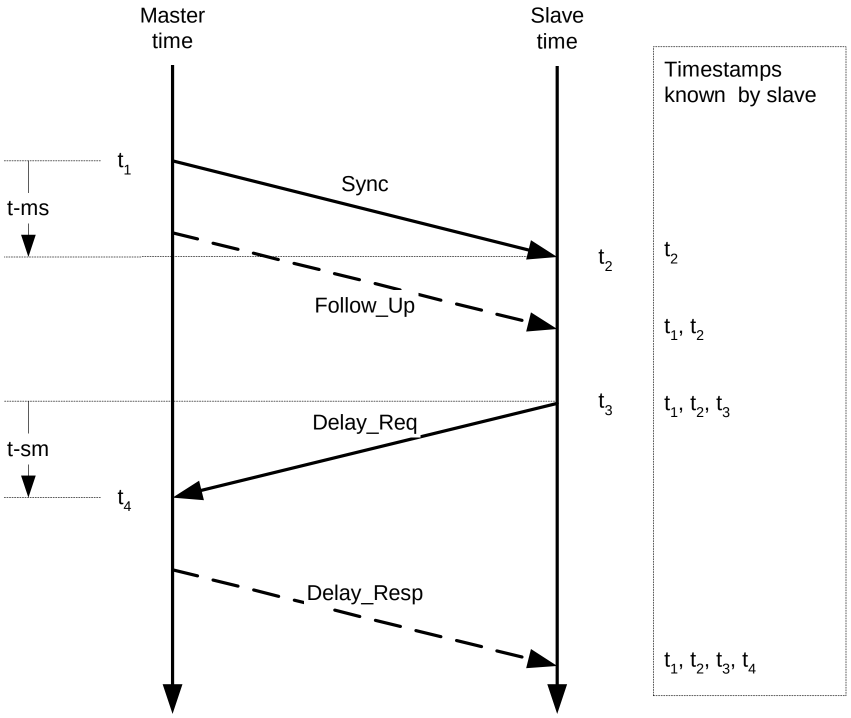


Figure 12: Basic synchronization message exchange

The message exchange pattern is:

a) The master sends a Sync message to the slave and notes the time, t_1, at which it was sent.

b) The slave receives the Sync message and notes the time of reception, t_2.

c) The master conveys to the slave the timestamp t_1 by:

   1) Embedding the timestamp t_1 in the Sync message. This requires some sort of hardware processing for highest accuracy and precision, or

   2) Embedding the timestamp t_1 in a Follow_Up message.

d) The slave sends a Delay_Req message to the master and notes the time, t_3, at which it was sent.

e) The master receives the Delay_Req message and notes the time of reception, t_4.

f) The master conveys to the slave the timestamp t_4 by embedding it in a Delay_Resp message.

At the conclusion of this exchange of messages, the slave possesses all four timestamps. These timestamps may be used to compute the offset of the slave's clock with respect to the master and the mean propagation time of messages between the two clocks, which in Figure 12 is the mean of t-ms and t-sm.

The computation of offset and propagation time assumes that the master-to-slave and slave-to-master propagation times are equal. Any asymmetry in propagation time introduces an error in the computed value of the clock offset. The computed mean propagation time differs from the actual propagation times due to the asymmetry.

6.6.4 Measuring link propagation delay in clocks supporting peer-to-peer path correction.

The mechanism for measuring the link delay between two ports that implement the peer delay mechanism (see 6.5.5) is illustrated in Figure 13. This measurement is conducted by all ports implementing the mechanism. Thus both ports sharing a link independently make the measurement and as a result both ports know the link delay. This allows the corrections outlined in 6.5.5 to be made irrespective of the direction taken by a Sync message. It is important that this measurement occur even on ports otherwise blocked by non-PTP algorithms used to eliminate cyclic topologies, so that up-to-date link delay measurements are available for all links in the event of a change in the path taken by a Sync message (see 6.5.5).

The link delay measurement starts with port-1 issuing and generating a timestamp, t_1, for a Pdelay_Req message. Port-2 receives and generates a timestamp, t_2, for this message. Port-2 returns and generates a timestamp, t_3, for a Pdelay_Resp message. To minimize errors due to any frequency offset between the two ports, port-2 returns the Pdelay_Resp message as quickly as possible after the receipt of the Pdelay_Req message.

Port-2 either:

1. returns the difference between the timestamps t_2 and t_3 in the Pdelay_Resp message,
2. returns the difference between the timestamps t_2 and t_3 in a Pdelay_Resp_Follow_Up message, or
3. returns the timestamps t_2 and t_3 in the Pdelay_Resp and Pdelay_Resp_Follow_Up messages respectively.

Port-1 generates a timestamp, t_4, upon receiving the Pdelay_Resp message. Port-1 then uses these four timestamps to compute the mean link delay.

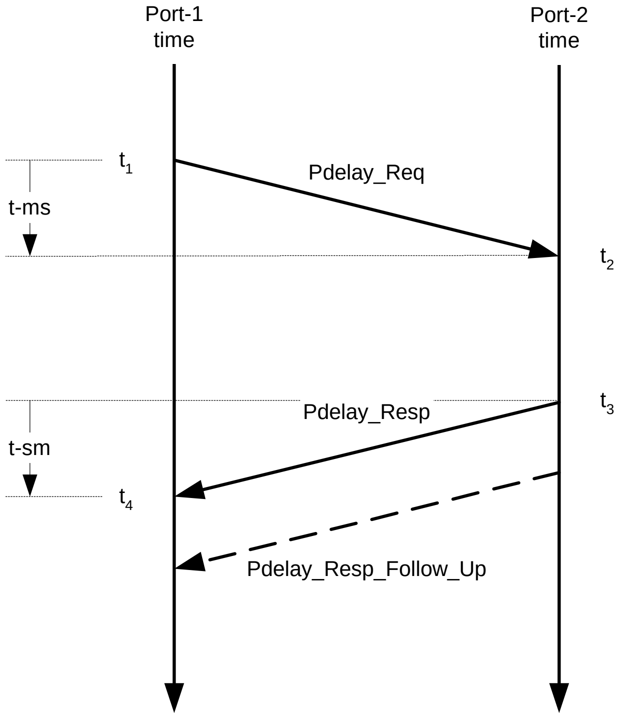


Figure 13: Link delay measurement

Any asymmetry in the propagation times t-ms and t-sm introduces an error into the computed value of the link delay. If the mechanism producing the timestamps t_1, t_2, t_3, and t_4 does not share the same definition of a second as the grandmaster clock issuing the Sync messages, a small but possibly significant error is introduced in the link delay measurement. The source of this error is a small difference between the frequency of the oscillators in the grandmaster and peer-to-peer clocks. In addition, if there is a difference between the frequency of the oscillators in the two peer-to-peer clocks , there is a small but possibly significant error due to the turnaround time, t_3─t_2, in the peer-to-peer clock. If this error is significant then the oscillator in the peer-to-peer clock can be frequency synchronized, i.e. syntonized, to that of the grandmaster, or the error can be computed and corrected based on the evaluated frequency difference between the two oscillators.

6.6.5 Generation of message timestamps

A timestamp event is generated at the time of transmission and reception of any event message. The timestamp event occurs when the message's timestamp point crosses the boundary between the node and the network.

The generation of the timestamps, indicated in Figure 12 and Figure 13, is modeled in Figure 14.

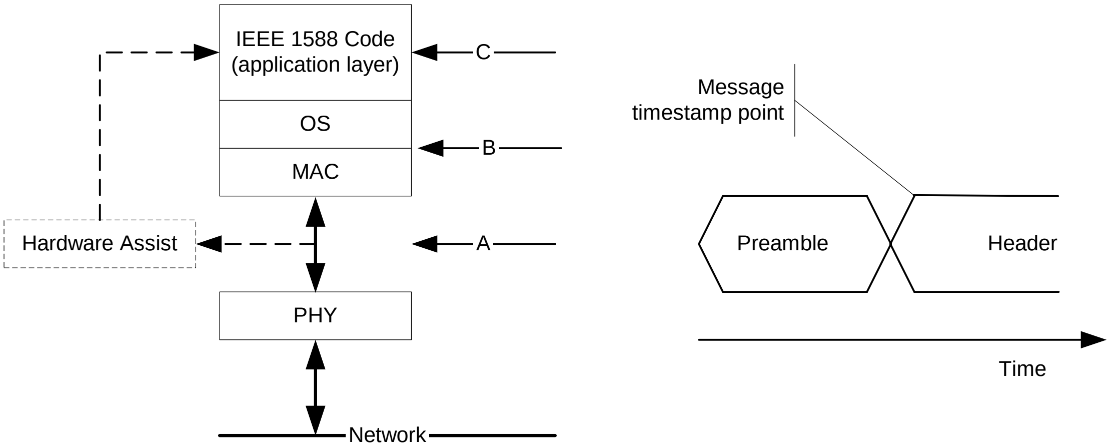


Figure 14: Timestamp generation model

PTP timing messages are issued from the PTP application code in one clock and received and processed by the PTP application code in another clock. These messages typically have a preamble specified by the physical layer of the communication protocol in use on the network. The preamble is followed by one or more protocol specific headers and then user data such as the PTP payload. For every such transport mechanism the PTP standard specifies a particular point in PTP timing messages, often the start of frame signal, as a distinguished point termed the message timestamp point. As a PTP message traverses the protocol stack in a node, the timestamps are generated when the message timestamp point passes a defined point in the stack. This point may be in the application layer, illustrated by "C" in Figure 14, in the kernel or interrupt service routines, illustrated by "B", or in the physical layer of the protocol stack illustrated by "A". In general the closer this point is to the actual network connection, the smaller the timing errors introduced by fluctuations in the time taken to traverse the lower layers. In cases where timestamps are generated in the physical layer, some sort of hardware assist circuitry, illustrated by the dotted lines, is often employed. In this case the timestamp is conveyed to the PTP code via a path outside of the normal path followed by the PTP timing message itself. To ensure that the timestamps are associated with the correct message, the hardware assist often captures additional information from the PTP timing message that is passed with the timestamp to the PTP code.

Implementations that generate PTP timestamps at the physical layer must take the mapping to the on-the-wire format, see 5.4, into account when designing packet recognizers, timestamp generators, and any other form of hardware assist to PTP that is done at the physical layer.

It is possible to design devices where the timestamps are generated between the media access controller (MAC) and the physical layer (PHY) or even within the PHY. In such devices it is likely that all or part of the IEEE code will be embedded and executed within low-level silicon without the use of an operating system.

6.7 PTP communications overview

6.7.1 PTP communication topology

6.7.1.1 General

In systems composed solely of boundary and ordinary clocks, the operation of PTP produces an acyclic graph structure for PTP messages irrespective of the actual underlying network connectivity, following techniques described by [M22].

In all other cases, PTP assumes that the underlying bridging or routing protocols ensure that PTP message forwarding avoids loops. In particular, the protocol assumes that multicast PTP messages are not looped indefinitely within the communication path. The protocol does not assume that a multicast message sent by one PTP port is received on only one of the boundary clock's ports. The protocol does not assume that only a single copy of multicast message sent by one PTP port is received on another PTP port; however, receipt of duplicate copies may impact the precision of time transfer and therefore networks should be designed to avoid this behavior.

6.7.1.2 Hierarchical topology

Different applications favor different topologies. For many systems the hierarchical topology illustrated in Figure 15 is preferred.

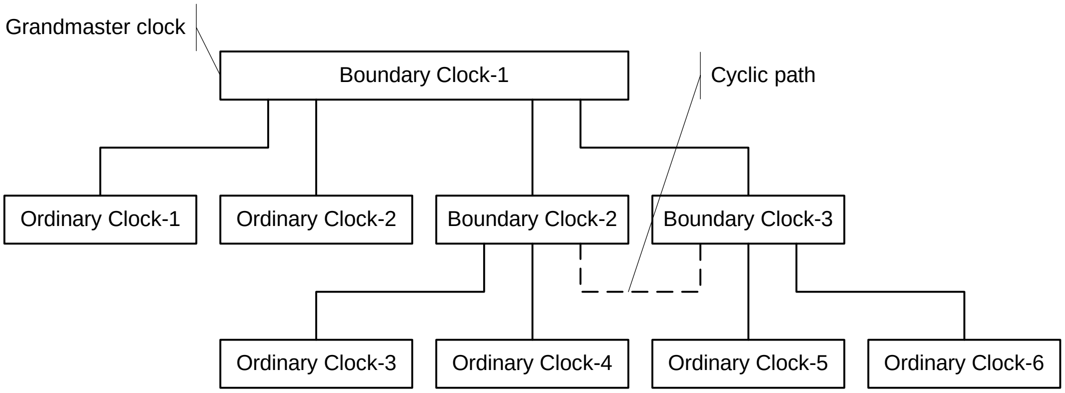


Figure 15: Hierarchical topology

With the exception of the "cyclic" path shown in dashed lines, the other devices in Figure 15 form a tree structured hierarchy with the boundary clocks forming the branch points in the tree. Since boundary clocks can have many ports, this topology allows large numbers of devices to be synchronized with only a few boundary clocks between any slave and the grandmaster. In this example assume that the dashed path is not present. If boundary clock-1 is selected as the grandmaster as shown, the maximum number of intervening boundary clocks is 1. The worst case would be if one of the ordinary clocks, say clock-3, is selected as the grandmaster. Then the maximum number of boundary clocks to the most distant slave, say clock-6, would then be 3.

As noted in 6.2, the protocol expects the underlying topology to avoid forwarding loops. In Figure 15 if the dashed path is present, a loop exists involving the three boundary clocks. The operation of the best master clock and state decision algorithms noted in 6.6.2 breaks this loop for PTP messages.

6.7.1.3 Linear topology

Some applications require long linear topologies as shown in Figure 16 rather than the star or hierarchical topologies of Figure 15. Shown in Figure 16 are two long linear chains of end-to-end transparent clocks with boundary clock-1 as the grandmaster.

Synchronization between an ordinary and a boundary clock involves an exchange of PTP timing messages between the two clocks, for example over path-A. These PTP timing messages are not visible by other clocks in the system. Based on the PTP timing messages, the slave of such a master-slave pair executes some sort of servo mechanism to reduce the clock offset errors.

Cascading servo loops can lead to accumulation of phase error along the cascade. Transparent clocks avoid cascaded servo loops by passing the PTP timing messages through the clock in the manner of an ordinary bridge or router but in addition measuring the time spent by a PTP timing message within the transparent clock. These “residence” times are accumulated in a correction field in the PTP timing messages, which allows the slave to correct the timestamps, effectively removing the timing fluctuations that would otherwise be introduced by the bridges. The penalty is that the master, in this example boundary clock-1, has to process PTP timing messages, i.e., Delay_Req, from all slaves in the linear chain, rather than just from an adjacent boundary clock.

NOTE—This scaling penalty on the master can be avoided by using peer-to-peer transparent clocks or boundary clocks.

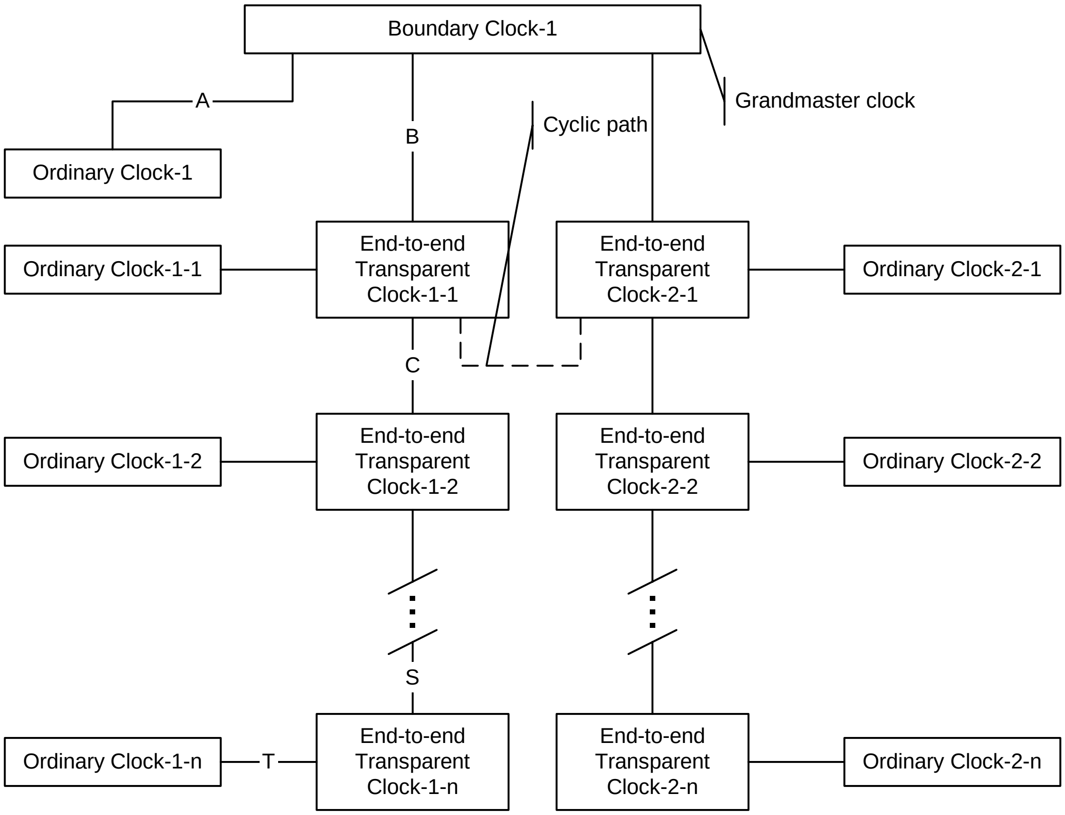

Boundary Clock-1
A
B
Cyclic path
Grandmaster clock
Ordinary Clock-1
Ordinary Clock-1-1
End-to-end Transparent Clock-1-1
End-to-end Transparent Clock-2-1
Ordinary Clock-2-1
C
Ordinary Clock-1-2
End-to-end Transparent Clock-1-2
End-to-end Transparent Clock-2-2
Ordinary Clock-2-2
S
Ordinary Clock-1-n
T
End-to-end Transparent Clock-1-n
End-to-end Transparent Clock-2-n
Ordinary Clock-2-n

Figure 16: Linear topology

6.7.1.4 Rapid reconfiguration in multiply connected topologies

In many applications devices are placed in a multiply connected topology, for example a mesh, as illustrated in Figure 17 or a ring in which alternate paths are logically removed to produce an acyclic topology by a protocol outside of PTP. In the event of a path failure these external protocols reconfigurethe network to restore connectivity. Since PTP operates on top of this underlying, rapidly reconfiguring network, PTP may have to readjust the corrections for path length between master and slave after a reconfiguration.

The peer-to-peer transparent clock is designed for use in this environment.

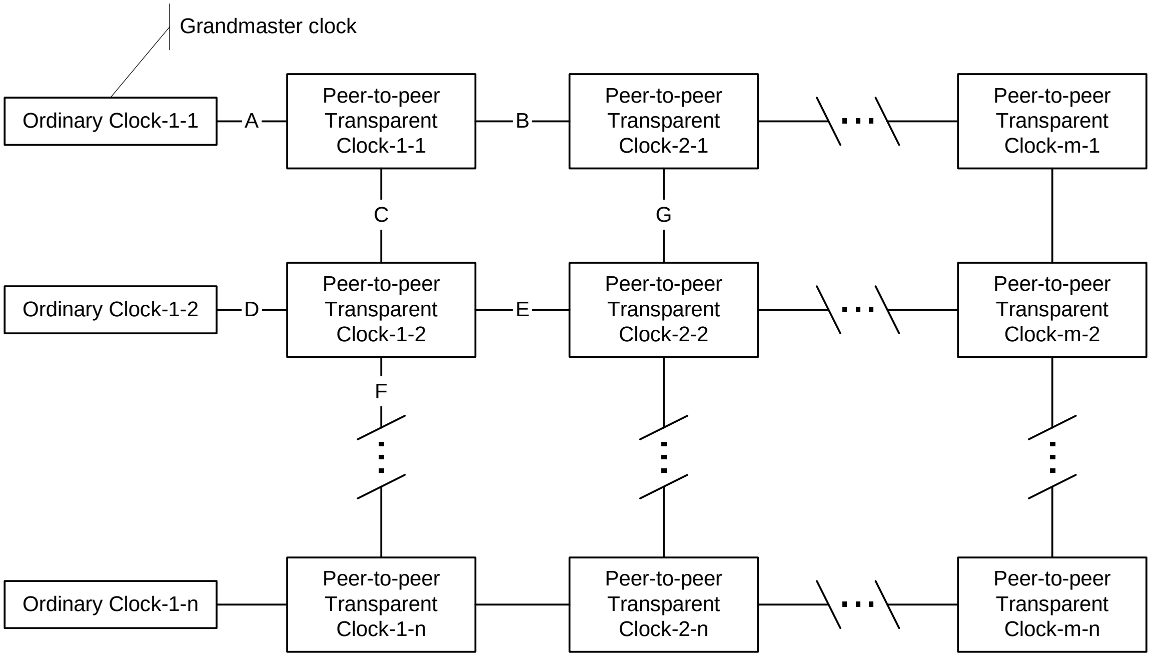

Grandmaster clock
Ordinary Clock-1-1
A
Peer-to-peer Transparent Clock-1-1
B
Peer-to-peer Transparent Clock-2-1
Peer-to-peer Transparent Clock-m-1
C
G
Ordinary Clock-1-2
D
Peer-to-peer Transparent Clock-1-2
E
Peer-to-peer Transparent Clock-2-2
Peer-to-peer Transparent Clock-m-2
F
Ordinary Clock-1-n
Peer-to-peer Transparent Clock-1-n
Peer-to-peer Transparent Clock-2-n
Peer-to-peer Transparent Clock-m-n

Figure 17: Multiply connected topology

Shown in Figure 17 are several peer-to-peer transparent clocks connected in a mesh topology. The operation of the peer-to-peer transparent clock depends on some external protocol eliminating cyclic paths in the network. As in the case of end-to-end transparent clocks, the peer-to-peer transparent clocks are usually associated with an ordinary clock for use in a sensor or other device. The difference between the two types of clocks is in the way that the path length corrections are made.

Suppose that initially the path between ordinary clock-1-1 (the grandmaster) and ordinary clock-1-2 (the slave) was A, B, G, E, D as determined by some non-PTP protocol. A Sync message from ordinary clock-1-1 would be corrected by peer-to-peer clock-1-1 for residence time in the peer-to-peer clock and for link delay A. Likewise peer-to-peer-2-1 would further correct for its residence time and link delay B and so on, so that ordinary clock-1-2 receives a Sync message corrected for residence times in peer-to-peer clocks 1-1, 2-1, 2-2, and 1-2 and link delays A, B, G, and E. Ordinary clock-1-2, which also supports the peer link delay mechanism, corrects for the last link delay D.

Assume that the network is reconfigured such that the new path between these two clocks is now A, C, D. Peer-to-peer clock-1-1 does the same correction as before. However in this case peer-to-peer clock-1-2 now receives the Sync message directly from peer-to-peer clock-1-1 and therefore corrects for its residence time and link delay C, which it has previously measured. It is this prior measurement of link delays on all links, whether active or not, that allows this rapid reconfiguration.

6.7.1.5 Bridging between different network protocols

There is no requirement that all of the PTP communication paths use the same underlying communication media or technology. Boundary clocks are used to bridge between different network transport technologies as illustrated in Figure 18.

In Figure 18, boundary clock-1 and ordinary clocks-1, 3, and 5 are assumed to communicate via network paths labeled A implementing one technology, for example UDP/IP. Boundary clock-2 and ordinary clock-clock-2, clock-4, and clock-6 communicate via network paths labeled B implementing a second technology, for example DeviceNet. One of the boundary clocks, boundary clock-2 in the figure, is a bridge that supports technology A on port-2 and technology B on the remaining ports.

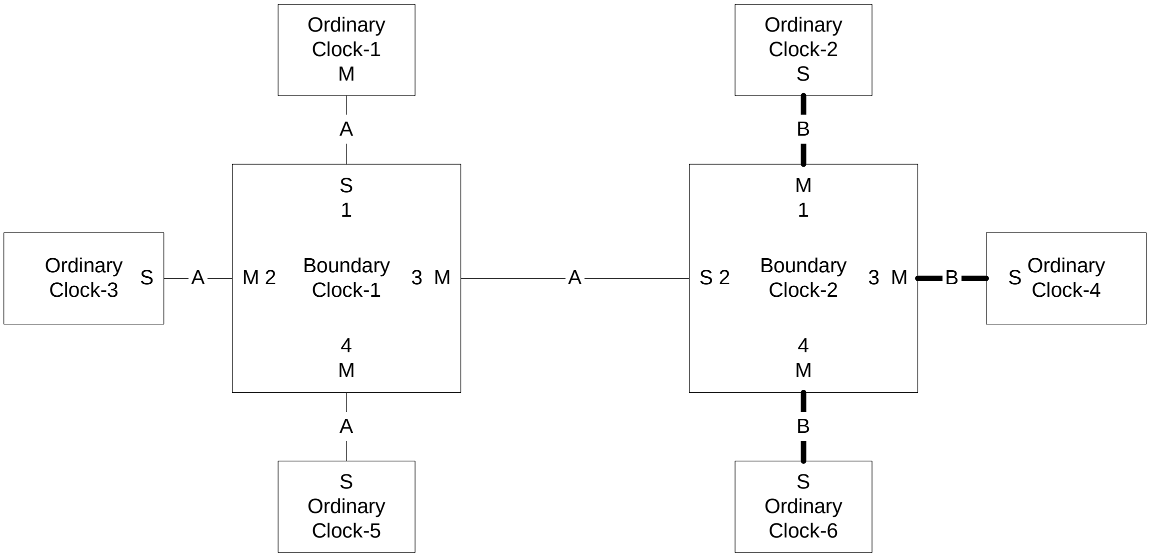


Figure 18: Bridging disparate technologies

In most cases such bridging involves not only a change in network transport protocols, in the above example between UDP/IP and DeviceNet, but possibly other PTP characteristics such as update rates. In this case the boundary clock is the only device that maintains sufficient PTP state to perform the bridging function. In some cases the only bridging function required is translating packet formats or other physical layer issues. In this case it is possible to design transparent clocks to perform the bridging since no PTP state information is required.

6.7.2 System startup

To provide more orderly behavior when a clock comes on line, an ordinary or boundary clock listens for Announce messages from a master for a configurable time interval. If no Announce message is received within this time, the clock assumes it is the master until such time as a ‘better’ clock appears.

An additional mechanism to support more orderly reconfiguration of systems when clocks are added or deleted, clock characteristics change, or connection topology changes is embodied in the PRE_MASTER state. In this state a clock port behaves exactly as it would if it were in the MASTER state except that it does not place certain classes of messages on the port communication path. A clock port remains in this pre-master state long enough to allow changes to propagate from points in the system between the local clock and possible masters visible from the port.

7. Characterization of PTP entities

7.1 domains

A domain consists of one or more PTP devices communicating with each other as defined by the protocol. A domain shall define the scope of PTP message communication, state, operations, data sets and timescale. PTP devices may participate in multiple domains, however unless otherwise specified in this standard the operation of the protocol and the timescale in different domains is independent.

NOTE 1—The mechanism for limiting PTP operation and communications to a domain may involve for example: communication specific techniques such as router table configuration, limiting the physical connectivity, and reliance on application layer processing of the domainNumber field of PTP messages.

NOTE 2—Implementers of PTP nodes need to consider the resources required to support multiple domains. This is particularly true for ordinary and boundary clocks which maintain much more state than transparent clocks. The inability to process the protocol in a timely fashion due to resource limitations may lead to deterioration in the synchronization performance, thrashing, or failure of the protocol. Users need to be aware of this limitation when selecting nodes and designing their systems.

The domain is identified by an integer in the range of 0 to 255 as specified in Table 2.

Table 2: Domains

domain (decimal)	Definition
0	Default domain
1	Alternate domain 1
2	Alternate domain 2
3	Alternate domain 3
4-127	User defined domains
128-255	Reserved

The domain with number 0 is known as the default domain. The value of the domain shall be configurable subject to limits established by a PTP profile.

7.2 PTP timescale

7.2.1 General

The timescale for a domain is established by the grandmaster clock.

There are two types of timescales supported by PTP:

- The timescale PTP: In normal operation the epoch is the PTP epoch and the timescale is continuous, see 7.2.4. The unit of measure of time is the SI second as realized on the rotating geoid.
- The timescale ARB (arbitrary): In normal operation the epoch is set by an administrative procedure. The epoch is permitted to be reset during normal operation. Between invocations of the administrative procedure the timescale is continuous. Additional invocations of the administrative procedure may introduce discontinuities in the overall timescale.

7.2.2 Epoch

The epoch is the origin of the timescale of a domain.
The PTP epoch is 1 January 1970 00:00:00 TAI, which is 31 December 1969 23:59:51.999918 UTC.

NOTE 1—The PTP epoch coincides with the epoch of the common Portable Operating System Interface (POSIX) algorithms for converting elapsed seconds since the epoch to the ISO 8601:2004 printed representation of time of day, see [M16] and [M17].

NOTE 2—See Annex B for information on converting between common timescales.

7.2.3 UTC Offset

When the epoch is PTP, it is possible to calculate UTC time using the value of curretItcOffset.timePropertiesDS data set The value of currentUtcOffset shall be: currentUtcOffset = TAI ─ UTC.

NOTE—As of 0 hours 1 January 2006 UTC, UTC was behind TAI by 33 seconds. At that moment the PTP defined value of currentUtcOffset became +33 seconds [M23].

7.2.4 Measurement of time within a domain

Within a domain, time shall be measured as elapsed time since the epoch.

7.3 PTP communications

7.3.1 Messaging model

While the standard is written based on the multicast model, it is permitted to create an implementation based on a unicast model providing that the behavior of the protocol is preserved.

7.3.2 Message attributes

All PTP related communications occur via PTP messages. PTP messages have the following attributes:

a) Message class,

b) Message source PortIdentity,

c) Message type,

d) Message sequenceIdnumber, and

e) Flags defining options.

These attributes are defined in 7.3.

7.3.3 Message class

7.3.3.1 Event messages

The event message class consists of the following message types:

a) Sync: A Sync message is transmitted by a master to its slaves. It either contains the time of its transmission or is followed by a Follow_Up message containing this time. It may be used by a receiving node to measure the packet transmission delay from the master to the slave. The appearance of a Sync message at the reference plane of a PTP port is an event to which a local clock shall assign a timestamp, the <syncEventIngressTimestamp> or <syncEventEgressTimestamp>, based on the value of the local clock.

b) Delay_Req: A Delay_Req message is a request for the receiving node to return the time at which the Delay_Req message was received, using a Delay_Resp message. The appearance of a Delay_Req message at the reference plane of a PTP port is an event to which a local clock shall assign a timestamp, the <delayReqEventIngressTimestamp> or <delayReqEventEgressTimestamp>, based on the value of the local clock.

c) Pdelay_Req: A Pdelay_Req message is transmitted by a PTP port to another PTP port as part of the peer delay mechanism, see 11.4, to determine the delay on the link between them. The appearance of a Pdelay_Req message at the reference plane of a PTP port is an event to which a local clock shall assign a timestamp, the <pdelayReqEventIngressTimestamp> or <pdelayReqEventEgressTimestamp>, based on the value of the local clock.

d) Pdelay_Resp: A Pdelay_Resp message is transmitted by a PTP port in response to the receipt of a Pdelay_Req message. There are several options for conveying timestamp information in the Pdelay_Resp message:

   1) the difference between the time of transmission of the Pdelay_Response message and time of receipt of the Pdelay _Req message is conveyed in the Pdelay_Resp message,

   2) the difference between the time of transmission of the Pdelay_Respose message and time of receipt of the Pdelay _Req message is conveyed in a Pdelay_Resp_Follow_Up message that follows the Pdelay_Resp message, or

   3) the time of receipt of the Pdelay_Req message is conveyed in the Pdelay_Resp message and the time of transmission of the Pdelay_Resp message is conveyed in a Pdelay_Resp_Follow_Up message that follows the Pdelay_Resp message.

   The appearance of a Pdelay_Resp message at the reference plane of a PTP port is an event to which a local clock shall assign a timestamp, the <pdelayRespEventIngressTimestamp> or <pdelayRespEventEgressTimestamp>, based on the value of the local clock.

Event messages shall be assigned the timestamps defined above as specified in 7.3.4.

7.3.3.2 General messages

The general message class consists of the following message types:

a) Announce: Announce messages provide status and characterization information of the transmitting node and its grandmaster. This information is used by the receiving node when executing the best master clock algorithm.

b) Follow_Up: In a two-step ordinary or boundary clock, see 3.1.44, the Follow_Up message communicates the value of the <syncEventEgressTimestamp> for a particular Sync message.

c) Delay_Resp: The Delay_Resp message communicates the value of the <delayReqEventIngressTimestamp> to the slave port issuing the Delay_Req message.

d) Pdelay_Resp_Follow_Up: In a two-step clock supporting the peer delay mechanism, the Pdelay_Resp_Follow_Up message carries the transmit timestamp <pdelayRespEventEgressTimestamp> generated by a PTP port at the transmission of a Pdelay_Resp message.

e) Management: Management messages communicate information and commands used to manage clocks.

f) Signaling: Signaling messages carry information, requests, and commands between clocks.

General messages are not required to be timestamped.

7.3.4 Generation of event message timestamps

7.3.4.1 Event message timestamp point

Unless otherwise specified in a transport specific Annex to this standard, the message timestamp point for an event message shall be the beginning of the first symbol following the start of frame delimiter.

7.3.4.2 Event timestamp generation

All PTP event messages are timestamped on egress and ingress. The timestamp shall be the time at which the event message timestamp point passes the reference plane marking the boundary between the PTP node and the network.

NOTE 1—If an implementation generates event message timestamps using a point other than the message timestamp point, then the generated timestamps should be appropriately corrected by the time interval between the actual time of detection and the time the message timestamp point passed the reference plane. Failure to make these corrections results in a time offset between the slave and master clocks.

NOTE 2—In general the timestamps may be generated at a point removed from the reference plane. Furthermore, the time offset from the reference plane is likely to be different for inbound and outbound event messages. To meet the requirement of 7.3.4.2, the generated timestamps should be corrected for these offsets. Figure 19 illustrates these offsets. Based on this model the appropriate corrections are:

<egress timestamp> = <egress measured timestamp> + egress_latency
<ingress timestamp> = <ingress measured timestamp> ─ ingress_latency

where the actual timestamps <egress timestamp> and <ingress timestamp> measured at the reference plane are computed from the detected, i.e. measured, timestamps by their respective latencies. Failure to make these corrections results in a time offset between the slave and master clocks.

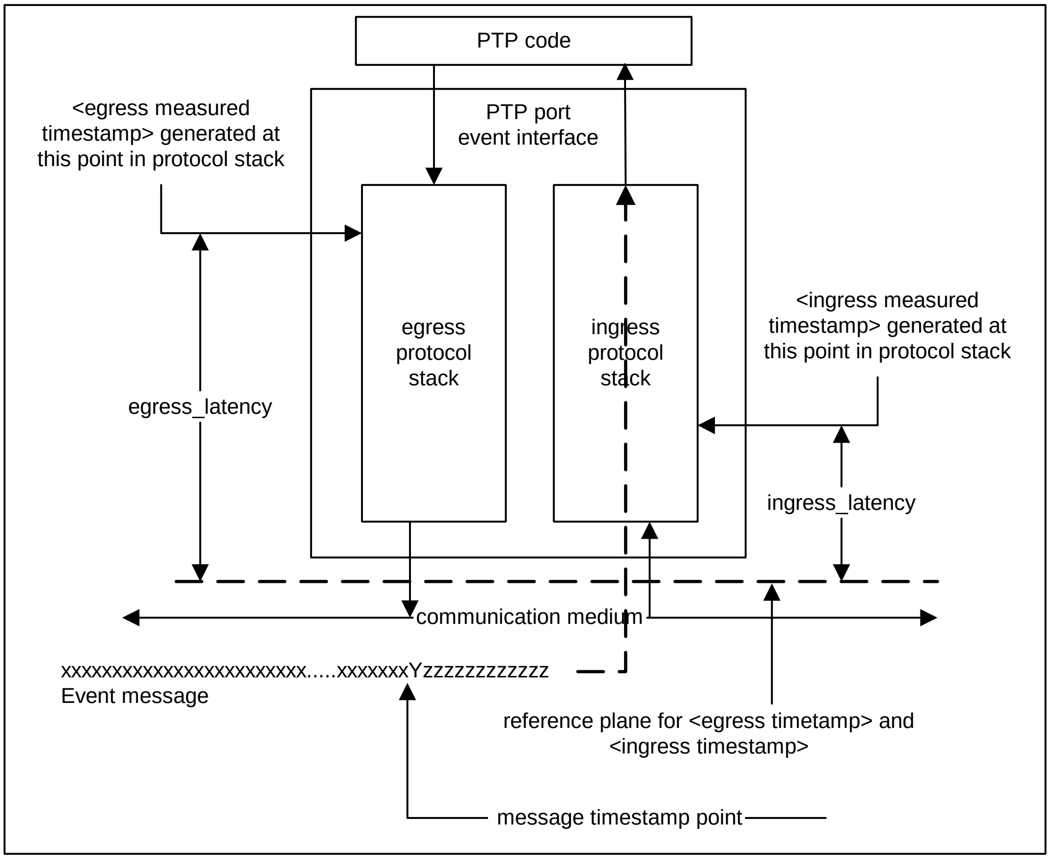


Figure 19: Definition of latency constants

7.3.5 Message sourcePortIdentity

Each PTP message contains a sourcePortIdentity field that identifies the egress port, see 13.3.2.8.

7.3.6 Message types

Each PTP message contains a messageType field that names the PTP message, see 13.3.2.2.

NOTE—PTP message headers also contain an additional field named "controlField", see 13.3.2.10, provided to maintain backward compatibility with hardware designs based on version 1 of this standard.

7.3.7 Message sequenceId

With the exceptions noted below, each port on a PTP ordinary, boundary or transparent clock shall maintain a separate sequenceId pool for each message type sent to a destination address. Multicast addresses are considered to be single destination addresses, and each unicast node is considered a destination address.

The sequenceId of the message shall be one greater than the sequenceId of the previous message of the same message type sent to the same message destination address by the transmitting port, subject to the constraints of the rollover of the UInteger16 data type used for the sequenceId field.

Two PTP messages bearing the same values for the messageType and domainNumber fields and transmitted from the same transmitting protocol address of a PTP node are identical if the values of the sequenceId fields are identical consistent with the rollover properties defined above for the sequenceId.

Separate pools of sequenceId shall not be maintained for the following message types:

- Pdelay_Resp
- Follow_Up
- Delay_Resp
- Pdelay_Resp_Follow_Up, and
- Management messages that are a response to another management message.

For these exceptions, the sequenceId value is specified in Clause 13 and in 15.4.

7.3.8 Flag-based Indicators

7.3.8.1 unicastFlag

A TRUE value of the flag unicastFlag, see 13.3.2.6, indicates that the message was transmitted as a unicast message.

7.3.8.2 alternateMasterFlag

A TRUE value of the flag alternateMasterFlag, see 13.3.2.6, indicates that the message is transmitted from a port not in the MASTER state.

7.3.8.3 twoStepFlag

A TRUE value of the flag twoStepFlag, see 13.3.2.6, indicates that the message was sent from a two step clock.

7.4 PTP communication media

7.4.1 Network transport protocol

PTP communications occur on paths using transport protocols defined by the mapping annexes of this standard. The identification of the transport protocol in use for a communication path shall be indicated by the networkProtocol enumeration of Table 3.

Table 3: networkProtocol enumeration

Name	Value (hex)	Applicable annex
reserved	0	—
UDP/IPv4	1	Annex D
UDP/IPv6	2	Annex E
IEEE 802.3	3	Annex F
DeviceNet	4	Annex G
ControlNet	5	Annex H
PROFINET	6	Annex I
Reserved for assignment by the Precise Networked Clock Working Group of the IM/ST Committee	7-EFFF	—
Reserved for assignment in a PTP profile	F000-FFFD	—
Unknown Protocol	FFFE	—
reserved	FFFF	—

NOTE— Enumeration assignments for other transport mechanisms may be obtained by application to the Precise Networked Clock Working Group of the IM/ST Committee for values in the range 7-EFFF (hex).

7.4.2 Communication path asymmetry

Messages from master to slave and slave to master shall traverse the same network path in any system containing two-step clocks on such paths. Messages from requestor to responder and from responder to requestor shall traverse the same network path in any system containing two-step clocks on such paths. They should traverse the same path in all systems to minimize asymmetry.

The Precision Time Protocol requires that the transmission time of certain messages be measured between a master and a slave clock and between the slave and its master. For the peer delay mechanism it is required to measure the transmission time between a responder and requestor and between the requestor and responder. Typically these times are not the same. PTP characterizes the transmission times as follows:

— meanPathDelay, and

— delayAsymmetry.

The basis for these attributes is illustrated in Figure 20.

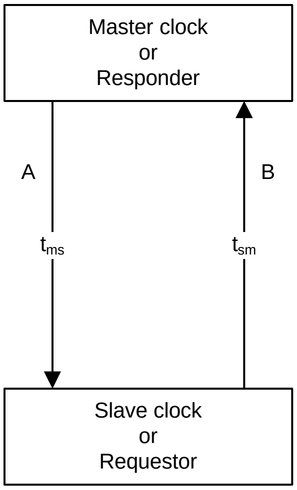


Figure 20: Propagation asymmetry

The meanPathDelay is the mean value of t_{ms} and t_{sm}, i.e. meanPathDelay = (t_{ms} + t_{sm})/2. The value of delayAsymmetry is required for the computations of the actual delay in the master to slave or responder to requestor direction, t_{ms}, used in Clause 11. In many cases the value of delayAsymmetry is below the error budget of the synchronization application.

The attribute delayAsymmetry is defined as follows:
t_{ms} = meanPathDelay + delayAsymmetry, and
t_{sm} = meanPathDelay − delayAsymmetry.

In other words, delayAsymmetry is defined to be positive when the master to slave or responder to requestor propagation time is longer than the slave to master or requestor to responder propagation time.

The measurement of delayAsymmetry is out of scope of this standard. However if known, the propagation asymmetry shall be modeled as specified in 7.4.2 for purposes of correcting timing computations, see 11.6.

7.5 PTP ports

7.5.1 General

The nodes in a PTP system interface with the network via entities called ports as shown in Figure 21.

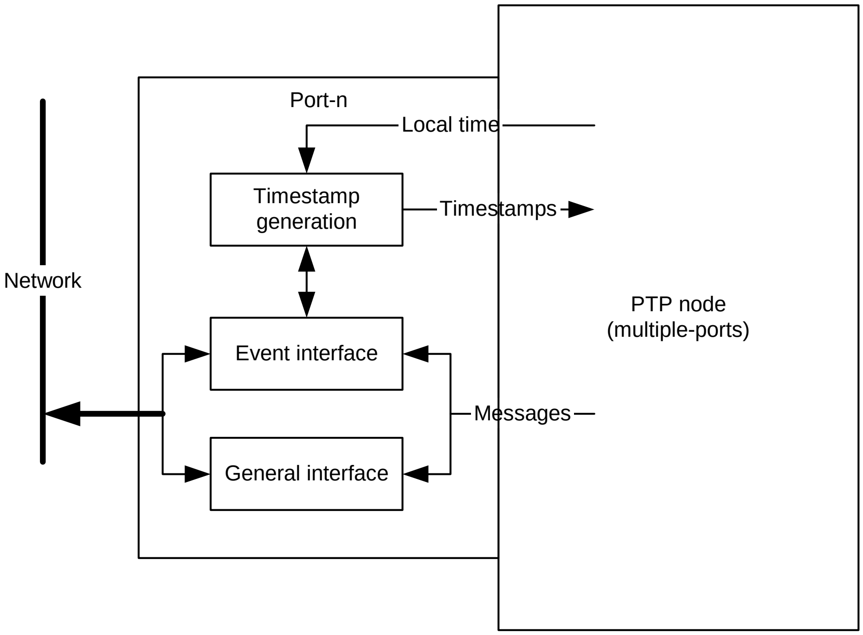


Figure 21: Port model

Each port on a PTP ordinary, boundary or transparent clock is modeled as supporting two interfaces, event and general, as shown in Figure 21. The model shows that timestamps are generated for event messages, see 7.3.4.2 but are not required for general messages.

NOTE—Figure 21 is a model. Unless otherwise stated in this standard there is nothing precluding implementations that, for example, have a single interface, timestamp all messages, and sort out the event messages later.

Each PTP port implements a single version of the protocol and uses a single transport protocol. More than one PTP port can connect to the network via a single physical port.

The attributes of PTP ports are described in 7.5.

7.5.2 PortIdentity

7.5.2.1 General

A PTP port is identified by an attribute portIdentity of type PortIdentity, see 5.3.5. The value is maintained in the portIdentity member of the portDS data set, see 8.2.5.2.1. A portIdentity consists of two attributes:

— portIdentity.clockIdentity

— portIdentity.portNumber.

7.5.2.2 clockIdentity

7.5.2.2.1 General

The clockIdentity is an 8 octet array.

The clockIdentity values shall be taken from either the IEEE EUI-64 individual assigned numbers as specified in subclause 7.5.2.2.2, or from the value set specified in subclause 7.5.2.2.3.

The value of the clockIdentity should be taken from the IEEE EUI-64 individual assigned numbers, see 7.5.2.2.2.

7.5.2.2.2 IEEE EUI-64 clockIdentity values

The most significant 3 octets of the clockIdentity shall be an OUI. The least significant two bits of the most significant octet of the OUI shall both be 0. The least significant bit of the most significant octet of the OUI is used to distinguish clockIdentity values specified by this subclause from those specified in subclause 7.5.2.2.3.

NOTE 1—The values of the least and next to least significant bits of the most significant octet of the OUI indicate respectively: whether the address is a group or individual address, and whether the address is administered universally by the IEEE or locally.

A clockIdentity shall be a EUI-64 or shall be a EUI-64 constructed from a EUI-48.

For devices that use an IEEE EUI-64 for the clockIdentity value:

— The OUI shall be owned by the organization creating an instance of a clockIdentity under the terms of this subclause.

— The organization owning the OUI shall ensure that the remaining 5 octets of the EUI-64 are unique within the scope of clockIdentitiers assigned by the organization.

— The 8 octets of the IEEE EUI-64 shall be assigned in order to the 8 octet array clockIdentity with most significant octet of the IEEE EUI-64 assigned to the clockIdentity octet array member with index 0.

Example [M5]:
The OUI for Company X is ACDE48$_{16}$. If Company X wished to generate a EUI-64 clockIdentity, a legal value would be: ACDE48234567ABCD$_{16}$ where the 5 octet array 234567ABCD$_{16}$ would be guaranteed by Company X to be unique among all Company X EUI-64 assigned numbers used as clockIdentifiers. The byte and bit representations of the clockIdentity are illustrated below from [B5]

OUI			**Extension identifier**					**field**
addr+0	**addr+1**	**addr+2**	**addr+3**	**addr+4**	**addr+5**	**addr+6**	**addr+7**	**order**
AC	DE	48	23	45	67	AB	CD	**hex**
10101100	11011110	01001000	00100011	01000101	01100111	10101011	11001101	**bits**

```
|   |    |                                                              |   |
|   |    group address bit                                              |   |
|  most significant byte                            least significant byte  |
most significant bit                                         least significant bit
```

Implementers may alternatively use a EUI-48 to create the EUI-64 clockIdentity. In this case:

— The EUI-48 shall be an Ethernet MAC address owned by the organization creating the instance of a clockIdentity under the terms of this subclause. The organization owning the MAC address shall guarantee that the MAC address is used in generating only a single instance of a clockIdentity, for example by requiring at the MAC address be a MAC address embedded in the device identified by the clockIdentity.

— The mapping rules for constructing the EUI-64 from the EUI-48 shall be those specified by the IEEE [M5].

— The 8 octets of the created IEEE EUI-64 shall be assigned in order to the 8 octet array clockIdentity with most significant octet of the IEEE EUI-64 assigned to the clockIdentity octet array member with index 0.

NOTE 2—When using an EUI-48, the first 3 octets, i.e. the OUI portion, of the IEEE EUI-48 are assigned in order to the first 3 octets of the clockIdentity with most significant octet of the IEEE EUI-64, i.e. the most significant octet of the OUI portion, assigned to the clockIdentity octet array member with index 0. Octets with index 3 and 4 have hex values FF and FE respectively. The remaining 3 octets of the IEEE EUI-48 are assigned in order to the last 3 octets of the clockIdentity [B5].

NOTE 3—The IEEE registration authority has deprecated the use of MAC-48 in any new design, [M5]

Example [M5]:
If Company X wished to use a EUI-48 based on the MAC address in an owned device of ACDE48234567$_{16}$ as part of the clockIdentity, the resulting clockIdentity would be: ACDE48FFFE234567$_{16}$. The byte and bit representations of the clockIdentity are illustrated below from [B5]

OUI			**Extension identifier**					**field**
addr+0	**addr+1**	**addr+2**	**Addr+3**	**addr+4**	**addr+5**	**addr+6**	**addr+7**	**order**
AC	DE	48	FF	FE	23	45	67	**hex**
10101100	11011110	01001000	11111111	11111110	00100011	01000101	01100111	**bits**

```
|    |      |                                                              |     |
|    |      group address bit                                              |     |
|    most significant byte                                 least significant byte     |
most significant bit                                                 least significant bit
```

Interpretations or functional behaviors dependant on an IEEE 1588 clockIdentity value except for those interpretations and functional behaviors defined in this standard shall not be permitted.

NOTE 4— The clockIdentity value is used by PTP as a unique identifier and not as a network address. While a network address may possibly be inferred from the clockIdentity for technologies using the underlying IEEE assigned numbers for that purpose, out of scope of this standard.

7.5.2.2.3 Non-IEEE EUI-64 clockIdentity values

For devices that do not use IEEE EUI-64, or EUI-48 based clockIdentity values:

— The most significant octet shall be set to FF$_{16}$. The least significant bit of this octet is used to distinguish clockIdentity values specified by this subclause from those specified in subclause 7.5.2.2.2.

— The next 8 bits, i.e. with octet index 1, shall be assigned a value from the non-EUI-64 addressTechnology enumeration of Table 4.

— Octets with indices 2 – 7 shall be assigned values such that the value of the resulting 6 octet array is unique within the protocol technology defined by the octets with index 0 and 1. Technologies using this option, and for which the unique technology specific identifier is less than 6 octets shall left-justify the device's unique technology specific identifier in the field and pad the unused octets with zeroes. In other words, the clockIdentity octet with index 2 shall be significant in all technologies using this option.

— The assignment of protocol technology specific unique identifiers shall be the responsibility of the standards or industry body governing the specifications of the applicable technology.

The 16-bit values for the non-IEEE administered numbers, the non-EUI-64 addressTechnology, shall be as defined in Table 4.

Table 4: Non EUI-64 addressTechnology enumeration

Octet[0] (hex)	**Octet[1]** (hex)	**Communication protocol**
FF	00	Reserved
	01	DeviceNet
	02	ControlNet
	03	PROFINET
	04 to FD	Reserved for assignment by the Precise Networked Clock Working Group of the IM/ST Committee

Octet[0] (hex)	**Octet[1]** (hex)	**Communication protocol**
	FE	Version 1 devices
	FF	Closed system outside the scope of this standard

NOTE 1—The clockIdentity is used by PTP as a unique identifier and not as a network address. While a network address may possibly be inferred from the clockIdentity for technologies using the underlying IEEE assigned numbers for that purpose, such interpretation is out of scope for this standard.

NOTE 2—Protocols listed in this table use addresses not readily mapped into the EUI-64 framework. IEEE 1588 version 1 devices use Ethernet MAC addresses for version 1 UUID fields. The translation between version 1 and version 2 in boundary clocks is simplified by including specific provision for version 1 devices in this table, see 18.3.8.

Example [M5]:
The XYZ Association is the standards organization governing the XYZ network protocol of a future or current annex to this standard. The XYZ Association is assigned a value FF05 (hex) from Table 4. Suppose that one of their member companies, Company Y obtained from the XYZ Association, the 6 octet unique clockIdentity: 23456789ABCD (hex). The resulting 8 octet clockIdentity would be: FF0523456789ABCD (hex). The byte and bit representations are illustrated below.

<table>
<tr><th colspan="3">OUI</th><th colspan="5">Extension identifier</th><th>Field</th></tr>
<tr><td><b>addr+0</b></td><td><b>addr+1</b></td><td><b>addr+2</b></td><td><b>Addr+3</b></td><td><b>addr+4</b></td><td><b>addr+5</b></td><td><b>addr+6</b></td><td><b>addr+7</b></td><td><b>order</b></td></tr>
<tr><td>FF</td><td>05</td><td>23</td><td>45</td><td>67</td><td>89</td><td>AB</td><td>CD</td><td><b>hex</b></td></tr>
<tr><td>11111111</td><td>00000101</td><td>00100011</td><td>01000101</td><td>01100111</td><td>10001001</td><td>10101011</td><td>11001101</td><td><b>bits</b></td></tr>
<tr><td colspan="9">| | | | |
| | group address bit | |
| most significant byte least significant byte |
most significant bit least significant bit</td></tr>
</table>

7.5.2.2.4 Reserved clockIdentity value

The clockIdentity value of all ones shall be reserved for designating all clocks in a domain.

NOTE- This is consistent with the rules specified by the IEEE [M5].

7.5.2.3 portNumber

The value of the portNumber for a port on a PTP node supporting a single PTP port shall be 1. The values of the port numbers for the N ports on a PTP node supporting N PTP ports shall be 1, 2, …N, respectively. The all-zeros and all-ones portNumber values are reserved. The all-ones portNumber, with value $FFFF_{16}$, is used as ‘all-ports' indicator in Management messages, see subclause 15.3.1, and in Signaling messages, see subclause 13.12.1. The all-zeros portNumber is used in data set comparison between portIdentity and clockIdentity, see subclause 7.5.2.4 and Table 12, designating the internal portNumber of the clock. The all-zero portNumber may also be used to represent NULL portNumber value, e.g., it can represent an uninitialized or invalid portNumber value..

7.5.2.4 Ordering of clock and portIdentity values

Two clockIdentity values X and Y are compared as follows:

a) If every octet in X is equal to the corresponding octet in Y, then X = Y.

b) Otherwise, consider the most significant position in which the octets differ, and treat the octets in that position as unsigned integers. If the octet belong to X is smaller than the octet belonging to Y, then X < Y, otherwise X > Y.

Two portIdentities A and B of type PortIdentity with attributes clockIdentity and portNumber are compared as follows:

a) If A.clockIdentity is less than B.clockIdentity, then A<B.

b) Otherwise, if A.clockIdentity is greater than B.clockIdentity, then A>B.

c) Otherwise, if the value of A.portNumber is less than the value of B.portNumber, then A<B.

d) Otherwise, if the value of A.portNumber is greater than the value ofB.portNumber, then A>B.

e) Otherwise, A=B.

A portIdentity A of type PortIdentity with attributes clockIdentity and portNumber and a clockIdentity B of type ClockIdentity are compared as follows:

a) If A.clockIdentity is less than B.clockIdentity, then A<B.

b) Otherwise, if A.clockIdentity is greater than B.clockIdentity, then A>B.

c) Otherwise, B<A.

7.5.3 State

There are two kinds of PTP ports: stateful and stateless. PTP stateful ports support the state mechanism of 9.2. PTP stateful ports are characterized by the current state of the PTP state machine associated with the port. PTP stateless ports do not support the PTP state machine and do not have this state attribute.

NOTE—This definition of stateful and stateless PTP ports does not rule out state mechanisms other than that of 9.2 that may apply to a PTP port.

7.5.4 Path delay measurement mechanism

There are two mechanisms for measuring the propagation time of an event message. These mechanisms are:

- delay-response mechanism, see 11.3, which measures the propagation time between two stateful PTP ports.
- peer delay mechanism, see 11.4, which measures the propagation time between two ports supporting this peer delay mechanism.

NOTE—The peer delay mechanism may be supported on both stateful and stateless PTP ports.

7.5.5 versionNumber

The versionNumber attribute indicates the version of this standard implemented on the port. For this edition of the standard, the versionNumber attribute has the value 2.

NOTE— Auto-negotiation of versionNumber is not precluded but is out of scope of this standard.

7.6 PTP device characterization

7.6.1 PTP device type

There are five types of PTP devices:

a) Ordinary clock,

b) Boundary clock,

c) End-to-end transparent clock,

d) Peer-to-peer transparent clock, and

e) Management node.

All PTP devices are identified by a clockIdentity attribute.

In addition, ordinary and boundary clocks are characterized by the attributes:

a) priority1

b) priority2

c) clockClass

d) clockAccuracy

e) timeSource

f) offsetScaledLogVariance

g) numberPorts.

Ordinary and boundary clocks may keep statistics on the performance of their parent using the attributes:

a) observedParentOffsetScaledLogVariance

b) observedParentClockPhaseChangeRate.

7.6.2 PTP device attributes

7.6.2.1 clockIdentity

The clockIdentity value for a clock or management node shall be as specified in 7.5.2.2.

7.6.2.2 priority1

The attribute priority1 is used in the execution of the best master clock algorithm, see 9.3.2. Lower values take precedence. The initialization value of priority1 is specified in a PTP profile. The value of priority1 shall be configurable to any value in the range 0 to 255, unless restricted by limits established by an applicable PTP profile.

NOTE— The operation of the best master clock algorithm selects clocks from a set with a lower value of priority1 over clocks from a set with a greater value of priority1.

7.6.2.3 priority2

The attribute priority2 is used in the execution of the best master clock algorithm, see 9.3.2. Lower values take precedence. The initialization value of priority2 is specified in a PTP profile. The value of priority2 shall be configurable to any value in the range 0 to 255, unless restricted by limits established by an applicable PTP profile.

NOTE— In the event that the operation of the best master clock algorithm fails to order the clocks based on the values of priority1, clockClass, clockAccuracy and scaledOffsetLogVariance, the priority2 attribute allows the creation of up to 256 priorities to be evaluated before the tie-breaker. The tie-breaker is based on the clockIdentity.

7.6.2.4 clockClass

The clockClass attribute of an ordinary or boundary clock denotes the traceability of the time or frequency distributed by the grandmaster clock. The interpretation and allowed values of clockClass shall be based on the definitions in Table 5.

Table 5: clockClass specifications

PTP clockClass (decimal)	Specification
0	Reserved to enable compatibility with future versions.
1-5	Reserved
6	Shall designate a clock that is synchronized to a primary reference time source. The timescale distributed shall be PTP. A clockClass 6 clock shall not be a slave to another clock in the domain.
7	Shall designate a clock that has previously been designated as clockClass 6 but that has lost the ability to synchronize to a primary reference time source and is in holdover mode and within holdover specifications. The timescale distributed shall be PTP. A clockClass 7 clock shall not be a slave to another clock in the domain.
8	Reserved
9-10	Reserved to enable compatibility with future versions.
11-12	Reserved
13	Shall designate a clock that is synchronized to an application specific source of time. The timescale distributed shall be ARB. A clockClass 13 clock shall not be a slave to another clock in the domain.
14	Shall designate a clock that has previously been designated as clockClass 13 but that has lost the ability to synchronize to an application specific source of time and is in holdover mode and within holdover specifications. The timescale distributed shall be ARB. A clockClass 14 clock shall not be a slave to another clock in the domain.
15-51	Reserved
52	Degradation alternative A for a clock of clockClass 7 that is not within holdover specification. A clock of clockClass 52 shall not be a slave to another clock in the domain.
53-57	Reserved
58	Degradation alternative A for a clock of clockClass 14 that is not within holdover specification. A clock of clockClass 58 shall not be a slave to another clock in the domain.
59-67	Reserved
68-122	For use by alternate PTP profiles
123-127	Reserved
128-132	Reserved
133-170	For use by alternate PTP profiles
171-186	Reserved
187	Degradation alternative B for a clock of clockClass 7 that is not within holdover specification. A clock of clockClass 187 may be a slave to another clock in the domain.
188-192	Reserved
193	Degradation alternative B for a clock of clockClass 14 that is not within holdover specification. A clock of clockClass 193 may be a slave to another clock in the domain.
194-215	reserved
216-232	For use by alternate PTP profiles

PTP clockClass (decimal)	**Specification**
233-247	Reserved
248	Default. This clockClass shall be used if none of the other clockClass definitions apply.
249-250	Reserved
251	Reserved for version 1 compatibility, see Clause 18.
252-254	Reserved
255	Shall be the clockClass of a slave-only clock, see 9.2.2.

NOTE 1—The clockClass number ranges 68 – 122 and 133-170 are reserved for definition in an alternate PTP profile. For example, it is expected that these ranges will be used by PTP profiles defining applications that distribute only frequency. The break at 128 allows these PTP profiles to select whether a clock not selected as master by the best master clock algorithm has its port in the PASSIVE or SLAVE states respectively.

NOTE 2—The clockClass number range 216-232 is expected to be used by PTP profiles that require clocks to be given preference based on some application specific precedence. For example, controllers might take precedence over sensors in an industrial application.

Unless otherwise specified in an PTP profile, degradation alternative B shall be selected.

Unless a clock is specifically designed to maintain clock accuracy during circumstances that result in the execution of the POWERUP event, see 9.2.6.2, the execution of this event shall preclude assigning a clockClass value of 6, 7, 13, or 14.

In a given domain, clocks with clockClass values less than 128 should be inherently at least as stable (low variance) as any clock with a clock class value greater than its own..

If the inherent characteristics of a clock change such that the clockClass or clockAccuracy designations no longer apply, the clock shall either:

— Upgrade or degrade its clockClass and clockAccuracy in such a way as to correctly specify the current clock characteristics, or

— Be placed in the FAULTY state.

7.6.2.5 clockAccuracy

The clockAccuracy characterizes a clock for the purpose of the best master clock (BMC) algorithm. The value of clockAccuracy shall be taken from the enumeration in Table 6. The value of this attribute shall be estimated by the clock to a precision consistent with the value of the selected enumeration, e.g. for 23_{16} a precision of plus or minus 0.5 μs and for 31_{16} a precision of 10 s. This estimate shall be based on the timeSource attribute, 7.6.2.6, the elapsed time since last synchronized to this time source, and the holdover specifications of the clock. If the information to determine this estimate is not available then the enumeration specification Unknown shall be used.
The clockAccuracy indicates the expected accuracy of a clock when it is the grandmaster, or in the event it become the grandmaster.

Table 6: clockAccuracy enumeration

Value_{16}	**Specification**
00-1F	reserved
20	The time is accurate to within 25 ns
21	The time is accurate to within 100 ns

Value$_{16}$	Specification
22	The time is accurate to within 250 ns
23	The time is accurate to within 1 us
24	The time is accurate to within 2.5 us
25	The time is accurate to within 10 us
26	The time is accurate to within 25 us
27	The time is accurate to within 100 us
28	The time is accurate to within 250 us
29	The time is accurate to within 1 ms
2A	The time is accurate to within 2.5 ms
2B	The time is accurate to within 10 ms
2C	The time is accurate to within 25 ms
2D	The time is accurate to within 100 ms
2E	The time is accurate to within 250 ms
2F	The time is accurate to within 1 s
30	The time is accurate to within 10 s
31	The time is accurate to >10 s
32-7F	reserved
80-FD	For use by alternate PTP profiles
FE	Unknown
FF	reserved

The ordering of clock accuracy in the operation of the best master clock algorithm, see 9.3.2, is specified as follows. When comparing clock accuracies, clock A shall be deemed better than clock B if the value of the clockAccuracy of A is lower than that of B.

NOTE— The range from 80-FD$_{16}$ is reserved for use by alternate PTP profiles. It is expected that this range will be used by PTP profiles defining applications that distribute only frequency to define accuracy specifications appropriate for frequency distribution.

7.6.2.6 timeSource

This is an information only attribute indicating the source of time used by the grandmaster clock. The value is not used in the selection of the grandmaster clock. The values shall be as specified in Table 7. These represent categories. For example, the GPS entry would include not only the GPS system of the U.S. Department of Defense but the European Galileo system and other present and future satellite-based timing systems.

Table 7: timeSource enumeration

Value$_{16}$	timeSource	Description
10	ATOMIC_CLOCK	Any device, or device directly connected to such a device, that is based on atomic resonance for frequency and that has been calibrated against international standards for frequency and, if the PTP timescale is used, time
20	GPS	Any device synchronized to any of the satellite systems that distribute time and frequency tied to international standards
30	TERRESTRIAL_RADIO	Any device synchronized via any of the radio distribution systems that distribute time and frequency tied to international standards
40	PTP	Any device synchronized to a PTP based source of time external to the domain

Value$_{16}$	timeSource	Description
50	NTP	Any device synchronized via NTP or Simple NTP (SNTP) to servers that distribute time and frequency tied to international standards
60	HAND_SET	Used in all cases for any device whose time has been set by means of a human interface based on observation of an international standards source of time to within the claimed clock accuracy
90	OTHER	Other source of time and/or frequency not covered by other values
A0	INTERNAL_OSCILLATOR	Any device whose frequency is not based on atomic resonance nor calibrated against international standards for frequency, and whose time is based on a free-running oscillator with epoch determined in an arbitrary or unknown manner
F0-FE	For use by alternate PTP profiles	
FF	Reserved	

All unused values are reserved.

NOTE 1— The values for clockClass, clockAccuracy, and timeSource should be consistent. For example, a class 6 atomic clock synchronized directly to the GPS system might claim an accuracy of 25 ns while the same atomic clock not synchronized to GPS but set via a user interface by a user observing a National Institute of Standards and Technology (NIST) server via the web might claim to be class 6 but with an accuracy of 10 s.

NOTE 2—The range from F0-FE$_{16}$ is reserved for use by alternate PTP profiles. It is expected that this range will be used by PTP profiles defining applications that distribute only frequency to define the nature of sources appropriate for frequency distribution.

NOTE 3—These designations may or may not carry over a power-fail restart but in any case should reflect the current status of the node. For example, a simple quartz oscillator at turn on would be INTERNAL_OSCILLATOR. If later the epoch was set by hand it would be HAND_SET while if it later synchronized to GPS it would be GPS. If it had a battery-backed up real-time clock, this status could survive a power-fail restart although the clockAccuracy and perhaps clockClass would be degraded.

7.6.2.7 numberPorts

The attribute, numberPorts, shall indicate the number of PTP ports on the PTP device.

7.6.3 PTP variance

7.6.3.1 General

Two variance estimates, as specified in 7.6.3.2, characterize ordinary and boundary clocks in a PTP system:

— Each such clock shall maintain an estimate, the offsetScaledLogVariance, see 7.6.3.5, of its inherent precision. This is the precision of the timestamps included in messages issued by the clock when it is not synchronized to another clock using the protocol.

— If such a clock, clock_A, is synchronized to another using the PTP protocol, it may maintain an estimate, the observedParentOffsetScaledLogVariance, see 7.6.4.3, of the precision of the clock to which it is synchronized as observed by clock_A.

7.6.3.2 Variance algorithm

The PTP variance, from which offsetScaledLogVariance and observedParentOffsetScaledLogVariance are derived, is based on the theory of Allan deviation as follows.

The Allan deviation $\sigma_{\mathrm{y}}(\tau)$ is estimated as follows:[M19]

$$\sigma_y(\tau)=\left[\frac{1}{2(N-2)\tau^2}\times\sum_{k=1}^{N-2}\left(x_{k+2}-2x_{k+1}+x_k\right)^2\right]^{\frac{1}{2}}$$

where x_k, x_{k+1}, and x_{k+2} are time residual measurements made at times t_{k}, $\mathrm{t}_{\mathrm{k}}+\tau$, and $\mathrm{t}_{\mathrm{k}}+2\tau$. τ is the sample period between measurements and *N* is the number of data samples. The term residual implies that any consistent systematic effects have been removed from the data.

The Allan deviation as stated is a second-order statistic on the variation of the frequency of the oscillator used as the basis of the time base.

The PTP variance is defined by $\sigma_{PTP}^2=\tau^2\times\frac{1}{3}\sigma_y^2$. An unbiased estimate of the PTP variance shall be computed as follows:

$$\sigma_{PTP}^2=\frac{1}{3}\left[\frac{1}{2(N-2)}\times\sum_{k=1}^{N-2}\left(x_{k+2}-2x_{k+1}+x_k\right)^2\right]$$

where x_{k}, x_{k+1}, and x_{k+2} are time residual measurements, made at times t_{k}, $\mathrm{t}_{\mathrm{k}}+\tau$, and $\mathrm{t}_{\mathrm{k}}+2\tau$, between the time provided by the measured clock and a local reference clock, and *N* is the number of data samples. For a PTP variance the quantity τ, the sample period, shall be the value defined in the applicable PTP profile. Subclauses 8.2.1.3.1.3 and 8.2.3.4 specify variances to be computed using the specifications in this clause. If these variances are computed during execution, data is only available in multiples of the sync interval, 7.7.2.3. In this case τ should be a multiple of the sync interval.

Implementations may compute a conservative estimate of the PTP variance rather than computing the exact value given here. Note this may be necessary in implementations with limited computational or memory resources.

NOTE—The dependence of the Allan deviation on the sample period provides information on the type of the underlying noise processes. The Allan deviation is not sensitive to constant offsets in time or in frequency, even though those offsets may be important in some applications of this standard. In addition, the Allan deviation does not provide a useful diagnostic when the noise spectrum contains "bright lines" — power-line induced variations at 60 Hz, for example. Finally, the Allan deviation is computed as an average over the ensemble of observations, and it is most useful when the data are statistically stationary. The deviation does not provide a good measure of the frequency or amplitude of occasional glitches, even though those sorts of events might also be important in some applications of this standard.

7.6.3.3 Variance representation

PTP variances shall be represented as follows:

a) An estimate of the variance, σ_{PTP}^2, specified in 7.6.3.2 is computed in units of seconds squared

b) The logarithm to the base 2 of this estimate is computed. The computation of the logarithm need not be more precise than the precision of the estimate of the variance.

c) The logarithm is multiplied by 2^8 to produce a scaled value.

d) This scaled value is modified per the hysteresis specification of 7.6.3.3 to produce the reported value.

e) The reported value is represented as a 2's complement Integer16. The value 8000_{16} is added to the reported value represented in this form and any overflow is ignored. The result, i.e., the offset scaled reported value, is cast as a UInteger16.

f) This offset scaled reported value, represented as UInteger16, shall be the value of the log variances specified in 7.6.3.

NOTE 1—For example, suppose the PTP variance value is $1.414 \times 2^{-73} = 1.497 \times 10^{-22}$ s^2. Then, $\log_2(1.414 \times 2^{-73})$ = -73 + 0.5 = -72.5. If this were expressed as an Integer16, it would truncate to -72. To retain some precision the value is scaled by 2^8 to yield a scaledLogVariance of -18560, $D780_{16}$, which retains 8 bits more precision. To this is added 8000_{16} to yield the offset scaled reported value 5780_{16}.

NOTE 2—The smallest variance that can be represented is 2^{-128} or ~ $3x10^{-39}$ s^2, which results in an offsetScaledLogVariance of 0000_{16}. The maximum variance that can be represented is ~ $2^{+127.99609}$, which results in an offsetScaledLogVariance of $FFFF_{16}$.

NOTE 3—This representation ensures that the ordering of variances algorithm of 7.6.3.4 produces identical results in all implementations. This cannot be guaranteed with a floating-point representation.

The largest possible positive number, $FFFF_{16}$, for the offsetScaledLogVariance attribute shall indicate that the variance is either too large to be represented, or has not been computed.

Since variance values are used in the selection of the best master clock, see 9.3.2, implementations in which variance values are computed during operation shall include hysteresis in the estimation of the variances to preclude thrashing in the process of selecting the master clock. The magnitude of this hysteresis applied to the 2^8 scaled values of the $\log_2$ of the reported estimates shall be PTP_SCALED_LOG_VARIANCE_HYSTERESIS, as shown in Figure 22. This hysteresis shall be applied to the scaled values of the logarithm of the estimate used to generate the offset scaled values of the logarithm of the estimate that is actually reported and used in the computations of the best master clock algorithm (see 9.3). Sufficient local state needs to be be maintained to allow correct implementations of the hysteresis properties for both increasing and decreasing trends in the actual variance estimate. The value of PTP_SCALED_LOG_VARIANCE_HYSTERESIS is 2^7.

NOTE— A value of 2^7 corresponds to a change in the value of the actual $\log_2$(actual estimate) of 1/2. Fluctuations in computed $\log_2$(estimated variance) below 1/2 are not be reported due to this hysteresis requirement.

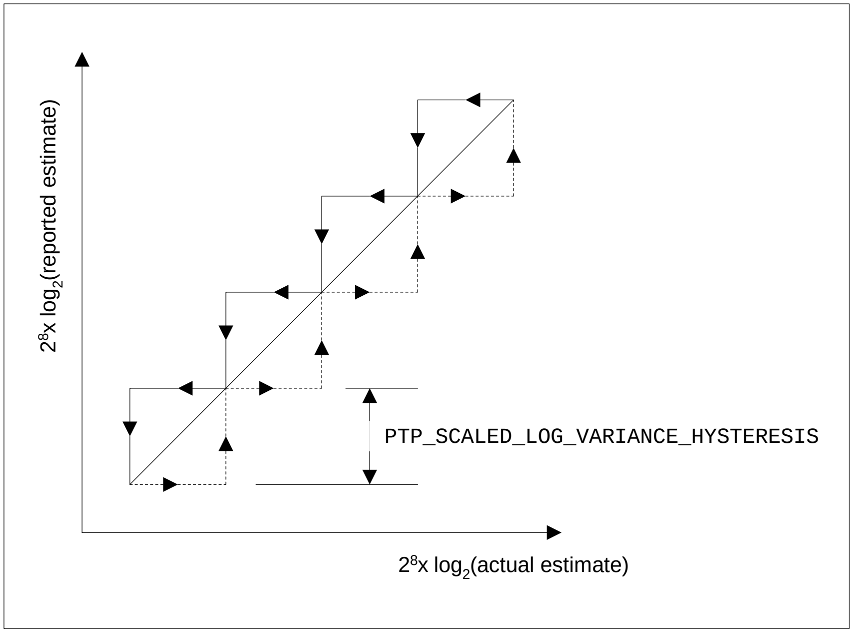

2^8x $\log_2$(reported estimate)
PTP_SCALED_LOG_VARIANCE_HYSTERESIS
2^8x $\log_2$(actual estimate)

Figure 22: Scaled log variance hysteresis

7.6.3.4 Ordering of variances

The ordering of variances shall be computed on the offset, scaled, logarithmic representation of the variance.

7.6.3.5 Computation of offsetScaledLogVariance

The computation of the value of offsetScaledLogVariance for the defaultDS data set shall be based on the characteristics of the local clock as measured by a perfect clock. The value of offsetScaledLogVariance shall:

- Be a static constant determined by the manufacturer, or
- Be computed based on measured or modeled behavior of the components of the local clock and its environment.

The value of offsetScaledLogVariance shall be computed and represented as described in 7.6.3.2 and 7.6.3.3.

The value of offsetScaledLogVariance shall be an estimate of the variations of the local clock from a linear timescale when it is not synchronized to another clock using the protocol. The reference clock when not synchronized to another clock may be an atomic clock, a GPS receiver, a stable local oscillator, a suite of clocks synchronized via NTP, etc. These sources may contribute to the variance estimate.

The value of offsetScaledLogVariance shall be the variance applicable to an ensemble of measurements that include error contributions from:

- Synchronization to its reference clock,
- Variation in clock phase change rate, and noise characteristics of the local oscillator,
- Sampling quantization errors, fluctuations in inbound and outbound latency, and other fluctuations of the local clock.

7.6.4 Parent clock statistics

7.6.4.1 General

A clock in the slave state may maintain statistics on the observed performance of its parent as defined in 7.6.4. The two optional statistics are:

- observedParentOffsetScaledLogVariance and
- observedParentClockPhaseChangeRate.

7.6.4.2 parentStats

This attribute shall indicate whether the values of observedParentOffsetScaledLogVariance and observedParentClockPhaseChangeRate have been measured and are valid. A TRUE value shall indicate valid data.

7.6.4.3 observedParentOffsetScaledLogVariance

The observedParentOffsetScaledLogVariance of a local clock shall be the variance of the parent clock's phase as measured by the local clock. This measurement is optional.

The value of observedParentOffsetScaledLogVariance shall be computed and represented per 7.6.3.2 and 7.6.3.3.

7.6.4.4 observedParentClockPhaseChangeRate

The observedParentClockPhaseChangeRate shall be an estimate of the parent clock's phase change rate as measured by the slave clock, see 3.1.23. The reported value shall be the fractional frequency offset multiplied by 2^{+40}. If the estimate exceeds the capacity of its data type this value shall be set to the largest or smallest allowable value, as appropriate. A positive sign indicates that the parent clock is faster than the clock of the slave clock. This measurement is optional.

NOTE 1—The minimum phase change rate that can be expressed is 2^{-40} or approximately 1 ps per s.

NOTE 2—This value is dependent on the measurement time interval used. When this value is critical for an application domain, the time interval should be specified in the applicable PTP profile.

7.6.5 numberPorts

The attribute numberPorts shall be the number of ports on a device that supports PTP.

7.7 PTP timing characterization

7.7.1 General

Subclause 7.7 specifies timing and timeout attributes of the protocol.

7.7.2 Message transmission intervals

7.7.2.1 General interval specification

For each of the message types Announce, Sync, Delay_Req, and Pdelay_Req, the mean time interval between successive messages shall be represented as the logarithm to the base 2 of this time interval measured in seconds on the local clock of the device sending the message. The values of these logarithmic attributes shall be selected from integers in the range -128 to 127 subject to further limits established in an applicable PTP profile. These intervals are communicated via the logMessageInterval field of PTP messages. The interpretation of the logMessageInterval depends on the message type, see 13.3.2.11. A node shall, with 90% confidence, issue messages with intervals within +/- 30% of the stated value of this attribute.

7.7.2.2 Announce message transmission interval

The logAnnounceInterval shall specify the mean time interval between successive Announce messages i.e. the announceInterval.

The logAnnounceInterval should be uniform throughout a domain. The behavior of domains in which this is not so is outside the scope of this standard.

NOTE 1—The value of logAnnounceInterval is a compromise between the desired responsiveness to changes in the network in determining the master slave hierarchy in a domain, and the communication and computation load imposed by transmission of these messages.

NOTE 2—It may be desirable for the logAnnounceInterval to be different in regions of different communication technologies, e.g. wired and wireless technologies. Systems where the logAnnounceInterval varies from region to region will still operate correctly. However, it is possible that the regions with short intervals will experience more reconfiguration while waiting for the slower regions to select masters using the best master clock algorithm than would be the case in a system with uniform values of the interval.

7.7.2.3 Sync (multicast) message transmission interval

The logSyncInterval shall specify the mean time interval between successive Sync messages when transmitted as multicast messages.

NOTE 1—It may be desirable for the logSyncInterval to be different in regions of different communication technologies, e.g. wired and wireless technologies.

NOTE 2—The value of logSyncInterval is a compromise between the stability and precision of the local clocks and the communication and computation load imposed by transmission of these messages.

7.7.2.4 Delay_Req message transmission interval

The logMinDelayReqInterval shall specify the minimum permitted mean time interval between successive Delay_Req messages sent by a slave to a specific port on the master, see 9.5.11.2.

This value is determined and advertised by a master clock based on the ability of the master clock to process the Delay_Req message traffic. The value shall be an integer with the minimum value being logSyncInterval, i.e. at the same rate as Sync messages, and a maximum value of logSyncInterval+5, i.e. one Delay_Req message every 32 Sync messages.

NOTE— The value of log_delay_req_interval is a compromise between the responsiveness in changes to path delay and the communication and computation load imposed by transmission of these messages.

7.7.2.5 Pdelay_Req message transmission interval

The logMinPdelayReqInterval shall specify the minimum permitted mean time interval between successive Pdelay_Req messages sent over a link.

NOTE— The value of logMinPdelayReqInterval is a compromise between the fluctuation in link delay and startup time and the communication and computation load imposed by transmission of these messages.

7.7.3 PTP timeouts

7.7.3.1 announceReceiptTimeout

The value of this attribute shall specify the number of announceIntervals that have to pass without receipt of an Announce message before the occurrence of the event ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES, see 9.2.6.11. The range shall be 2 to 255 subject to further restrictions of a PTP profile. The minimum value should be 3.

The product (portDS.announceReceiptTimeout)* $2^{\text{logAnnounceInterval}}$ should be uniform throughout a domain. The behavior of domains where this is not so is outside the scope of this standard.

NOTE— The value of this attribute is a compromise between rapid response to failed devices or network paths and undue thrashing of the best master clock algorithm in determining the master-slave hierarchy due to occasional missed Announce messages.

8. PTP data sets

8.1 General specifications for data set members

8.1.1 Introduction to data sets

8.1.1.1 Introduction to data set specifications

Each data set member specification includes:

- The formal name for the member.
- A reference or definition of the semantics associated with the member.
- Initialization and configuration properties, see 8.1.3.

8.1.1.2 Ordinary and boundary clocks

For each ordinary clock and boundary clock, the following ‘clock data sets’ shall be maintained locally as the basis for protocol decisions and for providing values for message fields:

- defaultDS (see 8.2.1)
- currentDS (see 8.2.2)
- parentDS (see 8.2.3)
- timePropertiesDS (see 8.2.4)
- portDS (one data set for each port of an ordinary or boundary clock, see 8.2.5)

8.1.1.3 Transparent clocks

For each transparent clock, the following ‘transparent clock data sets’ specified in the subclasses of Clause 8 should be maintained locally as the basis for protocol decisions and for providing values for message fields:

- transparentClockDefaultDS data set (see 8.3.2)

- transparentClockPortDS data set (one data set for each port, see 8.3.3)

8.1.2 Initialization classification

8.1.2.1 General

Every member of a data set is classified as static, dynamic or configurable.

8.1.2.1.1 Static members

The values of static members are inherent physical or operational properties of the clock or of the protocol.

8.1.2.1.2 Dynamic members

The values of dynamic members are not directly changed by users but may change:

- As a result of protocol operations. For example, the portDS.portState may change as a result of protocol events. A management message may cause an event that results in a change in portDS.portState but only indirectly via the action of the state machine and in some designs could be set by a management message.
- Due to changes in the internal properties of the clock. For example, the offsetScaledLogVariance may change due to temperature effects on the local oscillator.
- Due to interactions with timing systems external to PTP. For example, a clock that is able to synchronize to the GPS system may change the value of its clockQuality, time, or timePropertiesDS data set members when it first locks to GPS.

8.1.2.1.3 Configurable members

The values of configurable members can only be changed using management messages or implementation-specific configuration means.

Unless otherwise stated in this standard, when the value of a configurable member is updated, the update value shall take effect immediately upon update.

Unless otherwise stated in this standard, the update values for configurable members shall be restricted in range to the most restrictive of the range values specified in Clause 7 or specified in the applicable PTP profile.

NOTE—For example, the value of the domain can be changed via a management message but is otherwise unaffected by the protocol or internal changes in the clock.

8.1.3 Clock data set initialization properties

8.1.3.1 General initialization specifications

The initialization properties of a data set member shall be determined by its classifications as static, dynamic, or configurable.

Data set members shall be initialized before leaving the INITIALIZATION state of an ordinary or boundary clock and before beginning normal operation of a transparent clock as specified in 8.1.3.2, 8.1.3.3, and 8.1.3.4.

8.1.3.2 Initialization of static data set members

Static members shall be initialized to the implementation-specific value meeting the specifications for the member.

8.1.3.3 Initialization of dynamic data set members

Dynamic members shall be initialized to the first of the following values that applies:

a) The value mandated in the specification for the data set member,

b) The value that represents the properties of the clock or protocol at the time of initialization,
NOTE—For example, the clockQuality value may depend on whether the clock is synchronized to GPS at initialization.

c) The value in non-volatile read-write storage if implemented, NOTE—For example, an implementation may store the value of meanPathDelay in non-volatile storage on the assumption that the network is unlikely to be reconfigured frequently.

d) Implementation-specific value.

If dynamic member values are maintained in non-volatile read-write memory, the manufacturer should pre-load this memory with the applicable value from point “a”, “b”, or “d” above.

The values from point “a”, “b”, or “d” above are called the dynamic data set initialization values.

8.1.3.4 Initialization of configurable data set members

Configurable members shall be initialized to the first one of the following values that applies:

a) The value indicated in the member specifications as the initialization value not subject to PTP profile specification,

b) Last value configured by management messages or implementation-specific means and maintained in non-volatile read-write storage if implemented,

c) The default initialization value for the member as specified in the PTP profile implemented by the device.

If the configured values are maintained in non-volatile read-write memory, the manufacturer should pre-load this memory with the applicable initialization value from point “a” or “c” above.

The values from point “a” or “c” above are called the configurable data set initialization values.

8.1.3.5 Operation of non-volatile read-write storage of data set members

The current values of one or more dynamic or configurable data set members may be maintained in non-volatile read-write storage.

NOTE—For example, this may allow more rapid configuration of systems after a power outage.

The contents of non-volatile read-write memory shall be reset to the applicable dynamic or configurable data set initialization values, 8.1.3.3 and 8.1.3.4, upon receipt of the management message RESET_NON_VOLATILE_STORAGE or the implementation-specific equivalent.

The current values of the applicable dynamic and configurable data set members shall be copied into non-volatile read-write memory:

- On receipt of the management message SAVE_IN_NON_VOLATILE_STORAGE or the implementation-specific equivalent, or
- At implementation-specific times.

8.2 Data sets for ordinary and boundary clocks

8.2.1 defaultDS data set member specifications

8.2.1.1 General

The members of this data set are:

- defaultDS.twoStepFlag

- defaultDS.clockIdentity
- defaultDS.numberPorts
- defaultDS.clockQuality
- defaultDS.priority1
- defaultDS.priority2
- defaultDS.domainNumber
- defaultDS.slaveOnly

8.2.1.2 Static members of the defaultDS data set

8.2.1.2.1 defaultDS.twoStepFlag

The value of defaultDS.twoStepFlag shall be TRUE if the clock is a two-step clock; otherwise the value shall be FALSE.

8.2.1.2.2 defaultDS.clockIdentity

The value of defaultDS.clockIdentity shall be the clockIdentity, see 7.6.2.1, of the local clock.

8.2.1.2.3 defaultDS.numberPorts

The value of defaultDS.numberPorts shall be the number of PTP ports on the device. For an ordinary clock this shall be the value 1.

8.2.1.3 Dynamic members of the defaultDS data set

8.2.1.3.1 defaultDS.clockQuality

8.2.1.3.1.1 defaultDS.clockQuality.clockClass

The value of defaultDS.clockQuality.clockClass shall follow the clockClass specifications of 7.6.2.4.

The initialization value of defaultDS.clockQuality.clockClass shall be selected as follows:

a) The value is dependent on the initialization value of the defaultDS data set member defaultDS.slaveOnly, see 8.2.1.4.4, which shall be initialized prior to initialization of the defaultDS.clockQuality.clockClass member.

b) If defaultDS.slaveOnly is TRUE, the initialization value shall be 255 as specified in 7.6.2.4.

c) If defaultDS.slaveOnly is FALSE and a PTP profile specifies the clockClass to be 52, 58, 187, 193, or in the ranges 68 through 122, 133 through 170, or 216 through 232, then the PTP profile specified clockClass value shall be used for initialization.

d) If defaultDS.slaveOnly is FALSE and if the device is designed as a clockClass 6 or 13, the clockClass initialization value shall be 6 or 13 respectively if these represent the clockClass of the clock upon exiting the INITIALIZING state. If the clockClass 6 or 13 respectively does not represent the clock upon exiting the INITIALIZING state, the clockClass initialization value shall be:

    1) Either 52, 187 or 248, as specified in a PTP profile, for a clock designed as class 6

2) Either 58, 193 or 248, as specified in a PTP profile, for a clock designed as class 13

e) Else, the value shall be 248.

NOTE—A clock that is designed to be clockClass 6 or 13 can include implementation-specific measures to ensure it meets the specifications of 7.6.2.4 for these clockClasses before initializing the clockClass member. For example, the clock can synchronize to the GPS system as part of the activities in the INITIALIZING state prior to initializing the clockClass member. If such a clock is unable to meet the specifications of 7.6.2.4 prior to leaving the INITIALIZING state then one of the degradation or the default alternatives for the clock is used.

8.2.1.3.1.2 defaultDS.clockQuality.ClockAccuracy

The value of defaultDS.clockQuality.ClockAccuracy is the clockAccuracy member of the defaultDS.clockQuality member, see 5.3.7.

The initialization value of defaultDS.clockQuality.ClockAccuracy shall be selected as follows:

a) The value is dependent on the initialization value of the defaultDS.clockQuality.clockClass, see 8.2.1.3.1.1, which shall be initialized prior to initialization of clockAccuracy.

b) The clockAccuracy initialization value shall represent the clockAccuracy of the clock at the time of initialization as specified in 7.6.2.5.

8.2.1.3.1.3 defaultDS.OffsetScaledLogVariance

The value of defaultDS.OffsetScaledLogVariance is the offsetScaledLogVariance member of the defaultDS.clockQuality member, see 5.3.7.

The initialization value of the offsetScaledLogVariance shall reflect the inherent characteristics of the local clock at the time of initialization as specified in 7.6.3.5.

8.2.1.4 Configurable members of the defaultDS data set

8.2.1.4.1 defaultDS.priority1

The value of defaultDS.priority1 is the priority1 attribute, see 7.6.2.2, of the local clock.

8.2.1.4.2 defaultDS.priority2

The value of defaultDS.priority2 is the priority2 attribute, see 7.6.2.3, of the local clock.

8.2.1.4.3 defaultDS.domainNumber

The value of defaultDS.domainNumber is the domain attribute, see 7.1, of the local clock.

8.2.1.4.4 defaultDS.slaveOnly

The value of defaultDS.slaveOnly shall be TRUE if the clock is a slave-only clock, see 9.2.2. The value shall be FALSE if the clock is a non-slave-only clock, see 9.2.3.

8.2.2 currentDS data set member specifications

8.2.2.1 General

The members of this data set are:

- currentDS.stepsRemoved
- currentDS.offsetFromMaster
- currentDS.meanPathDelay

All members of the currentDS data set are dynamic.

8.2.2.2 currentDS.stepsRemoved

The value of currentDS.stepsRemoved is the number of communication paths traversed between the local clock and the grandmaster clock.

The initialization value shall be 0.

NOTE—For example, currentDS.stepsRemoved for a slave clock on the same PTP communication path as the grandmaster clock has a value of 1, indicating that a single path was traversed

8.2.2.3 currentDS.offsetFromMaster

The value of currentDS.offsetFromMaster is an implementation-specific representation of the current value of the time difference between a master and a slave as computed by the slave, see 11.2. The data type should be TimeInterval. The initialization value shall be either:

- The value in non-volatile read-write storage if implemented, or
- Implementation-specific.

8.2.2.4 currentDS.meanPathDelay

The value of currentDS.meanPathDelay is an implementation-specific representation of the current value of the mean propagation time between a master and slave clock as computed by the slave, see 11.2. The data type should be TimeInterval. The initialization value shall be either:

- The value in non-volatile read-write storage if implemented, or
- Implementation-specific.

8.2.3 parentDS data set member specifications

8.2.3.1 General

The members of this data set are:

- parentDS.parentPortIdentity
- parentDS.parentStats
- parentDS.observedParentOffsetScaledLogVariance
- parentDS.observedParentClockPhaseChangeRate
- parentDS.grandmasterIdentity
- parentDS.grandmasterClockQuality
- parentDS.grandmasterPriority1

- parentDS.grandmasterPriority2

The values of the parentDS data set shall be initialized after the values in the defaultDS data set.

All members of the parentDS data set are dynamic.

8.2.3.2 parentDS.parentPortIdentity

The value of parentDS.parentPortIdentity is the source portIdentity of the port on the master that issues the Sync messages used in synchronizing this clock.

The initialization value shall be:

- The parentDS.parentPortIdentity.clockIdentity member is the value of the clockIdentity field of the defaultDS data set.
- The parentDS.portNumber member is 0.

8.2.3.3 parentDS.parentStats

The value of parentDS.parentStats shall be TRUE if all of the following conditions are satisfied:

- The clock has a port in the SLAVE state.
- The clock has computed statistically valid estimates of the observedParentOffsetScaledLogVariance and observedParentClockPhaseChangeRate members.

Otherwise the value shall be FALSE.

The initialization value shall be FALSE.

8.2.3.4 parentDS.observedParentOffsetScaledLogVariance

The value of parentDS.observedParentOffsetScaledLogVariance shall be an estimate of the parent clock's PTP variance as observed by the slave clock, computed and represented as described in 7.6.3.5. The computation of this value is optional but, if not computed, the value of parentDS.parentStats shall be FALSE.

The initialization value shall be FFFF_{16}, see 7.6.3.3.

8.2.3.5 parentDS.observedParentClockPhaseChangeRate

The value of parentDS.observedParentClockPhaseChangeRate shall be an estimate of the parent clock's phase change rate as observed by the slave clock as defined in 7.6.4.4. If the estimate exceeds the capacity of its data type, see 15.5.3.5.1.4, this value shall be set to 7FFF FFFF_{16} or 8000 0000_{16}, as appropriate. A positive sign indicates that the parent clock's phase change rate is greater than the rate of the slave clock. The computation of this value is optional but, if not computed, the value of parentDS.parentStats shall be FALSE.

The initialization value shall be 7FFF FFFF_{16} irrespective of whether the computation is implemented in the local clock.

A value equal to 7FFF FFFF_{16} indicates that either the value exceeds the capacity of the data type or that the value has not been computed.

NOTE— This value is dependent on the measurement time interval used. When this value is critical for an application domain, the time interval should be specified in the applicable PTP profile.

8.2.3.6 parentDS.grandmasterIdentity

The value of parentDS.grandmasterIdentity is the clockIdentity attribute, see 7.6.2.1, of the grandmaster clock.

The initialization value shall be the clockIdentity member of the defaultDS data set.

8.2.3.7 parentDS.grandmasterClockQuality

The value of parentDS.grandmasterClockQuality is the clock quality attribute, see 7.6.2.4, 7.6.2.5, and 7.6.3, of the grandmaster clock.

The initialization value shall be the value of the defaultDS.clockQuality member.

8.2.3.8 parentDS.grandmasterPriority1

The value of parentDS.grandmasterPriority1 is the priority1 attribute, see 7.6.2.2, of the grandmaster clock.

The initialization value shall be the value of the defaultDS.priority1 member.

8.2.3.9 grandmasterPriority2

The value of grandmasterPriority2 is the priority2 attribute, see 7.6.2.3, of the grandmaster clock.

The initialization value shall be the value of the parentDS.priority2 member.

8.2.4 timePropertiesDS data set member specifications

8.2.4.1 timePropertiesDS data set members

The members of this data set are:

- timePropertiesDS.currentUtcOffset
- timePropertiesDS.currentUtcOffsetValid
- timePropertiesDS.leap59
- timePropertiesDS.leap61
- timePropertiesDS.timeTraceable
- timePropertiesDS.frequencyTraceable
- timePropertiesDS.ptpTimescale
- timePropertiesDS.timeSource

All members of the currentDS data set are dynamic.

The timePropertiesDS.ptpTimescale member shall be initialized before the other members of this data set.

8.2.4.2 timePropertiesDS.currentUtcOffset

In PTP systems whose epoch is the PTP epoch, the value of timePropertiesDS.currentUtcOffset is the offset between TAI and UTC; otherwise the value has no meaning. The value shall be in units of seconds.

The initialization value shall be selected as follows:

a) If the timePropertiesDS.ptpTimescale, see 8.2.4.8, is TRUE the value is the value obtained from a primary reference if the value is known at the at the time of initialization, else

b) The value shall be the current number of leap seconds, 7.2.3, when the node is designed.

NOTE— A clock that is designed to be clockClass 6 can include implementation-specific measures to ensure it meets the specifications of 7.6.2.4 for clockClass 6 and therefore has access to the UTC offset value before initializing the timePropertiesDS.currentUtcOffset member. For example, the clock can synchronize to the GPS system as part of the activities in the INITIALIZATION state prior to initializing the currentUtcOffset member.

8.2.4.3 timePropertiesDS.currentUtcOffsetValid

The value of timePropertiesDS.currentUtcOffsetValid is TRUE if the timePropertiesDS.currentUtcOffset is known to be correct.

The initialization value shall be TRUE if the value of timePropertiesDS.currentUtcOffset is known to be correct, otherwise it shall be FALSE.

8.2.4.4 timePropertiesDS.leap59

In PTP systems whose epoch is the PTP epoch, a TRUE value for timePropertiesDS.leap59 shall indicate that the last minute of the current UTC day contains 59 seconds.

If the epoch is not PTP, the value shall be set to FALSE.

The initialization value shall be selected as follows:

a) If the timePropertiesDS.ptpTimescale, see 8.2.4.8, is TRUE the value shall be the value obtained from a primary reference if known at the at the time of initialization, else

b) The value shall be FALSE.

8.2.4.5 timePropertiesDS.leap61

In PTP systems whose epoch is the PTP epoch, a TRUE value for timePropertiesDS.leap61 shall indicate that the last minute of the current UTC day contains 61 seconds.

If the epoch is not PTP, the value shall be set to FALSE.

The initialization value shall be selected as follows:

a) If the timePropertiesDS.ptpTimescale, see 8.2.4.8, is TRUE the value is the value obtained from a primary reference if known at the at the time of initialization, else

b) The value shall be FALSE.

8.2.4.6 timePropertiesDS.timeTraceable

The value of timePropertiesDS.timeTraceable is TRUE if the timescale and the value of timePropertiesDS.currentUtcOffset are traceable to a primary reference; otherwise the value shall be FALSE.

The initialization value shall be selected as follows:

a) If the timePropertiesDS.ptpTimescale, see 8.2.4.8, is TRUE and the time and the value of timePropertiesDS.currentUtcOffset are traceable to a primary reference at the time of initialization, the value shall be TRUE, else

b) The value shall be FALSE.

8.2.4.7 timePropertiesDS.frequencyTraceable

The value of timePropertiesDS.frequencyTraceable is TRUE if the frequency determining the timescale is traceable to a primary reference; otherwise the value shall be FALSE.

The initialization value shall be selected as follows:

a) If the frequency is traceable to a primary reference at the time of initialization the value shall be TRUE, else

b) The value shall be FALSE.

8.2.4.8 timePropertiesDS.ptpTimescale

The value of timePropertiesDS.ptpTimescale is TRUE if the clock timescale of the grandmaster clock, see 7.2.1, is PTP and FALSE otherwise.

The initialization value shall be selected as follows:

a) If the clock timescale, see 7.2.1, is PTP and this is known at the time of initialization the value shall be set to TRUE, else

b) The value shall be FALSE, indicating that the timescale is ARB.

8.2.4.9 timePropertiesDS.timeSource

The value of timePropertiesDS.timeSource is the source of time used by the grandmaster clock.

The initialization value shall be selected as follows:

a) If the timeSource, see 7.6.2.6, is known at the time of initialization the value shall be set to that value, else

b) The value shall be INTERNAL_OSCILLATOR.

8.2.5 portDS data set member specifications

8.2.5.1 Port data set members

For the single port of an ordinary clock and for each port of a boundary clock, the following ‘port data set’ shall be maintained as the basis for protocol decisions and providing values for message fields. The number of such data sets shall be the value of defaultDS.numberPorts.

The members of this data set are:

- portDS.portIdentity
- portDS.portState
- portDS.logMinDelayReqInterval
- portDS.peerMeanPathDelay
- portDS.logAnnounceInterval
- portDS.announceReceiptTimeout
- portDS.logSyncInterval
- portDS.delayMechanism
- portDS.logMinPdelayReqInterval

— portDS.versionNumber.

8.2.5.2 Static members of the portDS data set

8.2.5.2.1 portDS.portIdentity

The value of portDS.portIdentity shall be the PortIdentity attribute of the local port, see 7.5.2.

8.2.5.3 Dynamic members of the portDS data set

8.2.5.3.1 portDS.portState

The value of portDS.portState shall be the value of the current state of the protocol engine associated with this port, see 9.2, and shall be taken from the enumeration in Table 8.

Table 8: PTP state enumeration

PTP state enumeration	Value
INITIALIZING	1
FAULTY	2
DISABLED	3
LISTENING	4
PRE_MASTER	5
MASTER	6
PASSIVE	7
UNCALIBRATED	8
SLAVE	9
—	All other values reserved

The initialization value shall be INITIALIZING.

8.2.5.3.2 portDS.logMinDelayReqInterval

The value of portDS.logMinDelayReqInterval is the logarithm to the base 2 of the minDelayReqInterval, see 7.7.2.4. The initialization value is implementation-specific consistent with 7.7.2.4.

8.2.5.3.3 portDS.peerMeanPathDelay

If the value of the portDS.delayMechanism member is peer-to-peer (P2P), the value of portDS.peerMeanPathDelay shall be an estimate of the current one-way propagation delay on the link attached to this port computed using the peer delay mechanism, see 11.4. The data type should be TimeInterval. If the value of the portDS.delayMechanism member is end-to-end (E2E), this member's value shall be zero. The initialization value shall be zero.

8.2.5.4 Configurable members of the portDS data set

8.2.5.4.1 portDS.logAnnounceInterval

The value of portDS.logAnnounceInterval shall be the logarithm to the base 2 of the of the mean announceInterval, see 7.7.2.2.

8.2.5.4.2 portDS.announceReceiptTimeout

The value of portDS.announceReceiptTimeout shall be an integral multiple of announceIntervals, see 7.7.3.1.

NOTE—The announceInterval is equal to the value of $2^{logAnnounceInterval}$.

8.2.5.4.3 portDS.logSyncInterval

The value of portDS.logSyncInterval shall be the logarithm to the base 2 of the mean Sync interval for multicast messages, see 7.7.2.3.

NOTE—The rates for unicast transmissions are negotiated separately on a per port basis and are not constrained by 8.2.5.4.3.

8.2.5.4.4 portDS.delayMechanism

The value of portDS.delayMechanism shall indicate the propagation delay measuring option used by the port in computing meanPathDelay. The value shall be taken from the enumeration in Table 9. The initialization value is implementation-specific unless otherwise stated in a PTP profile.

Table 9: Delay mechanism enumeration

Delay mechanism	**Value$_{16}$**	**Specification**
E2E	01	The port is configured to use the delay request-response mechanism
P2P	02	The port is configured to use the peer delay mechanism
DISABLED	FE	The port does not implement the delay mechanism, see Note.
NOTE— This value shall not be set by a clock except when the applicable PTP profile specifies that the clock syntonize only and that neither path delay mechanism is to be used.		

NOTE—Subclause 9.1 permits reconfiguration. Auto-configuration is allowed but is out of scope.

8.2.5.4.5 portDS.logMinPdelayReqInterval

The value of portDS.logMinPdelayReqInterval shall be the logarithm to the base 2 of the minimum mean Pdelay_Req interval, see 7.7.2.5.

8.2.5.4.6 portDS.versionNumber

The value of portDS.**versionNumber** shall indicate the PTP version in use on the port.

8.3 Data sets for transparent clocks

8.3.1 General

Optionally, a transparent clock shall maintain a single copy of each the default and port data sets.

NOTE—Unlike ordinary and boundary clocks a transparent clock does not maintain separate data sets for each domain. With the exception of syntonization, transparent clocks are domain independent.

8.3.2 transparentClockDefaultDS data set member specifications

8.3.2.1 General

The members of this data set are:

- transparentClockDefaultDS.clockIdentity
- transparentClockDefaultDS.numberPorts
- transparentClockDefaultDS.delayMechanism
- transparentClockDefaultDS.primaryDomain

8.3.2.2 Static members of the transparentClockDefaultDS data set

8.3.2.2.1 transparentClockDefaultDS.clockIdentity

The value of transparentClockDefaultDS.clockIdentity shall be the clockIdentity attribute, see 7.6.2.1, of the local clock.

8.3.2.2.2 transparentClockDefaultDS.numberPorts

The value of transparentClockDefaultDS.numberPorts shall be the number of PTP ports of the device.

8.3.2.3 Configurable members of the transparentClockDefaultDS data set

8.3.2.3.1 transparentClockDefaultDS.delayMechanism

If the transparent clock is an end-to-end transparent clock, the value of transparentClockDefaultDS.delayMechanism shall be E2E, see Table 9. If the transparent clock is a peer-to-peer transparent clock, the value shall be P2P.

8.3.2.3.2 transparentClockDefaultDS.primaryDomain

The value of transparentClockDefaultDS.primaryDomain shall be the domain number of the primary syntonization domain, see 10.1. The initialization value shall be 0.

8.3.3 transparentClockPortDS data set member specifications

8.3.3.1 General

The members of this data set are:

- transparentClockPortDS.portIdentity
- transparentClockPortDS.logMinPdelayReqInterval
- transparentClockPortDS.faultyFlag
- transparentClockPortDS.peerMeanPathDelay

8.3.3.2 Static members of the portDS data set

8.3.3.2.1 transparentClockPortDS.portIdentity

The value of transparentClockPortDS.portIdentity shall be the PortIdentity attribute of the local port, see 7.5.2.

8.3.3.3 Dynamic members of the portDS data set

8.3.3.3.1 transparentClockPortDS.logMinPdelayReqInterval

The value of transparentClockPortDS.logMinPdelayReqInterval shall be the logarithm to the base 2 of the minimum of the mean value of the Pdelay_Req interval, see 7.7.2.5.

8.3.3.3.2 transparentClockPortDS.faultyFlag

The value of transparentClockPortDS.faultyFlag shall be TRUE if the port is faulty, and FALSE if the port is operating normally. The initialization value shall be FALSE.

8.3.3.3.3 transparentClockPortDS.peerMeanPathDelay

If the value of the transparentClockDefaultDS.delayMechanism member is P2P, the value of transparentClockPortDS.peerMeanPathDelay shall be the estimate of the current one-way propagation delay on the link attached to this port computed using the peer delay mechanism, see 11.4. If the value of the transparentClockDefaultDS.delayMechanism member is E2E, the value shall be zero. The data type should be TimeInterval. The initialization value shall be zero.

9. PTP for ordinary and boundary clocks

9.1 General protocol requirements for PTP ordinary and boundary clocks

Ordinary and boundary clocks:

a) May operate within more than one domain, see 7.1. The operation of each domain shall be independent of the others.

b) When required by the state machine of 9.2, shall synchronize per 12.2.

c) Shall correct for path delay using one of the following options:

   1) Delay-response mechanism, see 11.3, or

   2) Peer delay mechanism, see 11.4.

A port may implement both the Delay-response and the Peer-delay mechanisms provided only one mechanism is active at any time. The method of selection is outside the scope of this standard. Auto-configuration is not prohibited.

Clocks synchronize only to the clock selected using the best master algorithm.

An ordinary or boundary clock that receives an Announce, Sync, Follow_Up, or Delay_Resp message in which the value of the header flag, alternateMasterFlag, is TRUE shall disregard the message except as provided in Clause 17.

An ordinary clock shall contain a single PTP port obeying the requirements of 9.2.

A boundary clock shall contain multiple PTP ports each obeying the requirements of 9.2.

9.2 State protocol

9.2.1 General state requirements

All ordinary and boundary clocks shall implement the state machine and state behavior of 9.2.

9.2.2 Slave-only ordinary clocks

An ordinary clock may be designed to be a slave-only or a non-slave-only clock. An implementation may optionally provide the ability to configure to a slave-only mode via the management message SLAVE_ONLY or by implementation dependent means. A slave-only clock shall implement the state machine illustrated in Figure 24.

NOTE—A slave-only clock can never enter the MASTER state. Systems should therefore contain at least one non-slave-only clock. A slave-only clock uses a different state machine than a non-slave-only clock and has a different clockClass number, see 7.6.2.4.

9.2.3 Non-slave-only clocks

A boundary clock shall not be a slave-only clock. Boundary clocks and ordinary clocks not designed or configured as slave-only shall implement the state machine illustrated in Figure 23.

9.2.4 State definitions

The behavior of the states of a port associated with the state machines of Figure 23 and Figure 24 shall be as defined in Table 10 with the exception of the provisions for unicast messages specified in 16.1.1.

Table 10: PTP portState definition

PTP portState	Description
INITIALIZING	While a port is in the INITIALIZING state, the port initializes its data sets, hardware, and communication facilities. No port of the clock shall place any PTP messages on its communication path. If one port of a boundary clock is in the INITIALIZING state then all ports shall be in the INITIALIZING state
FAULTY	The fault state of the protocol. A port in this state shall not place any PTP messages except for management messages that are a required response to another management message on its communication path. In a boundary clock, no activity on a faulty port shall affect the other ports of the device. If fault activity on a port in this state cannot be confined to the faulty port then all ports shall be in the FAULTY state.
DISABLED	The port shall not place any messages on its communication path. In a boundary clock, no activity at the port shall be allowed to affect the activity at any other port of the boundary clock. A port in this state shall disregard all PTP received messages except for management messages.
LISTENING	The port is waiting for the announceReceiptTimeout to expire or to receive an Announce message from a master. The purpose of this state is to allow orderly addition of clocks to a domain. A port in this state shall not place any PTP messages on its communication path except for Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up, or signaling messages, or management messages that are a required response to another management message.
PRE_MASTER	The port shall behave in all respects as though it were in the MASTER state except that it shall not place any messages on its communication path except for Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up, signaling or management messages.
MASTER	The port is behaving as a master port.
PASSIVE	The port shall not place any messages on its communication path except for Pdelay_Req, Pdelay_Resp, Pdelay_Resp_Follow_Up, or signaling messages, or management messages that are a required response to another management message.
UNCALIBRATED	One or more master ports have been detected in the domain. The appropriate master port has been selected and the local port is preparing to synchronize to the selected master port. This is a transient state to allow initialization of synchronization servos, updating of data sets when a new master port has been selected, and other implementation-specific activity.
SLAVE	The port is synchronizing to the selected master port.

With the exception of the DISABLED state, a port may make use of any information received in PTP messages provided such use does not violate the requirements of the protocol.

9.2.5 State machines

The state machines of 9.2.5 shall determine the allowed transitions for PTP stateful ports.
Subclause 4.3 specifies a notation used in Figure 23 and Figure 24.

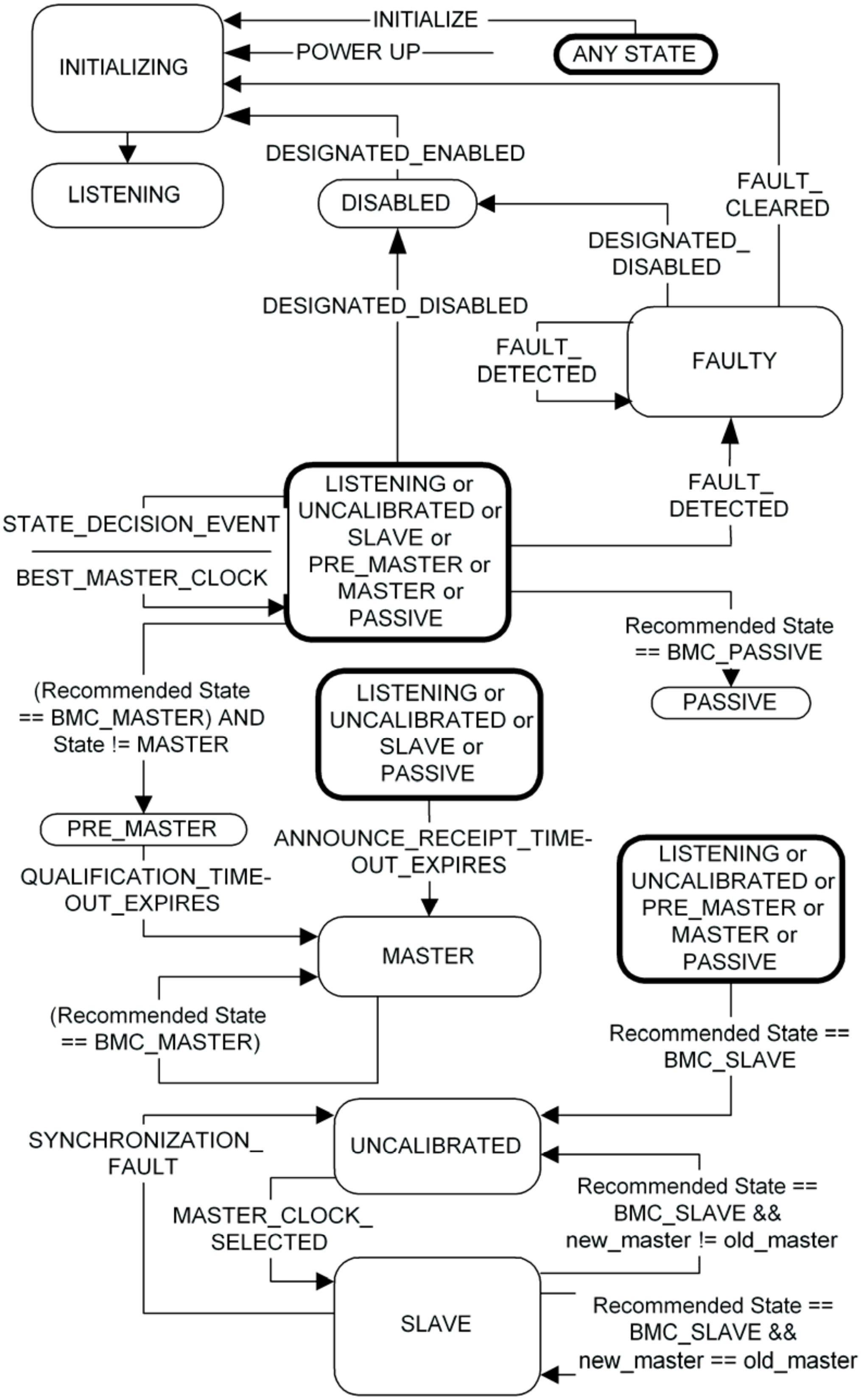


Figure 23: State machine for a full implementation

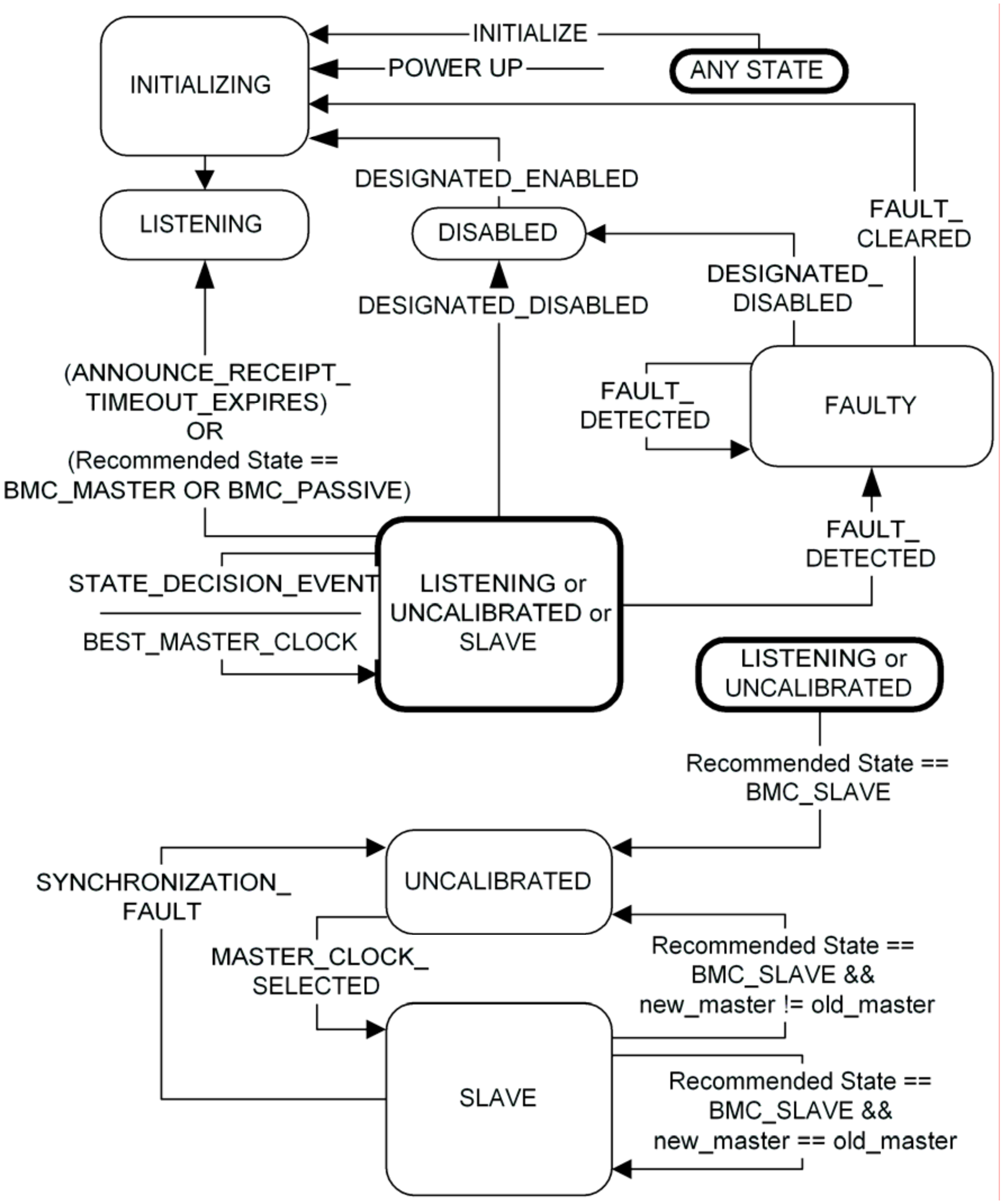


Figure 24: State machine for a slave-only implementation

In both Figure 23 and Figure 24, when a fault is cleared or a previously disabled port is enabled, the state machine makes a transition to the INITIALIZING state.

NOTE—Before making the subsequent transition to the LISTENING state, an implementation is not required to perform all the steps that it would after an INITIALIZE command or a power-up or reset. It is required to achieve the effect of performing those steps.

9.2.6 Events initiating PTP state transitions

9.2.6.1 General

The specifications for each event initiating a transition in the state machines of Figure 23 and Figure 24 are defined in 9.2.6.

9.2.6.2 POWERUP

The POWERUP event shall be instantiated by turning on the power to the device. It may also be instantiated by implementation-specific reset mechanisms, e.g. a reset button.

9.2.6.3 INITIALIZE

The INITIALIZE event shall be instantiated by the receipt of the INITIALIZE management message or its equivalent if required by the initializationKey field of this message.

9.2.6.4 DESIGNATED_ENABLED

The DESIGNATED_ENABLED event shall be instantiated by the receipt of the ENABLE_PORT management message

9.2.6.5 DESIGNATED_DISABLED

The DESIGNATED_DISABLED event shall be instantiated by the receipt of the DISABLE_PORT management message

9.2.6.6 FAULT_CLEARED

The FAULT_CLEARED event shall be instantiated by the clearing of the fault condition or conditions that prevents correct operation of the port.

NOTE—The clearing of all detected fault conditions may result from the actions of management messages or internal procedures.

9.2.6.7 FAULT_DETECTED

The FAULT_DETECTED event shall be instantiated by the occurrence of an internal condition that prevents the correct operation of the port.

9.2.6.8 STATE_DECISION_EVENT

The STATE_DECISION_EVENT is the mechanism for using the data in received Announce messages to determine which is the best master clock, and whether the local clock port or ports needs to change state. Every clock shall implement a mechanism generating the STATE_DECISION_EVENT, and the occurrence of this event shall implement the logic illustrated in Figure 25. In Figure 25, the best master clock algorithm is illustrated as the default best master clock algorithm specified in this standard. If an optional best master clock algorithm is specified, see 9.3.1, the portion of the figure within the box labeled best master clock algorithm may differ.

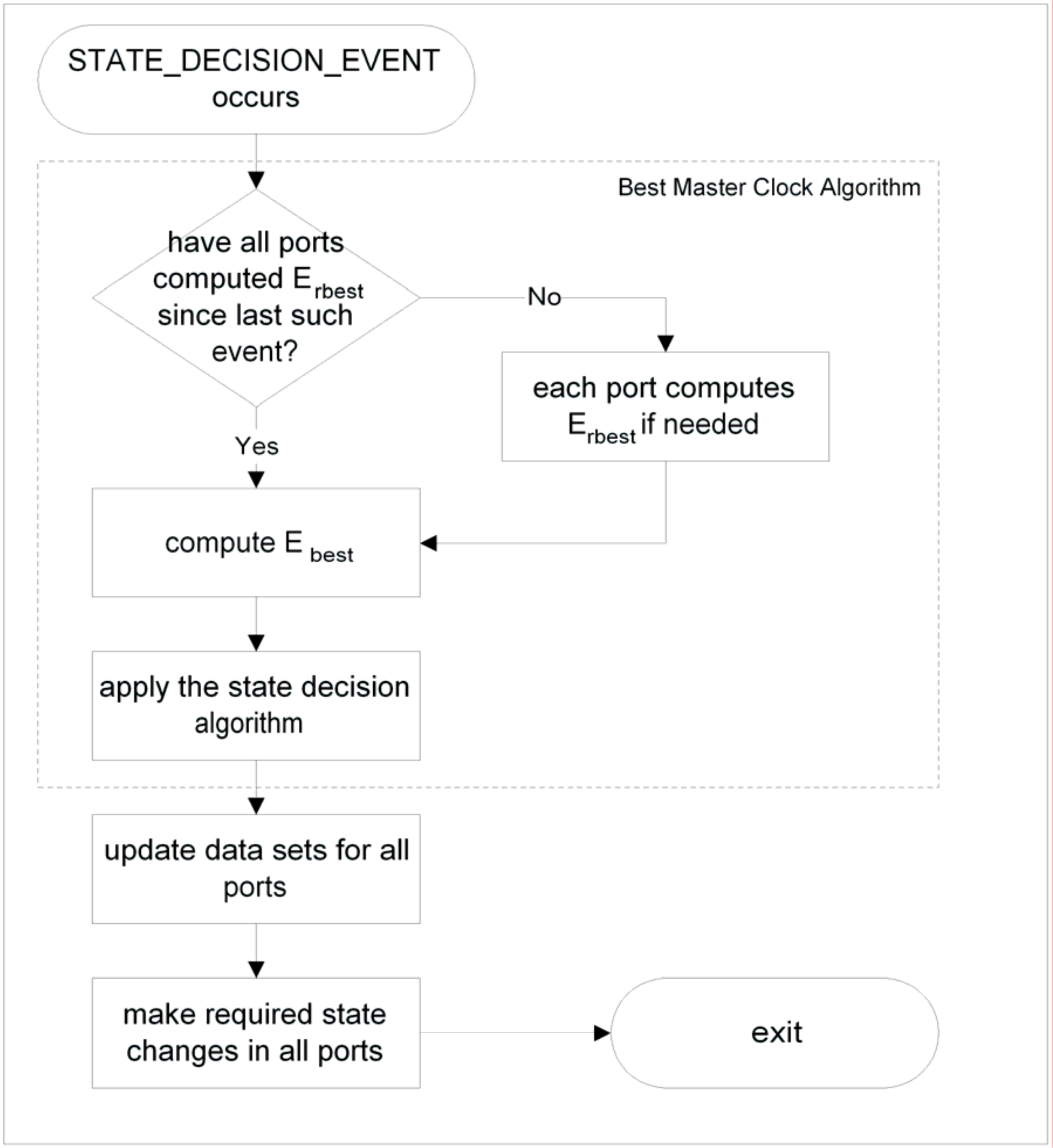

STATE_DECISION_EVENT occurs
Best Master Clock Algorithm
have all ports computed E_{rbest} since last such event?
No
Yes
each port computes E_{rbest} if needed
compute E_{best}
apply the state decision algorithm
update data sets for all ports
make required state changes in all ports
exit

Figure 25: STATE_DECISION_EVENT logic

The STATE_DECISION_EVENT shall:

- Logically occur simultaneously on all ports of a clock, and
- Occur at least once per Announce message transmission interval, and
- Not occur when any port is in the INITIALIZING state.

For nodes implementing the default best master clock algorithm, see 9.3.1, prior to or as the first action of the STATE_DECISION_EVENT logic, each port N not in the DISABLED or FAULTY states shall compute an updated value of E_{rbest}, see 9.3.2.3, reflecting the receipt of Announce messages since the last STATE_DECISION_EVENT. Following the computation of the set of E_{rbest} for all ports, the clock shall compute E_{best}.

For all nodes, the clock shall complete the following tasks, in the order given:

a) Apply the best master clock algorithm,

b) Update the appropriate data sets,

c) Instantiate the recommended state event in the state machine,

d) Make the required state changes in all ports.

These tasks shall be carried out atomically, see 3.1.2. This input information shall include the set of E_{rbest} values. These tasks shall be executed as defined in 9.3.

9.2.6.9 Recommended state

The recommended state event results from the exercise of the best master clock algorithm initiated by a STATE_DECISION_EVENT.

9.2.6.10 QUALIFICATION_TIMEOUT_EXPIRES

The expiration of the <qualificationTimeout interval> defines the QUALIFICATION_TIMEOUT_EXPIRES event. This timeout mechanism determines the interval a clock spends in the PRE_MASTER state.

The <qualificationTimeout interval> shall start when the port enters the PRE_MASTER state. The expiration shall occur after an interval computed as follows:

The interval shall be for N multiplied by the announce message interval, see 7.7.2.2, in seconds, where:

a) If the recommended state = MASTER event was based on decision points M1 or M2 of Figure 26, N shall be 0,

b) If the recommended state = MASTER event was based on decision point M3 of Figure 26, N shall be the value incremented by 1 (one) of the stepsRemoved field of the currentDS data set.

9.2.6.11 ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES

Each protocol engine shall support a timeout mechanism defining a timeout with an interval announceReceiptTimeout announceInterval, see 7.7.3.1.

The ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES event occurs at the expiration of this timeout plus a random number uniformly distributed in the range (0,1) announceIntervals.

This timeout interval for a port shall start or be restarted when any of the following occur:

a) For a port in the UNCALIBRATED or SLAVE states, when an Announce message is received from the parent clock as indicated by the parentPortIdentity of the parentDS data set, or

b) At the expiration of the current announceReceiptTimeout interval unless otherwise specified in Clause 9, or

c) When entering the LISTENING, UNCALIBRATED, SLAVE, or PASSIVE state, or

d) For a port in the PASSIVE state when an Announce message is received from the clock that transmitted the announce message that caused the port to be in the PASSIVE state, as indicated by a comparison of the sourcePortIdentity fields of the respective messages.

This timeout mechanism for a port shall be stopped and not restarted when entering the INITIALIZING, PRE_MASTER, FAULTY, DISABLED or the MASTER states.

In addition to the state changes of Figure 23 and Figure 24, ports in the LISTENING, PASSIVE, UNCALIBRATED, or SLAVE states when the ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES event occurs shall update data sets prior to entering the MASTER state as follows:

- For an ordinary clock, update the port's data sets to the MASTER state configuration, see 9.3.5, as specified for decision M1,
- For a boundary clock with no other port in the SLAVE state, update the port's data sets to the MASTER state configuration, see 9.3.5, as specified for decision M1,
- For a boundary clock with a different port in the SLAVE state, update the port's data sets to the MASTER state configuration, see 9.3.5, as specified for decision M3.

9.2.6.12 SYNCHRONIZATION_FAULT

SYNCHRONIZATION_FAULT event instantiation is implementation-specific. This event should be instantiated whenever a clock is in the SLAVE state and the implementation detects implementation circumstances that require re-execution of functions that occur in the UNCALIBRATED state to ensure correct synchronization.

9.2.6.13 MASTER_CLOCK_SELECTED

MASTER_CLOCK_SELECTED event instantiation is implementation-specific. This event should be instantiated whenever a clock in the UNCALIBRATED state has satisfied all implementation and protocol mandated requirements necessary to ensure synchronization when in the SLAVE state.

9.2.7 Applying PTP events to the ports of a boundary clock

For a boundary clock, the events initiating a transition in the state machines of Figure 23 and Figure 24, see 9.2.6, shall be applied to the device's port state machines as specified in Table 11.

Table 11: Event applicability in boundary clocks

Event name	**Port applicability**
POWERUP	All ports
INITIALIZE	All ports
FAULT_DETECTED	All ports affected by the fault
FAULT_CLEARED	All ports affected by the fault
STATE_DECISION_EVENT	All ports.
Recommended state, see NOTE	All ports.
ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES or QUALIFICATION_TIMEOUT_EXPIRES	The port signaling the expiring timeout mechanism.
DESIGNATED_ENABLED or DESIGNATED_DISABLED	The ports specified by the initiating management message.
MASTER_CLOCK_SELECTED	The port signaling the event.
SYNCHRONIZATION_FAULT	The port signaling the event.
NOTE—recommended state is a condition resulting from the exercise of the best master clock algorithm initiated by a STATE_DECISION_EVENT	

9.3 Best master clock algorithm

9.3.1 Selection of the best master clock algorithm

PTP permits the use of two forms of best master clock algorithm:

— By default, the mechanism specified in 9.3.2, 9.3.3, and 9.3.4 or

— If specified in a PTP profile, an alternate best master clock algorithm.

Any alternate best master clock algorithm shall meet the following requirements:

a) The output of the algorithm shall provide the recommended state required for operation of the PTP state machines and state decision events of 9.2.5, 9.2.6.8, and 9.2.6.9. The recommended states shall meet the requirements of 9.2.4. Optionally, the alternate algorithm may be a dynamic algorithm, or it may be a static algorithm that simply configures the recommended state values on the ports of the node on which it is running.

b) The output of the algorithm shall provide the state decision codes for use in updating data sets, see 9.3.5, and any data required for implementing the updates based on these codes. These decision codes shall be as follows:

   1) M1: The port is in the MASTER state because it is on a clockClass 1 through 127 node and is the grandmaster port of the system.

   2) M2: The port is in the MASTER state because it is on a clockClass 128 or higher node and is the grandmaster port of the system

   3) M3: The port is in the MASTER state but it is not a port on the grandmaster clock of the system

   4) S1: The port is in the SLAVE state.

   5) P1: The port is in the PASSIVE state because it is on a clockClass 1 through 127 node and is either not on the grandmaster clock of the system or is PASSIVE to break a timing loop.

   6) P2: The port is in the PASSIVE state because it is on a clockClass 128 or higher node and is PASSIVE to break a timing loop

The BMC algorithm is run locally on all ports of every ordinary and boundary clock in a domain. Since it runs continually, it continually re-adapts to changes in the network or the clocks.

9.3.2 BMC algorithm

9.3.2.1 Overview and definition of terms

Subclause 9.3.2 specifies the way that a local clock determines which of all the visible clocks (including itself) is the “best.” From that it determines the next state, see Table 10. The algorithm runs independently on each clock in a domain. In other words, clocks do not negotiate which should be master and which should be slave — instead, each computes only its own state. The algorithm avoids configurations with two masters, or none, or ones that thrash.

NOTE— A port is visible to port-A if it is transmitting Announce messages that are received by port-A.

The best master clock algorithm BMC consists of two parts:

a) A data set comparison algorithm that determines which of two clock ports is better. This algorithm is specified in 9.3.4.

b) An algorithm that computes a recommendation for the state of each port involved. This algorithm is specified in 9.3.3. The recommendation is called “recommended state” in Figure 23, Figure 24, and Figure 26.

This determination is based on information contained in Announce messages received at the ports of a given clock and on the defaultDS data set values of the given clock.

9.3.2.2 General BMC specifications

The BMC algorithm on an ordinary or boundary clock C_0 characterized by a defaultDS data set D_0 and other data sets, and with ‘N’ ports shall be as follows:

a) For each port ‘r’ of C_0, qualified Announce messages received from ports of other clocks connected to the communication path used by port ‘r’ shall be compared.

b) For each port ‘r’ of C_0, the best of these messages E_{rbest} shall be determined using the data set comparison algorithm.

c) For the set of N ports of clock C_0, the best of all messages, E_{best}, shall be determined from the N E_{rbest} messages using the data set comparison algorithm.

d) For each of the N ports of clock C_0, the messages E_{best} and E_{rbest} and the defaultDS data set D_0 shall be used with the state decision algorithm to determine the BMC event applicable to the state machine, see 9.2.5, of each port.

e) Except for the provisions of the master cluster option, see 17.3, a port shall disregard all Announce messages with alternateMasterFlag TRUE in the exercise of the best master clock algorithm under the terms subclause 9.3.2. These messages shall not be entered into the foreign master data set for consideration by the best master clock algorithm. The alternate master option, see 17.4, specifies other uses for Announce messages with this flag set to TRUE.

9.3.2.3 Computation of E_{rbest}

Each port may determine E_{rbest} independently of activity on other ports. The determination of E_{best}, the application of the state decision algorithm, and the instantiation of any state changes required by the application of the results of these determinations shall be atomic, see 3.1.2.

In computing E_{rbest} a port ‘r’ shall:

a) Consider only qualified Announce messages received on port ‘r,’ and only the most recently received one from each sending port.

b) Include at least two qualified Announce messages from a foreign master clock if such messages exist. In this case, ‘r’ shall delete from its implementation-specific foreignMasterDS data set, see 9.3.2.4, all such records that were considered but not selected as E_{rbest}.

c) If port ‘r’ is in the SLAVE state, include the results of the previous computation of E_{rbest} on port ‘r.’ However, if there is a more recent qualified Announce message received on port ‘r’ from the same sending port, the values from that message shall be considered instead. If an ANNOUNCE_RECEIPT_TIMEOUT_EXPIRES event occurs, see 9.2.6.11, for the clock selected during the previous computation of E_{rbest} on port ‘r,’ then the previous computation of E_{rbest} on port ‘r’ shall not be included.

NOTE—Announce messages should be included from as many foreign master clocks as port resources permit.

9.3.2.4 foreignMasterDS data set specifications

9.3.2.4.1 General

Each port shall maintain an implementation-specific foreignMasterDS data set for purposes of qualifying Announce messages. Each entry of the data set contains two members:

— foreignMasterDS.foreignMasterPortIdentity,

— foreignMasterDS.foreignMasterAnnounceMessages.

9.3.2.4.2 foreignMasterDS.foreignMasterPortIdentity

This member’s value shall be the sourcePortIdentity field value of an Announce message received from the foreign master.

9.3.2.4.3 foreignMasterDS.foreignMasterAnnounceMessages

This member’s value shall be the number of Announce messages from the foreign master indicated in the foreignMasterDS.foreignMasterPortIdentity member that have been received within a time window of duration FOREIGN_MASTER_TIME_WINDOW.

9.3.2.4.4 FOREIGN_MASTER_TIME_WINDOW and FOREIGN_MASTER_THRESHOLD

The values of these two attributes determine the criteria for accepting an Announce message from a previously silent master clock for consideration in the operation of the best master clock algorithm, see 9.3.2. The values of these attributes shall be:

- FOREIGN_MASTER_TIME_WINDOW: 4 announceIntervals.
- FOREIGN_MASTER_THRESHOLD: 2 Announce messages received within the FOREIGN_MASTER_TIME_WINDOW.

9.3.2.4.5 Size of foreignMasterDS data set

The implementation-specific foreignMasterDS data set shall have a minimum capacity of 5 foreign master records.

9.3.2.5 Qualification of Announce messages

An Announce message S received by a port ‘r’ shall be qualified for consideration by the BMC algorithm as follows:

a) If S was sent from port ‘r’ or from any other port on the clock containing port ‘r,’ see 9.5.2, S shall not be qualified.

b) If S is not the Announce message most recently received on port ‘r’ from a given clock, S shall not be qualified.

c) If the sender of S is a foreign master clock F, and fewer than FOREIGN_MASTER_THRESHOLD non-identical Announce messages from F have been received within the most recent FOREIGN_MASTER_TIME_WINDOW interval, S shall not be qualified.

NOTE—The purpose of this window and threshold is to qualify only stable potential masters and prevent spurious transitions of the best master.

d) If the stepsRemoved field of S is 255 or greater, S shall not be qualified.

NOTE—This provision ensures that rogue frames are extinguished. This is a mandatory back up to the use of the PATH_TRACE option for this purpose, see 16.2. This stepsRemoved-based mechanism may cause the failure of PTP if the size of the network is such that there are possible loops involving 255 boundary clocks. This is extremely unlikely in practical applications.

e) Otherwise, S shall be qualified.

9.3.3 State decision algorithm

The state decision algorithm used in the best master clock algorithm shall be as defined by Figure 26. After a decision is reached by use of this algorithm, the data sets of the local clock shall be updated as specified in 9.3.5.

D_0 represents the characteristics of clock C_0 as specified in the data sets of C_0.

Comparisons of D_0 to E_{best} or E_{rbest} shall be made using the data set comparison algorithm. For the decision block in Figure 26, "E_{best} better by topology than E_{rbest}," comparisons of E_{best} to E_{rbest} shall be made using the data set comparison algorithm.

The determination of whether D_0 is clockClass 1 through 127 shall be made based on the value of the clockClass member of D_0.

The values for recommended state used in the protocol engine state machines of Figure 23 and Figure 24 shall be the values for recommended state given in Figure 26.

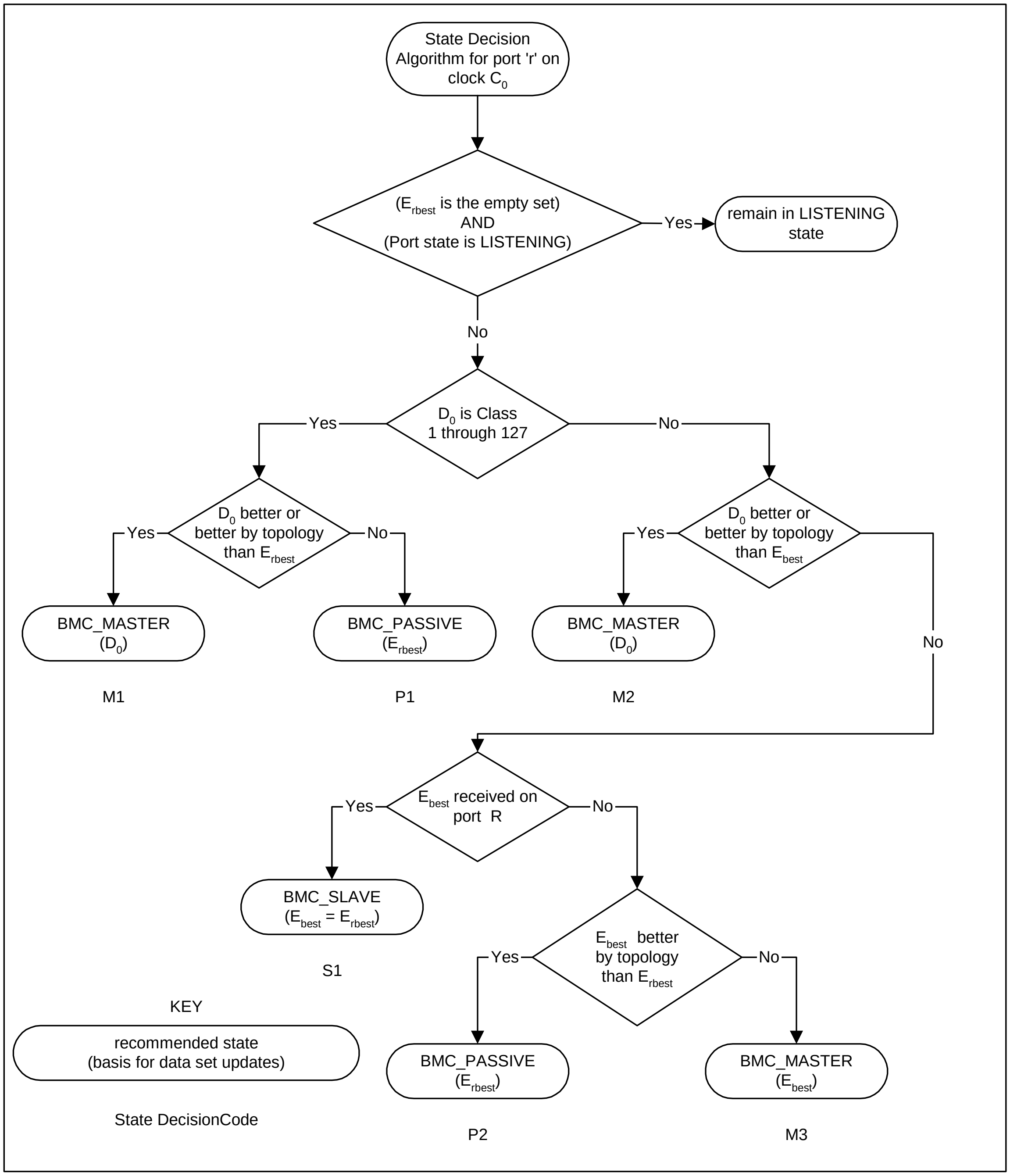

State Decision Algorithm for port 'r' on clock C_0
(E_{rbest} is the empty set) AND (Port state is LISTENING)
Yes
remain in LISTENING state
No
D_0 is Class 1 through 127
Yes
No
D_0 better or better by topology than E_{rbest}
Yes
No
D_0 better or better by topology than E_{best}
Yes
No
BMC_MASTER (D_0)
M1
BMC_PASSIVE (E_{rbest})
P1
BMC_MASTER (D_0)
M2
E_{best} received on port R
Yes
No
BMC_SLAVE (E_{best} = E_{rbest})
S1
E_{best} better by topology than E_{rbest}
Yes
No
BMC_PASSIVE (E_{rbest})
P2
BMC_MASTER (E_{best})
M3
KEY
recommended state (basis for data set updates)
State DecisionCode

Figure 26: State decision algorithm

9.3.4 Data set comparison algorithm

The best master clock algorithm compares one clock to another by comparing data sets that represent those clocks. The data set comparison algorithm shall be as defined by Figure 27, and Figure 28. The data sets are indicated in these figures as set A and set B. The sources for data set values are given in Table 12. If in comparing any two of the data sets D_0, E_{best}, or $E_{rbest,}$ one of the data sets is the empty set then the non-empty set is deemed the better set. D_0 is never the empty set; however E_{best} or E_{rbest} or both may be the empty set.

NOTE— The general process followed in this comparison is:

a) Find which clock derives its time from the better grandmaster. Choosing this, rather than which is the better clock, is essential for the stability of the algorithm.

b) If those properties are equivalent, use tie-breaking techniques.

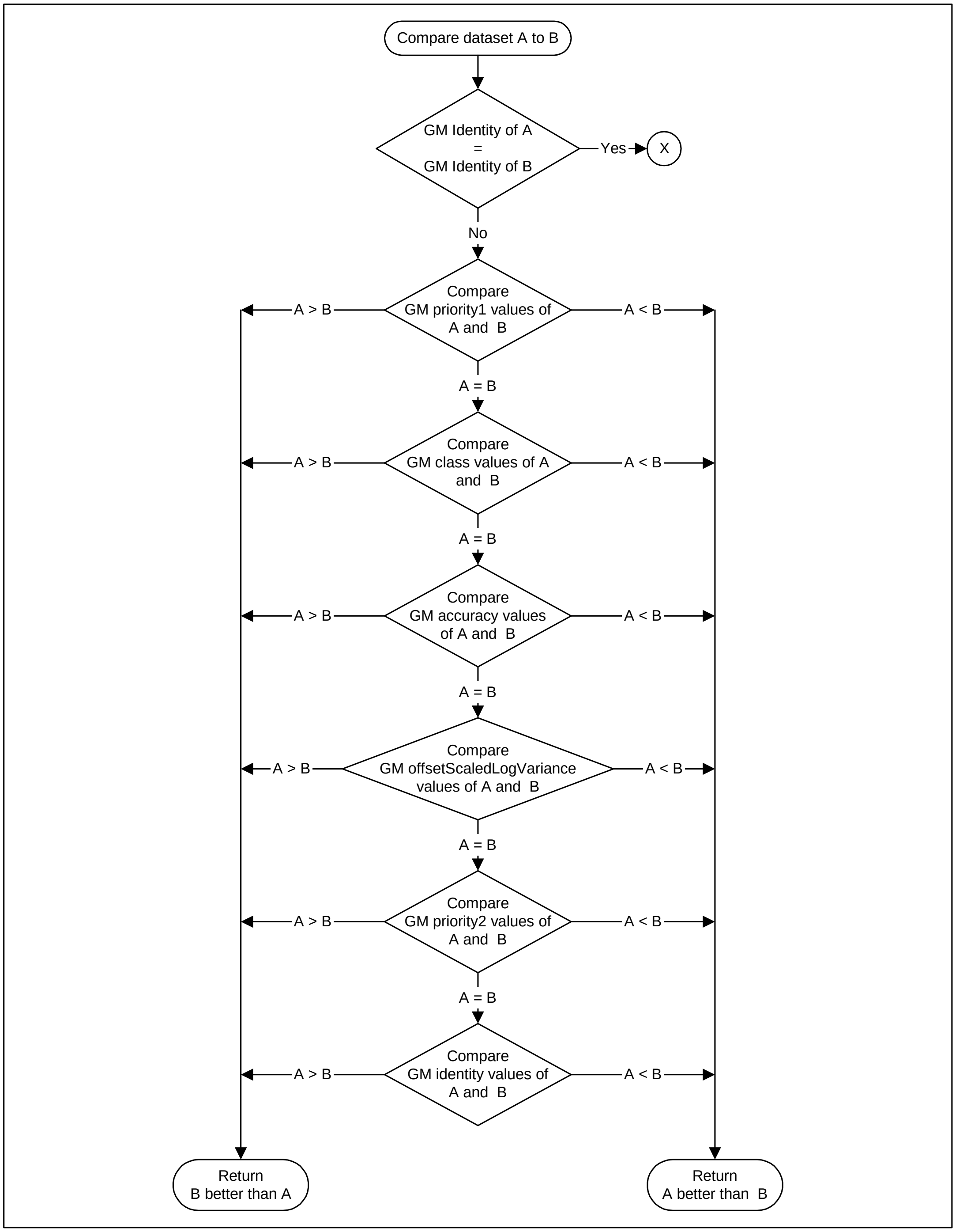

Compare dataset A to B
GM Identity of A = GM Identity of B
Yes
X
No
Compare GM priority1 values of A and B
A > B
A < B
A = B
Compare GM class values of A and B
A > B
A < B
A = B
Compare GM accuracy values of A and B
A > B
A < B
A = B
Compare GM offsetScaledLogVariance values of A and B
A > B
A < B
A = B
Compare GM priority2 values of A and B
A > B
A < B
A = B
Compare GM identity values of A and B
A > B
A < B
Return B better than A
Return A better than B

Figure 27: Data set comparison algorithm, part 1

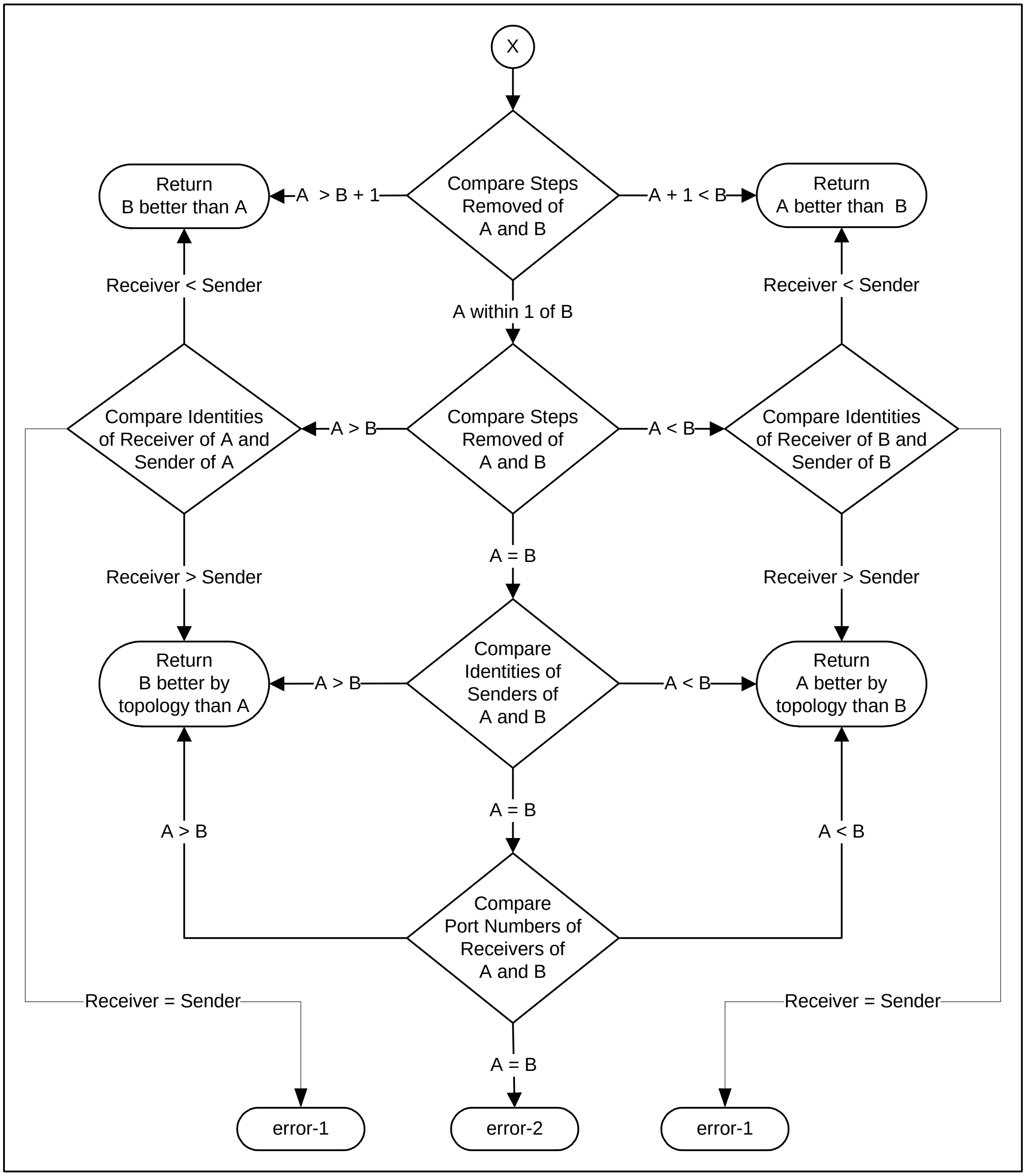

X
Compare Steps Removed of A and B
A > B + 1
Return B better than A
A + 1 < B
Return A better than B
A within 1 of B
Receiver < Sender
Receiver < Sender
Compare Identities of Receiver of A and Sender of A
A > B
Compare Steps Removed of A and B
A < B
Compare Identities of Receiver of B and Sender of B
Receiver > Sender
A = B
Receiver > Sender
Return B better by topology than A
A > B
Compare Identities of Senders of A and B
A < B
Return A better by topology than B
A = B
A > B
A < B
Compare Port Numbers of Receivers of A and B
Receiver = Sender
Receiver = Sender
A = B
error-1
error-2
error-1

Figure 28: Data set comparison algorithm, part 2

NOTE—The error returns are not used in the state decision algorithm of Figure 26. These returns are precluded by the terms of 9.3.2.5 but may be useful for diagnostic or error detection. The error conditions occur when terms of that Clause are violated. Error-1 indicates that one of the messages was transmitted and received on the same port. Error-2 indicates that the messages are duplicates or that they are an earlier and later message from the same grandmaster.

Table 12: Information sources for data set comparison algorithm

When considering this property as designated in Figure 27 and Figure 28	If the data set is E_{best} or E_{rbest}, use these fields of the associated Announce message	If the data set is D_0, use these fields of the local clock defaultDS data set
GM priority1	grandmasterpriority1	defaultDS.priority1
GM identity	grandmasterIdentity	defaultDS.clockIdentity
GM class	grandmasterClockQuality. clockClass	defaultDS.clockQuality. clockClass
GM accuracy	grandmasterClockQuality. clockAccuracy	defaultDS.clockQuality. clockAccuracy
GM offsetScaledLogVariance	grandmasterClockQuality. offsetScaledLogVariance	defaultDS.clockQuality. offsetScaledLogVariance
GM priority2	grandmasterPriority2	defaultDS.priority2
Steps Removed	stepsRemoved	The value 0
Identity of Senders	sourcePortIdentity	defaultDS.clockIdentity
Identity of Receiver	parentDS.portIdentity (of portDS data set of port receiving message)	defaultDS.clockIdentity
portNumber of Receivers	parentDS.portIdentity. portNumber (of portDS data set of port receiving message)	The value 0

9.3.5 Update of data sets

The update of the currentDS, parentDS, portDS, and timePropertiesDS data sets shall be as defined in Table 13, Table 14, Table 15, and Table 16 for the state decision codes, which are indicated by 'State Decision Code' in Figure 26 for the default best master clock algorithm and by 9.3.1 for a PTP profile defined alternate.

Member portDS.portState of the portDS data set shall be updated as changes are made by the protocol state machine of Figure 23 and Figure 24 associated with each port in accordance with 9.2.6.8.

Data set fields that are not included in Table 13, Table 14, Table 15, and Table 16 are not updated.

Table 13: Updates for state decision code M1 and M2

Update this field	**From the indicated field of the defaultDS data set of the clock unless otherwise stated**
currentDS data set	
currentDS.stepsRemoved	set to 0
currentDS.offsetFromMaster	set to 0
currentDS.meanPathDelay	set to 0
parentDS data set	
parentDS.parentPortIdentity	parentDS.clockIdentity set to the value of defaultDS.clockIdentity field. parentDS.parentPortIdentity.portNumber member is 0
parentDS.grandmasterIdentity	defaultDS.clockIdentity
parentDS.grandmaster_clockQuality	defaultDS.clockQuality
parentDS.grandmasterPriority1	defaultDS.priority1
parentDS.grandmasterPriority2	defaultDS.priority2
timePropertiesDS data set	
timePropertiesDS.currentUtcOffset	Follow rules in 9.4
timePropertiesDS.currentUtcOffsetValid	Follow rules in 9.4
timePropertiesDS.leap59	Follow rules in 9.4

timePropertiesDS.leap61	Follow rules in 9.4
timePropertiesDS.timeTraceable	Follow rules in 9.4
timePropertiesDS.frequencyTraceable	Follow rules in 9.4
timePropertiesDS.ptpTimescale	Follow rules in 9.4
timePropertiesDS.timeSource	Follow rules in 9.4
Port data set	
portDS.portState	State resulting from the application of the recommended state from Figure 26 to the state machine of Figure 23 or Figure 24 as appropriate.

Table 14: Updates for state decision code M3

Update this field	**From the indicated source**
Port data set	
portDS.portState	State resulting from the application of the recommended state from Figure 26 to the state machine of Figure 23 or Figure 24 as appropriate.

Table 15: Updates for state decision code P1, and P2

Update this field	**From the indicated source**
Port data set	
portDS.portState	State resulting from the application of the recommended state from Figure 26 to the state machine of Figure 23 or Figure 24 as appropriate.

Table 16: Updates for state decision code S1

Update this field	**From the indicated source**
currentDS data set	
currentDS.stepsRemoved	1 + value of stepsRemoved of E_{best} NOTE— E_{best} is an Announce message
parentDS data set	
parentDS.parentPortIdentity	sourcePortIdentity of E_{best}
parentDS.grandmasterIdentity	grandmasterIdentity of E_{best}
parentDS.grandmaster_clockQuality	grandmasterClockQuality of E_{best}
parentDS.grandmasterPriority1	grandmasterPriority1 of E_{best}
parentDS.grandmasterPriority2	grandmasterPriority2 of E_{best}
timePropertiesDS data set	
timePropertiesDS.currentUtcOffset	currentUTCOffset field of E_{best}
timePropertiesDS currentUtcOffsetValid	The logical value of the currentUtcOffsetValid bit of the flags field of E_{best}
timePropertiesDS.leap59	The logical value of the LI_59 bit of the flags field of E_{best}
timePropertiesDS.leap61	The logical value of the LI_61 bit of the flags field of E_{best}
timePropertiesDS.timeTraceable	The logical value of the timeTraceable bit of the flags field of E_{best}
timePropertiesDS.frequencyTraceable	The logical value of the frequencyTraceable bit of the flags field of E_{best}

Update this field	From the indicated source
timePropertiesDS.ptpTimescale	The logical value of the ptpTimescale bit of the flags field of E_{best}
timePropertiesDS.timeSource	The value of the timeSource field of E_{best}
Port data set	
portDS.portState	State resulting from the application of the recommended state from Figure 26 to the state machine of Figure 23 or Figure 24 as appropriate.

9.4 Grandmaster clocks

The grandmaster clock determines the timescale and related timescale properties of the domain.

NOTE— A clock in the MASTER state as a result of M1 or M2 of the state decision algorithm of Figure 26 is the grandmaster clock.

If the timescale is PTP, and the clock is class 6, 7, or 52, the timePropertiesDS data set members shall be set as follows:

— LI-59 and LI-61: if known, to the value obtained from a primary reference, else to FALSE

— timePropertiesDS.currentUtcOffset: if known, to the value obtained from a primary reference, else to the current number of leap seconds, 7.2.3, when the node is designed

— timePropertiesDS.currentUtcOffsetValid: to TRUE if the value of currentUtcOffset is known to be correct, else to FALSE

— timePropertiesDS.ptpTimescale: to TRUE

— timePropertiesDS.timeTraceable: to TRUE if the time traceable to a primary reference, else to FALSE

— timePropertiesDS.frequencyTraceable: to TRUE if the frequency is traceable to a primary reference, else to FALSE

— timePropertiesDS.timeSource: if known to the appropriate value from Table 7, else to INTERNAL_OSCILLATOR.

If the timescale is ARB then, for all clockClasses except 6, 7, or 52, unless otherwise specified, the timePropertiesDS data set members shall be set as follows:

— timePropertiesDS.LI-59 and timePropertiesDS.LI-61: to FALSE

— timePropertiesDS.currentUtcOffset: to the current number of leap seconds, 7.2.3, when the node is designed

— timePropertiesDS.currentUtcOffsetValid: to TRUE if the value of currentUtcOffset is known to be correct, else to FALSE

— timePropertiesDS.ptpTimescale: to FALSE

— timePropertiesDS.timeTraceable: to TRUE if the time traceable to a primary reference, else to FALSE

— timePropertiesDS.frequencyTraceable: to TRUE if the frequency is traceable to a primary reference, else to FALSE

— timePropertiesDS.timeSource: if known to the appropriate value from Table 7, else to INTERNAL_OSCILLATOR.

If either LI-59 or LI-61 is TRUE, it shall be set to TRUE during the interval prior to midnight (UTC) of the end of the current day and the later of the times:

— when the grandmaster determined that the value is TRUE, and

— 12 hours prior to midnight (UTC) of the current day.

The values of these variables shall be set to FALSE starting with the first Announce message in which the updated value of timePropertiesDS.currentUtcOffset appears.

The update of timePropertiesDS.currentUtcOffset, timePropertiesDS.LI-59 and timePropertiesDS.LI-61 shall occur within ±2 announceIntervals of midnight (UTC).

NOTE 1—The number of leap seconds, and hence currentUtcOffset, changes at midnight (UTC) by international standards. In practical implementations the actual update of currentUtcOffset and the setting of LI-59 or LI-61 to FALSE may occur slightly before or after midnight (UTC) due to variations in code execution time. The requirement of the ± 2 announceInterval update is easily implemented in masters and ensures, even in very large topologies, that the information is propagated to all slaves prior to noon of the following UTC day. This avoids confusion as to which midnight the indicated jump in UTC is to occur.

NOTE 2—Grandmaster clocks that use a recognized standard time source should distribute the PTP timescale.

9.5 Message processing semantics

9.5.1 Messaging behavior of ordinary and boundary clocks

Subclause 9.5 specifies the behavior for the following events that may occur in an ordinary clock, or for each PTP port of a boundary clock:

— Receipt of a message

— Transmission of a message.

The behavior resulting from these events may depend on the current state of the PTP port of an ordinary clock or boundary clock as specified in 9.2.5.

Except for management messages, all multicast PTP messages shall terminate at boundary and ordinary clocks.

A boundary clock that is also a bridge or router forwards all unicast PTP messages according to the forwarding rules of the network. Any other behavior is out of scope of this standard.

Execution of the protocol shall not result in any communication on the communication paths except as specified in the following clauses.

Any state or data changes or data set updates of the local clock resulting from the occurrence of any event or qualifying sequence of events defined in 9.5 shall be atomic, see 3.1.2. A qualifying sequence of events is defined to be the receipt of multiple announce messages within the same announce message transmission interval.

Requests to issue a PTP message resulting from state or data change or the occurrence of an event, shall be processed in first-in-first-out (FIFO) order by message class. A logically separate FIFO ordering shall be maintained for event and general messages. Once a message request is queued, the message shall be issued irrespective of any subsequent event occurrences.

Only PTP messages where the domainNumber field of the PTP message header, see 13.3.2.5, is identical to the defaultDS data set member domainNumber shall be accepted for processing by the protocol.

9.5.2 Receipt by a clock of any message from itself

9.5.2.1 General

The Precision Time Protocol makes no use of messages received by the same clock that sent them, and implementations should prevent such messages from being considered by the protocol. Subclauses 9.5.2.2 and 9.5.2.3specifies what to do when such messages are received.

9.5.2.2 Ordinary clocks and any single port of a boundary clock

A message received at the same port that issued the message shall be ignored. An exception may be made for implementation-specific diagnostic purposes beyond the scope of this standard.

This situation may be identified by comparing the sourcePortIdentity field members of the received message and the corresponding members of the portIdentity field of the portDS data set of the ingress port. The possibilities and interpretations are shown in Table 17.

NOTE— This condition may occur due to normal or abnormal properties of communication paths.

9.5.2.3 Additional constraints for boundary clocks

A port ‘N’ on a boundary clock might directly receive Announce messages issued by a different port ‘M’ of the same boundary clock. This occurs if both ports N and M communicate to the same communication path. This is an abnormal situation not detected by the best master clock algorithm. When this condition is detected in a set of the ports, the boundary clock shall place all of the involved ports except the port with the lowest portNumber, say N, in the PASSIVE state until such time as the normal operation of the protocol no longer determines that port N is to be in the MASTER state.

This situation may be identified by comparing the sourcePortIdentity field members of the received message and the corresponding members of the portIdentity field of the portDS data set of the ingress port. The possibilities and interpretations are shown in Table 17.

Table 17: Source identity comparisons

Given a message *m* arriving at port *n* of clock *c*, where the portDS.portIdentity field of *n* has members clockIdentity *a* and portNumber *n*:	
If sourcePortIdentity of *m* contains:	**The interpretation is:**
clockIdentity *a,* portNumber *n*	*m* was sent from port *n* on clock *c.*
clockIdentity *f ≠ a,* portNumber *n*	*m* was sent from a port on a clock other than *c.*
clockIdentity *a,* portNumber *q ≠ n*	*m* was sent from clock *c,* but from a port other than *n.*

9.5.3 Receipt of an Announce message from another clock

The logic for processing an Announce message shall be as defined in Figure 29. The states indicated in this figure refer to the current state of the port receiving the Announce message.

If the port receiving an Announce message is in the INITIALIZING or DISABLED states, the message shall be disregarded. If the port receiving an Announce message is in the FAULTY state, the message shall be disregarded except for implementation-specific purposes that otherwise meet the requirements of 9.2.

If the members of the sourcePortIdentity field of the received Announce message are identical with the corresponding members of the parentPortIdentity field of the parentDS data set of the receiving clock, then the message is from the current parent clock, i.e. the master clock to which the clock is synchronized.

When an Announce message from port F is received by port N from the communication path associated with N, port F is designated as a foreign master clock if any of the following is true:

- The port N is not in the SLAVE state, or
- The port N is in the SLAVE state and the members of the sourcePortIdentity field of the received message are not all identical with the corresponding members of the parentPortIdentity field of the parentDS data set of the receiving clock.

If an announce message is received from a foreign master clock, the implementation-specific foreignMasterDS data set, see 9.3.2.4, of the ingress port shall be updated as follows:

- If the members of the sourcePortIdentity field of the received Announce message are identical with the corresponding members of a foreignMasterPortIdentity field of the foreignMasterDS data set of the ingress port, then the foreignMasterAnnounceMessages field of that record shall be incremented.
- If the members of the sourcePortIdentity field of the received Announce message are not identical to the corresponding members of a foreignMasterPortIdentity field of the foreignMasterDS data set of the ingress port, then a new record shall be created in the foreignMasterDS data set with the foreignMasterPortIdentity field set to the value of the sourcePortIdentity field of the received Announce message and with the foreignMasterAnnounceMessages field value set to 0. Implementation-specific limitations on the capacity of the foreignMasterDS data set limits the number of such records. If the foreignmaster data set is full, then the Announce message shall be disregarded.

If an Announce message is received from the current parent clock, the data sets for the ingress port shall be updated according to Table 16, except that the source of each field shall be the received Announce message rather than $E_{best.}$

NOTE—The maximum number of records that can be stored in the foreignMasterDS data set, see 9.3.2.4.5, does not affect the correctness of protocol operation but may affect the speed of convergence in selecting the masters and slaves. Higher capacities permit a node to process and eliminate from consideration more foreign masters in each announceInterval rather than wait until space is available in the data set for succeeding messages from any remaining foreign masters.

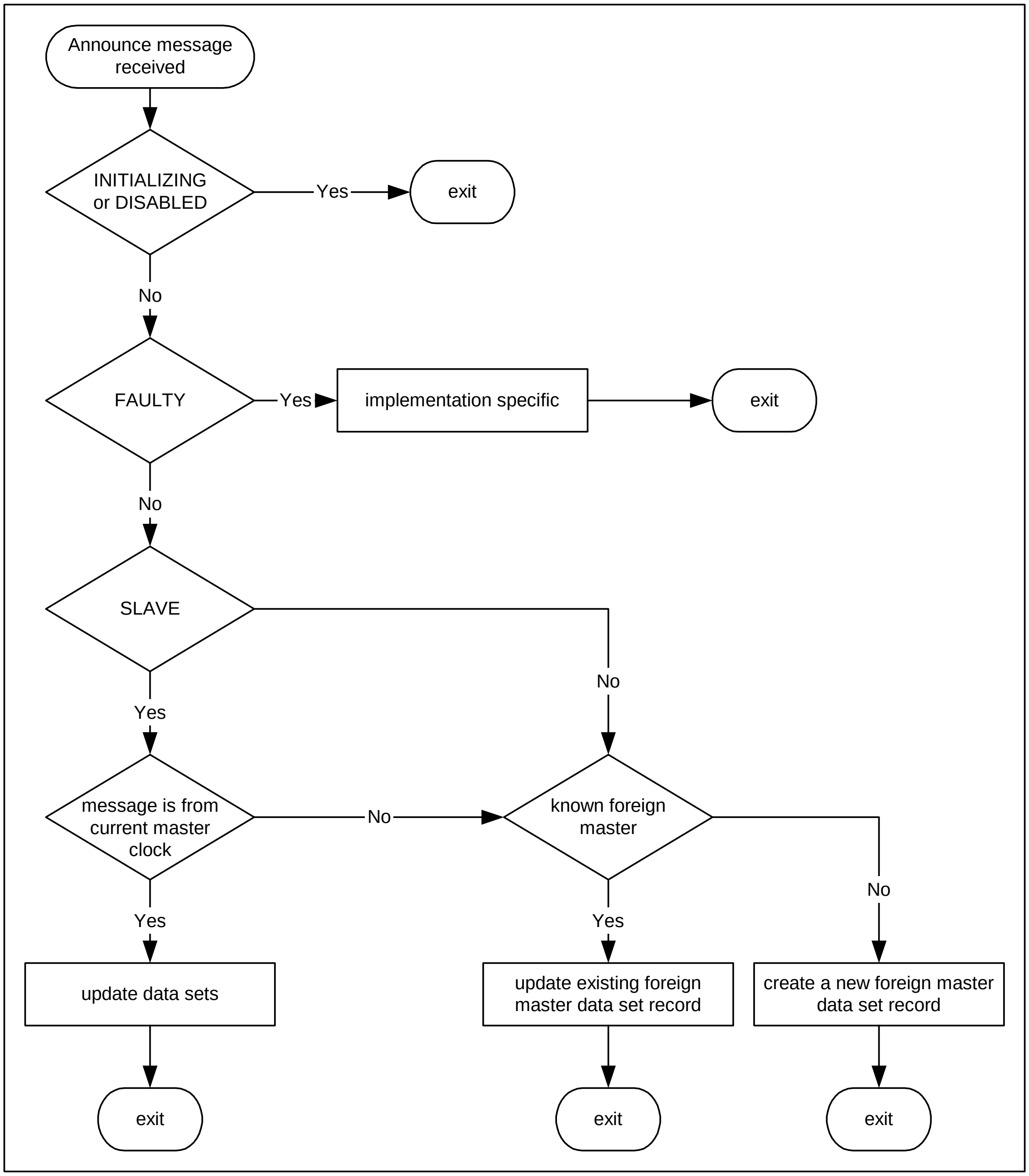

Announce message received
INITIALIZING or DISABLED
Yes
exit
No
FAULTY
Yes
implementation specific
exit
No
SLAVE
No
Yes
message is from current master clock
No
known foreign master
No
Yes
Yes
update data sets
update existing foreign master data set record
create a new foreign master data set record
exit
exit
exit

Figure 29: Receipt of Announce message logic

9.5.4 Receipt of a Sync message from another clock

The logic for processing a Sync message shall be as defined in Figure 30. The states indicated in this figure refer to the current state of the port receiving the Sync message.

If the port receiving a Sync message is in the INITIALIZING or DISABLED states, the message shall be disregarded. If the port receiving a Sync message is in the FAULTY state, the message shall be disregarded except for implementation-specific purposes that otherwise meet the requirements of 9.2.

The <syncEventIngressTimestamp> meeting the requirements of 7.3.4 shall be generated for the reception of the Sync message.

Received Sync messages should be processed as soon as possible after receipt.

If the members of the sourcePortIdentity field of the received Sync message are identical with the corresponding members of the parentPortIdentity field of the parentDS data set of the receiving clock, then the message is from the current master clock.

When the Sync message is received and all of the following conditions are met:

- The port receiving the Sync message is in the SLAVE or UNCALIBRATED state, and
- The twoStepFlag bit of the flags field of the received Sync message is FALSE (indicating that a Follow_Up message will not be received), and
- The Sync message was received from the current master clock

then:

- The Sync message correctionField shall be adjusted for asymmetry per 11.6.2.
- The local clock should be synchronized, per 12.2, based on the contents of the received Sync message and the <syncEventIngressTimestamp>.
- If required by subclause 9.5.11.2, the delay request-response mechanism specified in subclause 11.3 shall be initiated.

When the Sync message is received and all of the following conditions are met:

- The port receiving the Sync message is in the SLAVE or UNCALIBRATED state, and
- The twoStepFlag bit of the flags field of the received Sync message is TRUE (indicating that a Follow_Up message will be received), and
- The Sync message was received from the current master clock

then:

- The Sync message correctionField shall be adjusted for asymmetry per 11.6.2.
- If required by subclause 9.5.11.2, the delay request-response mechanism specified in subclause 11.3 shall be initiated.

NOTE—The specification in 11.3 results in a computation of the mean_path_delay based on the four timestamps associated with the Sync message and the Delay_Req message initiated by the receipt of the Sync message as specified in this subclause. In the case of the master being a two-step clock this will require the slave to wait for the Follow_Up and Delay_Resp messages from the master. As noted in the note in 11.3.2 implementations may choose to use previous values of Sync timestamps albeit with diminished accuracy in the result.

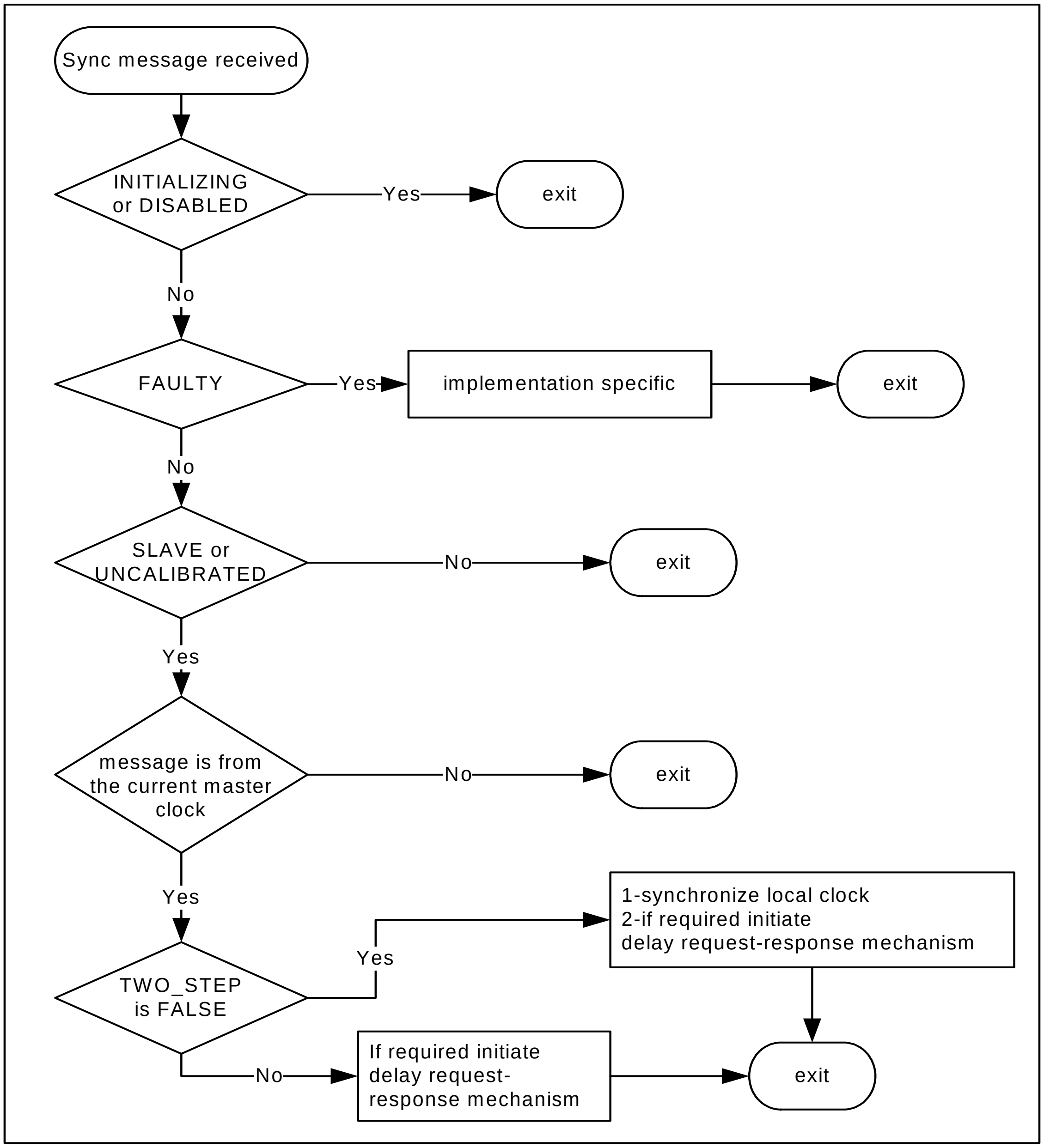


Figure 30 Receipt of Sync message logic

9.5.5 Receipt of a Follow_Up message from another clock

The logic for processing a Follow_Up message shall be as defined in Figure 31. The states indicated in this figure refer to the current state of the port receiving the Follow_Up message.

If the port receiving a Follow_Up message is in the INITIALIZING or DISABLED states, the message shall be disregarded. If the port receiving a Follow_Up message is in the FAULTY state, the message shall be disregarded except for implementation-specific purposes that otherwise meet the requirements of 9.2.

If the members of the sourcePortIdentity field of the received Follow_Up message are identical with the corresponding members of the sourcePortIdentity field of a prior Sync message and the sequenceId field of the received Follow_Up message matches the sequenceId field of the same prior Sync message, then the Follow_Up message and the Sync message are associated.

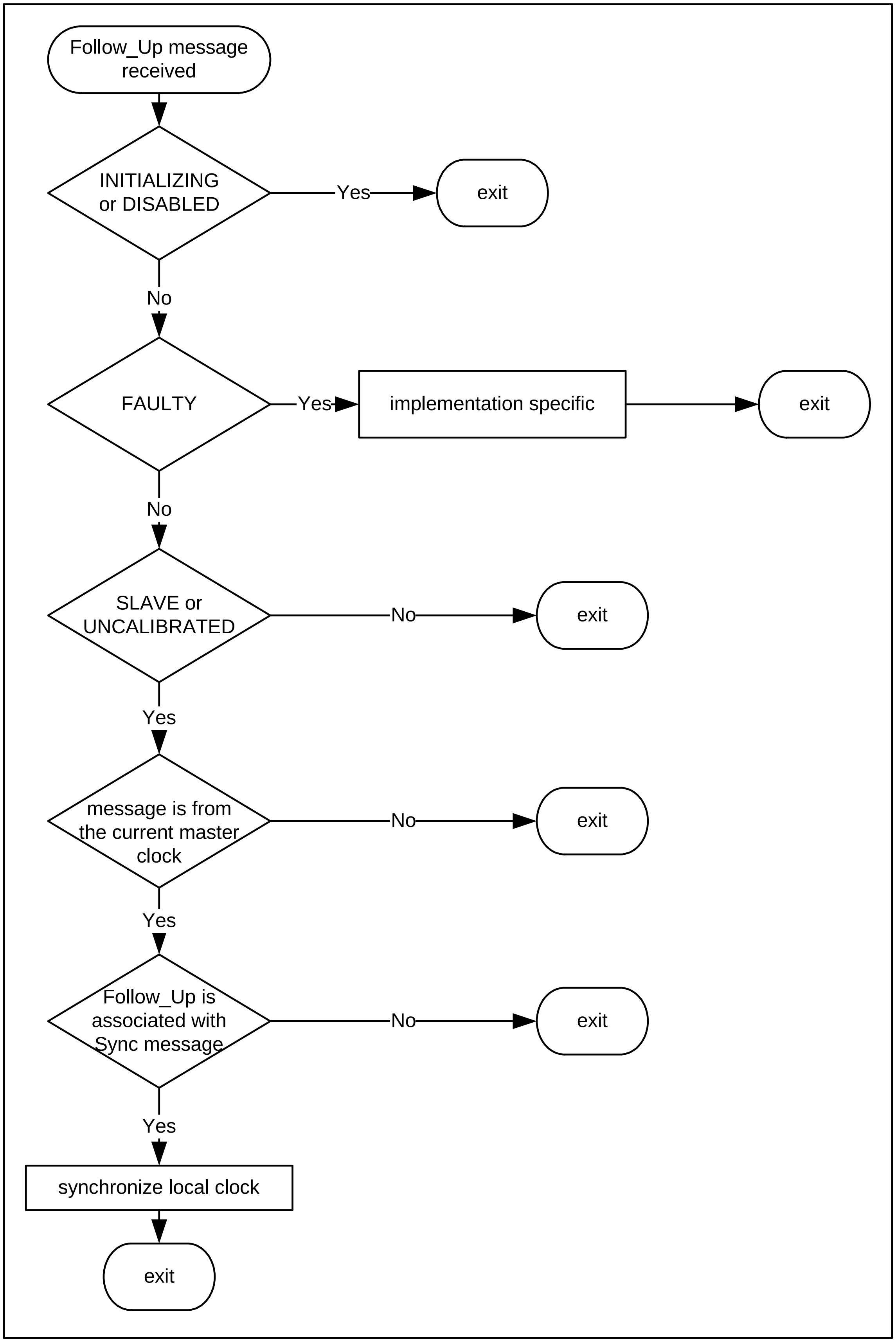
If the members of the sourcePortIdentity field of the associated Sync message are identical with the corresponding members of the parentPortIdentity field of the parentDS data set of the receiving clock, then the message is from the current master clock.

Follow_Up messages should be processed as soon as possible after receipt.

When the associated Sync message is received and all of the following conditions are met:

- The port receiving the Follow_Up message is in the SLAVE or UNCALIBRATED state, and
- The Sync message was received from the current master clock and
- The Follow_Up message is associated with this Sync message,

then the local clock should be synchronized, per 12.2, based on the contents of the received Follow_Up message and associated Sync message and the <syncEventIngressTimestamp> of the associated Sync message.


Follow_Up message received
INITIALIZING or DISABLED
Yes
exit
No
FAULTY
Yes
implementation specific
exit
No
SLAVE or UNCALIBRATED
No
exit
Yes
message is from the current master clock
No
exit
Yes
Follow_Up is associated with Sync message
No
exit
Yes
synchronize local clock
exit

1

Figure 31: Receipt of Follow_Up message logic

9.5.6 Receipt of a Delay_Req message from another clock

The logic for processing a Delay_Req message shall be as defined in Figure 32. The states indicated in this figure refer to the current state of the port receiving the Delay_Req message.

If the port receiving the Delay_Req message is in the INITIALIZING or DISABLED states, the message shall be disregarded. If the port receiving a Delay_Req message is in the FAULTY state, the message shall be disregarded except for implementation-specific purposes that otherwise meet the requirements of 9.2.

Unless otherwise specified in this standard, if the port receiving the Delay_Req message is not in the MASTER state, the message shall be disregarded.

The <delayReqEventIngressTimestamp> meeting the requirements of 7.3.4 shall be generated upon receipt of the Delay_Req message.

If the port receiving the Delay_Req message is in the MASTER state, the port shall transmit a Delay_Resp message subject to the conditions of 11.3 and 9.5.12.

Delay_Req messages should be processed as soon as possible after receipt.

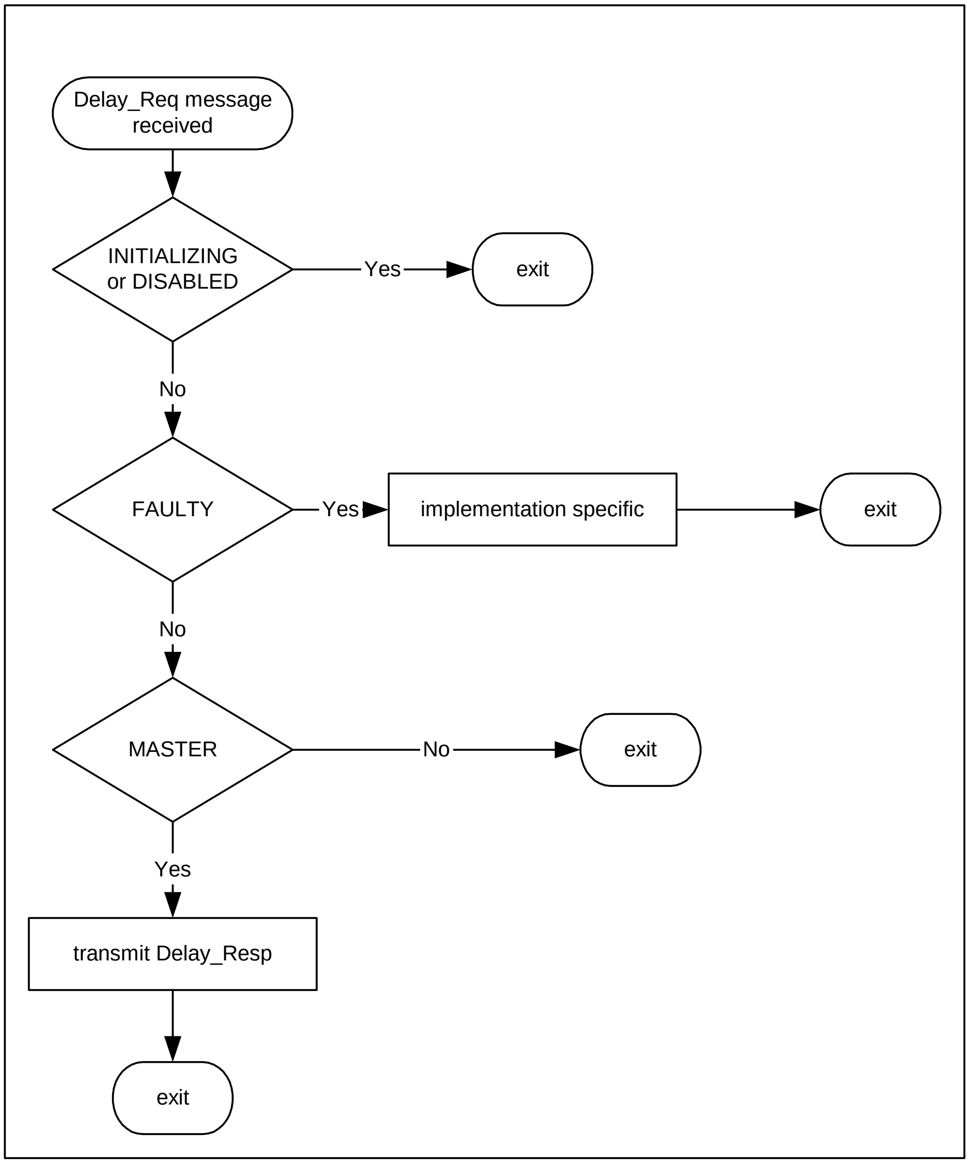


Figure 32: Receipt of Delay_Req message logic

9.5.7 Receipt of a Delay_Resp message from another clock

The logic for processing a Delay_Resp message shall be as defined in Figure 33. The states indicated in this figure refer to the current state of the port receiving the Delay_Resp message.

If the port receiving a Delay_Resp message is in the INITIALIZING or DISABLED states, the message shall be disregarded. If the port receiving a Delay_Resp message is in the FAULTY state, the message shall be disregarded except for implementation-specific purposes that otherwise meet the requirements of 9.2.

If the members of the requestingSourcePortIdentity field of the received Delay_Resp message are identical with the corresponding members of the sourcePortIdentity field of a prior Delay_Req message issued by the receiving clock, and the requestingSequenceId field of the received Delay_Resp message matches the sequenceId field of the same prior Delay_Req message, then the Delay_Resp message and the Delay_Req message are associated.

If the members of the sourcePortIdentity field of the received Delay_Resp message are identical with the corresponding members of the parentPortIdentity field of the parentDS data set of the receiving clock, then the Delay_Resp message is from the current master clock.

Delay_Resp messages should be processed as soon as possible after receipt.

When the port receives a Delay_Resp message subsequent to transmitting the associated Delay_Req message, and all of the following conditions are met:

- The port receiving the Delay_Resp message is in the SLAVE or UNCALIBRATED state, and
- The Delay_Resp message was received from the current master clock, and
- The Delay_Resp message is associated with the transmitted Delay_Req message,

then the receiving clock should:

a) Execute the delay request-response mechanism actions required by 11.3, based on the contents of the received Delay_Resp message and associated Delay_Req message.

b) Update the logMinDelayReqInterval member of the portDS data set to the value of the logMessageInterval member of the Delay_Resp message.

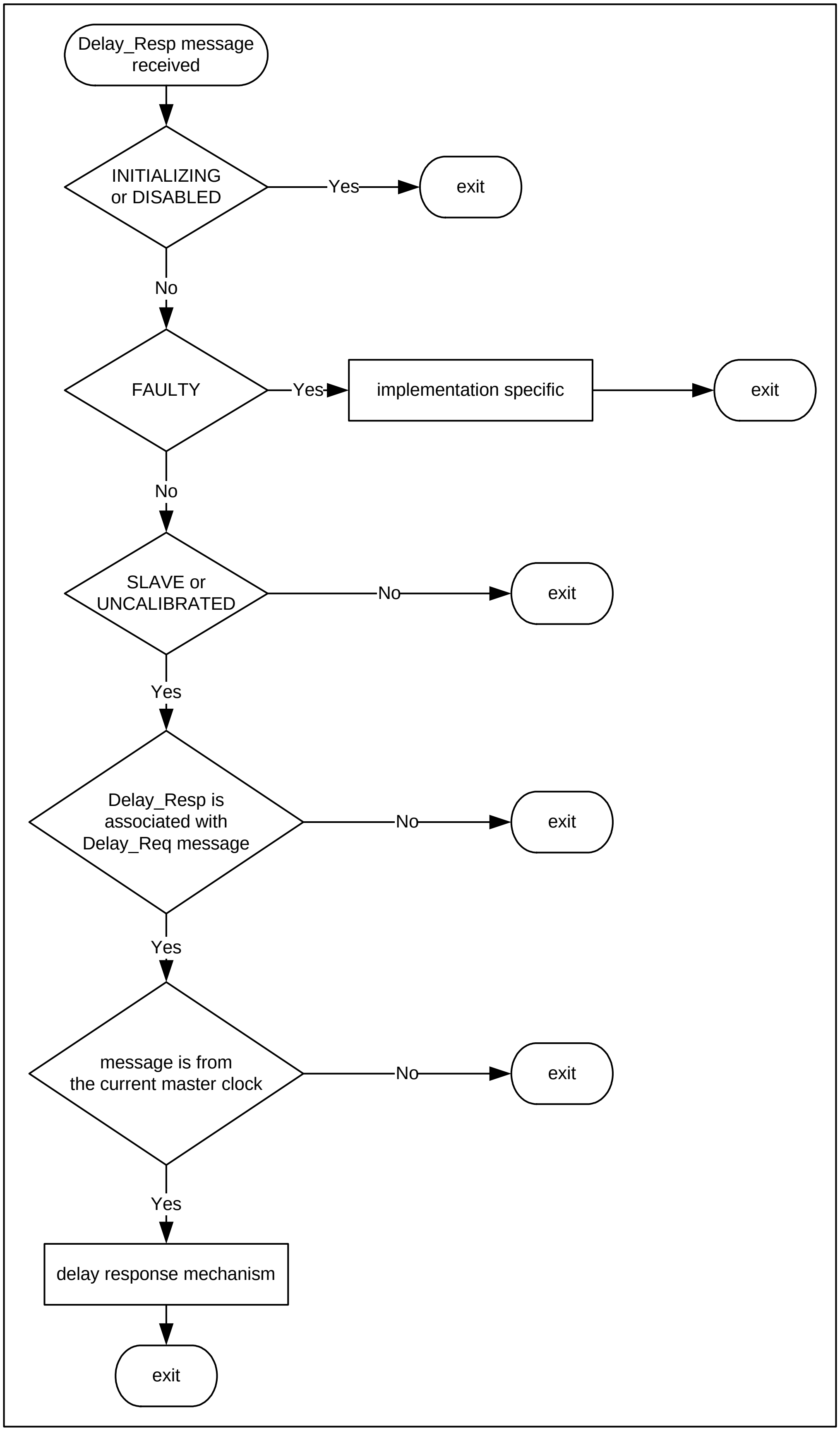

Delay_Resp message received
INITIALIZING or DISABLED
Yes
exit
No
FAULTY
Yes
implementation specific
exit
No
SLAVE or UNCALIBRATED
No
exit
Yes
Delay_Resp is associated with Delay_Req message
No
exit
Yes
message is from the current master clock
No
exit
Yes
delay response mechanism
exit

Figure 33: Receipt of Delay_Resp message logic

9.5.8 Transmission of an Announce message

Unless otherwise stated in this standard, a port shall not transmit an Announce message except as required by 9.5.8.

A port in the MASTER state shall periodically transmit an Announce message.

Such Announce messages shall be transmitted as a multicast, see 7.3.1, such that the logarithm to the base 2 of the mean value of the interval in seconds between message transmissions is the value of the logAnnounceInterval of the portDS data set of the transmitting clock. A node shall, with 90% confidence, issue messages with intervals within +/- 30% of the stated value of this attribute.

An optional unicast transmission at negotiated transmission intervals may be used for Announce messages subject to the terms of Clause 16.

NOTE—The capacity of the node to maintain this data for each unicast address may limit the number of unicast contracts negotiated under Clause 16. In some environments multicast transmissions may not be feasible. A unicast transmission is permitted provided the operation of the protocol is preserved, see 7.3.1.

9.5.9 Transmission of a Sync message

9.5.9.1 General specification

Unless otherwise stated in this standard, a port shall not transmit a Sync message except as required by 9.5.9.

9.5.9.2 General requirements

A port in the MASTER state shall periodically transmit a Sync message. Such Sync messages shall be transmitted as a multicast, see 7.3.1, such that the logarithm to the base 2 of the mean value of the interval in seconds between message transmissions is the value of the logSyncInterval of the portDS data set of the transmitting clock. A node shall, with 90% confidence, issue messages with intervals within +/- 30% of the stated value of this attribute.

An optional unicast transmission at negotiated transmission intervals may be used for Sync messages subject to the terms of Clause 16.

NOTE—The capacity of the node to maintain this data for each unicast address may limit the number of unicast contracts negotiated under Clause 16. In some environments multicast transmissions may not be feasible. A unicast transmission is permitted provided the operation of the protocol is preserved, see 7.3.1.

The fields of the transmitted Sync message shall conform to the requirements of Clause 13.

The <syncEventEgressTimestamp> meeting the requirements of 7.3.4 shall be generated upon transmission of the Sync message.

9.5.9.3 one-step clocks

The originTimestamp field of the Sync message shall be an estimate no worse than ± 1 second of the <syncEventEgressTimestamp> excluding any fractional nanoseconds.

The originTimestamp field of the Sync message should be the value of the <syncEventEgressTimestamp> excluding any fractional nanoseconds.

The sum of the Sync message correctionField and originTimestamp field shall be the value of the <syncEventEgressTimestamp> including any fractional nanoseconds.

9.5.9.4 two-step clocks

The originTimestamp field of the Sync message shall be 0 or an estimate no worse than ± 1 second of the <syncEventEgressTimestamp>.

The correction field of the sync message shall be set to 0.

A two-step clock shall transmit both a Sync and a Follow_Up message.

The port shall capture the sequenceId value of the Sync message as an input to the sequenceId field of the Follow_Up message. The mechanism for obtaining the value for the preciseOriginTimestamp and correctionField fields of the associated Follow_Up message shall be started.

9.5.10 Transmission of a Follow_Up message

Unless otherwise stated in this standard, a port shall issue a Follow_Up message only if required by 9.5.9.4. The Sync message whose transmission requires the transmission of a Follow_Up message is the associated Sync message.

The Follow_Up message should be transmitted as soon as possible after the transmission of the associated Sync message and shall be transmitted prior to the transmission of a subsequent Sync message to the same destination address.

The sequenceId field of the Follow_Up message shall be the value of the sequenceId field of the associated Sync message.

The preciseOriginTimestamp field of the Follow_Up message shall be an estimate no worse than ± 1 second of the <syncEventEgressTimestamp> of the associated Sync message.

The preciseOriginTimestamp field of the Follow_Up message should be the value of the <syncEventEgressTimestamp> of the associated Sync message excluding any fractional nanoseconds.

The sum of the correction fields in the Follow_Up and associated Sync messages added to the preciseOriginTimestamp field of the Follow_Up message shall be the precise value of the <syncEventEgressTimestamp> including any fractional nanoseconds.

If the Follow_Up message is associated with an optional unicast Sync message per 9.5.9.2 then the Follow_Up message shall also be transmitted as a unicast message to the same unicast address as the associated Sync message. These unicast Follow_Up messages shall meet all other requirements of 9.5.10.

9.5.11 Transmission of a Delay_Req message

9.5.11.1 General requirements

A clock shall issue a Delay_Req message on a port only if all of the following conditions are met:

- The port is in the SLAVE or UNCALIBRATED state, and
- The device is configured to execute the delay request-response mechanism, see 8.2.5.4.4, and

— The device is permitted to do so according to the timing requirements of 9.5.11.2.

Delay_Req messages shall be transmitted as multicast except if:

— The optional unicast provisions of Clause 16 are used

— Specified otherwise by a profile.

The fields of this Delay_Req message shall conform to the requirements of 11.3.

NOTE 1— If the clock uses a PTP profile that specifies syntonize only then the clock is not required to send Delay_Req messages. In this case maintaining delay request state information is not required.

NOTE 2— For clocks that use the peer delay mechanism, see 9.5.13.

9.5.11.2 Timing requirements

The transmission of Delay_Req messages from a port shall be limited as follows:

— The initial Delay_Req message may be transmitted when required.

— Subsequent Delay_Req messages shall be transmitted such that the logarithm to the base 2 of the mean value of the interval in seconds between message transmissions is not less than the value of the logMinDelayReqInterval of the portDS data set.

— The transmission intervals shall be taken from a uniform random distribution with a minimum width of $2^{logMinDelayReqInterval}$ seconds. The mean should be computed over no more than 30 sequential transmissions of Delay_Req messages.

NOTE— The logMinDelayReqInterval value is the logMessageInterval field of the last Delay_Resp message received in response to a Delay_Req message issued by the port.

9.5.12 Transmission of a Delay_Resp message

Unless otherwise stated in this standard, a port shall issue a Delay_Resp message when it is associated with the receipt of a Delay_Req message as required by 9.5.6 and meets the other requirements of 9.5.11.

The Delay_Req message whose transmission may require the transmission of a Delay_Resp message is the associated Delay_Req message.

A clock shall issue a Delay_Resp message on a port when all of the following conditions are met:

— The port is in the MASTER state or is required as the result of an optional unicast contract, Clause 16, and

— The device is configured to execute the delay request-response mechanism, see 8.2.5.4.4

— The transmission is the result of the receipt of the associated Delay_Req message, see 9.5.6.

Delay_Resp messages shall be transmitted as multicast if the associated delay request was sent as multicast.

Delay_Resp messages shall be transmitted as unicast if the associated delay request was sent as unicast.

The fields of this Delay_Resp message shall conform to the requirements of 11.3.

The receiveTimestamp field of the Delay_Resp shall be the <delayReqEventIngressTimestamp> of the associated Delay_Req message.

Prior to transmitting the Delay_Resp message the port shall in order:

a) Compute the updated value for the logMinDelayReqInterval field, see 7.7.2.4.

b) Use this computed value to update logMinDelayReqInterval member of the portDS data set, see 8.2.5.3.2.

c) Ensure the message field values are the current values as specified in 11.3.2.

d) Insert into logMessageInterval, the value as specified in Table 24.

The Delay_Resp message should be transmitted as soon as possible after the receipt of the associated Delay_Req message.

9.5.13 Transmission of a Pdelay_Req message

9.5.13.1 General requirements

A clock shall issue a Pdelay_Req message on a port only if all of the following conditions are met:

— The device is configured to execute the peer delay mechanism, see 8.2.5.4.4 and

— The device is permitted by the timing requirements of 9.5.13.2.

The fields of this Pdelay_Req message shall conform to the requirements of 11.4.3.

NOTE— If the clock uses a PTP profile that specifies syntonize only then the clock is not required to invoke the peer delay or delay request mechanism. In this case maintaining peer delay mechanism state information is not required.

9.5.13.2 Timing requirements

The transmission of a Pdelay_Req messages from a requesting port shall be limited as follows:

— The initial Pdelay_Req message may be transmitted when required.

— Subsequent Pdelay_Req messages shall be transmitted such that the logarithm to the base 2 of the mean value of the interval in seconds between message transmissions is no smaller than the value of the logMinPdelayReqInterval member of the portDS data set.

9.5.14 Transmission of Pdelay_Resp message

Pdelay_Resp messages should be transmitted as soon as possible after the receipt of the associated Pdelay_Req message.

9.5.15 Transmission of Pdelay_Resp_Follow_Up message

Pdelay_Resp_Follow_Up messages should be transmitted as soon as possible after the transmission of the associated Pdelay_Resp message.

9.6 Changes in the local clock

Changes due to 1) internal properties of the local clock, for example, a disruption of a local oscillator, or 2) interaction outside of PTP, for example, changes in information received from a GPS receiver to which the local clock is directly synchronized, may result in updates to the data sets.

All resulting changes to data sets shall be treated atomically with respect to any other activity accessing the data sets, including the activity specified in 9.2.6.8.

10. PTP for transparent clocks

10.1 General requirements for both end-to-end and peer-to-peer transparent clocks

All transparent clocks shall forward all non-PTP messages according to the addressing rules of the network.

For all transparent clocks, all PTP version 1 messages:

a) Shall be forwarded according to the addressing rules of the network.

b) Transparent clocks should not make any residence time or path delay corrections to PTP version 1 messages.

c) Residence time and path delay correction of PTP version 1 messages is not precluded but is out of scope of this standard.

All transparent clocks should syntonize to a grandmaster clock per Clause 12.

Syntonization to multiple domains should be supported.

For clocks implementing syntonization, the term primary syntonization domain is defined to be domain-0 or, if the default is configured to a new domain, the configured default domain.

If the transparent clock implements syntonization it shall do so as follows:

a) By default it shall syntonize to a master clock of the primary syntonization domain,

b) The primary syntonization domain may be configured to a domain other than domain 0,

c) If syntonization to multiple domains is implemented, separate syntonization to a master clock in each of the multiple domains shall be maintained.

d) All residence time and link delay corrections made to event messages by a transparent clock shall be based on residence time and link delay measurements that in precedence order:

   1) Are made with a clock syntonized to the same domain as indicated in the domainNumber field of the ingress event message, or

   2) Are made with a clock syntonized to the primary syntonization domain.

If the transparent clock does not implement syntonization, all residence time and link delay corrections made to event messages by a transparent clock shall be based on residence time and link delay measurements made with the unsyntonized free-running clock

10.2 End-to-end transparent clock requirements

All PTP version 2 and higher messages shall be forwarded according to the addressing rules of the network.

The processing of all PTP version 2 and higher event messages and any affected general messages shall meet the residence transit time correction requirements of 11.5.

An end-to-end transparent clock shall not implement peer delay mechanism of 11.4.

10.3 Peer-to-peer transparent clock requirements

The peer delay mechanism of 11.4 shall be implemented.

All PTP version 2 and higher Announce, Sync, Follow_Up, Management and Signaling messages shall be forwarded according to the addressing rules of the network.

The processing of all PTP version 2 and higher Sync and Follow_Up messages shall meet the residence transit time correction requirements of 11.5 and the path delay correction requirements of 11.4.5.

All PTP version 2 and higher Delay_Req and Delay_Resp messages should be discarded.

11. Clock offset, path delay, residence time, and asymmetry corrections

11.1 General specifications

The specifications in Clause 11 provide mechanisms for conveying timestamps generated at the sources of event messages along with any corrections needed to ensure that the recipient of the event message receives the most accurate timestamp possible. The actual distribution of the time information between the originTimestamp or preciseOriginTimestamp and the correctionField fields is implementation dependent, providing the distribution shall be such that a receiving device performing the computations on timestamp fields and correction fields, as specified in the following clauses, obtains the most accurate timestamp possible.

Clause 11 specifies:

a) The computation of the offset in time between a slave and master clock, i.e. offsetFromMaster.

b) The delay request-response mechanism: This mechanism measures the meanPathDelay between a pair of ports, each of which supports the state machine of 9.2.5.

c) The peer delay mechanism: This mechanism measures the meanPathDelay between a pair of ports, each of which supports the peer delay mechanism.

d) The correction for meanPathDelay in peer-to-peer transparent clocks.

e) The correction of event messages in transparent clocks based on the measurement of the residence time within a transparent clock, and

f) The correction of timestamps for path asymmetry.

NOTE 1—The recommendation that the timestamps themselves be the best possible estimate of the time enables simple devices that only need approximate time to ignore correction fields.

NOTE 2—The latitude in the distribution of time between the timestamp and correction field allows flexibility in the device design. For example, a device may generate an approximate time when a message is assembled and insert the resulting required correction into the appropriate correction field. An unavoidable circumstance is the representation of fractional nanoseconds. Fractional nanoseconds cannot be represented in the Timestamp data type and are transmitted in a correction field, leaving it to the receiving device to combine the two to get the actual timestamp.

Examples of these correction mechanisms are in Annex C.

11.2 Computation of clock offset in ordinary and boundary clocks

The time error between a slave and master ordinary or boundary clock is defined as:
offsetFromMaster = <Time on the slave clock> ─ <Time on the master clock> where all times are measured at the same instant.

The offsetFromMaster value shall be computed by the slave as follows:

a) Upon receipt of a Sync message the slave shall generate a timestamp <syncEventIngressTimestamp> corrected for latency per 7.3.4. If the delayAsymmetry, see 7.4.2, of the path connected to the ingress port is known, the corrections of 11.6 shall be made.

b) If the twoStepFlag bit of the flags field of the of the Sync message, is FALSE, indicating that a Follow_Up message will not be received, then the
offsetFromMaster = <syncEventIngressTimestamp> ─ <originTimestamp> – <meanPathDelay> ─ correctionField of Sync message

c) If the twoStepFlag bit of the flags field of the of the Sync message, is TRUE, indicating that a Follow_Up message will be received, then the offsetFromMaster = <syncEventIngressTimestamp> ─ <preciseOriginTimestamp> – <meanPathDelay> ─ correctionField of Sync message ─ correctionField of Follow_Up message

Where:

a) The <originTimestamp> shall be the value of the originTimestamp field in the received Sync message,

b) The <preciseOriginTimestamp> shall be the value of the preciseOriginTimestamp field in the received Follow_Up message,

c) If the port is configured to use the delay request-response mechanism, then the <meanPathDelay> shall be that specified in 11.3

d) If the port is configured to use the peer delay mechanism, then the <meanPathDelay> shall be that specified in 11.4.

11.3 Delay request-response mechanism

11.3.1 Delay request-response mechanism general requirements

The delay request-response mechanism measures the meanPathDelay between a pair of PTP ports each of which supports the state machine of 9.2.5. The delay request-response mechanism uses the messages Sync, Delay_Req, Delay_Resp and possibly Follow_Up as shown in the timing diagram of Figure 34. This mechanism shall be executed independently in each supported domain of the two clocks.

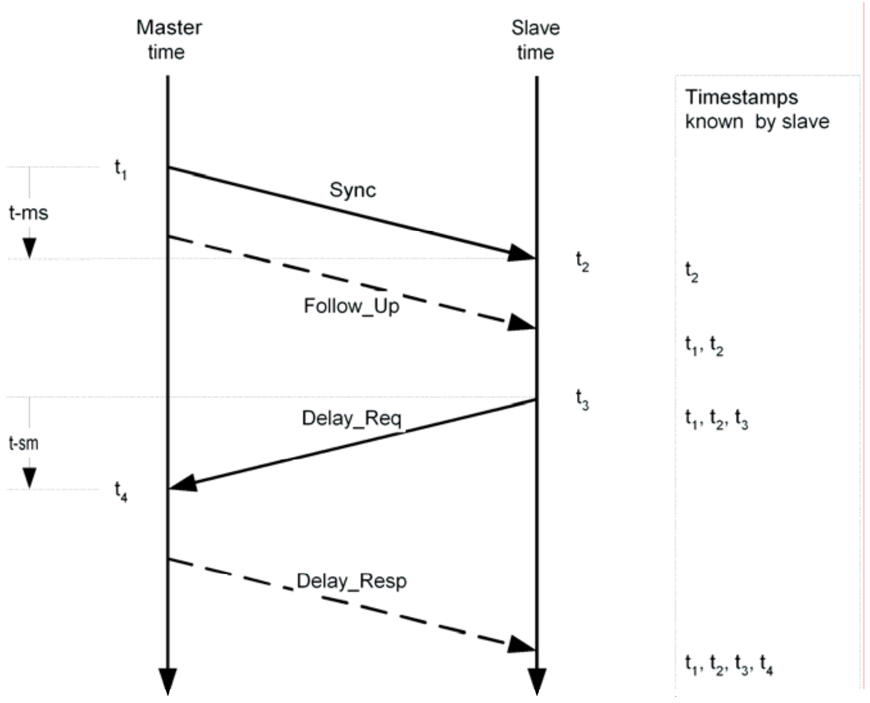


Figure 34: Delay request-response path length measurement

The timestamps t_1 and t_2 for the Sync message and t_3 and t_4 for the Delay_Req message of Figure 34 shall be measured as defined in 7.3.4.2. Timestamps t_1 and t_4 shall be measured using the time of the master node, and the timestamps t_2 and t_3 shall be measured using the time of the slave node.

If the delayAsymmetry, see 7.4.2, of the paths connected to the ingress and egress ports is known, the corrections of 11.6 shall be implemented.

NOTE—The nominal value of the meanPathDelay is computed as meanPathDelay = $[(t_2 – t_1) + (t_4 – t_3)]/2 = [(t_2 – t_3) + (t_4 – t_1)]/2$

11.3.2 Delay request-response mechanism operational specifications

The actual value of the meanPathDelay shall be measured and computed as follows for each instance of a delay request-response measurement:

a) The master node prepares and issues a Sync message per 9.5.9.3. If the node is a two-step clock it also prepares and issues a Follow_Up message per 9.5.9.4.

b) The slave node shall:

   1) Upon receipt of the Sync message from the master generate timestamp t_2

   2) If asymmetry corrections are required, modify the correctionField of the received Sync message per 11.6.2.

   3) Prepare a Delay_Req message with the correctionField, see 13.3.2.7, set to 0. The originTimestamp shall be set to 0 or an estimate no worse than ± 1 second of the egress time of the Delay_Req message.

   4) If asymmetry corrections are required, modify the correctionField per 11.6.3.

   5) Send the Delay_Req message and generate and save timestamp t_3

c) Upon receipt of the Delay_Req message, the master node shall :

   1) Generate timestamp t_4.

   2) Prepare a Delay_Resp message,

   3) Copy the sequenceId field from the Delay_Req message to the sequenceId field of the Delay_Resp message.

   4) Copy the sourcePortIdentity field from the Delay_Req message to the requestingPortIdentity field of the Delay_Resp message.

   5) Copy the domainNumber field from the Delay_Req message to the domainNumber field of the Delay_Resp message.

   6) Set the correctionField of the Delay_Resp message to 0.

   7) Add the correctionField of the Delay_Req message to the correctionField of the Delay_Resp message.

   8) Set the receiveTimestamp field of the Delay_Resp message to the seconds and nanoseconds portion of the time t_4.

   9) Subtract any fractional nanosecond portion of t_4 from the correctionField of the Delay_Resp message.

   10) Issue the Delay_Resp message.

d) Upon receipt of the Delay_Resp message by the slave:

1) If the received Sync message indicated that a Follow_Up message will not be received, the meanPathDelay shall be computed as: meanPathDelay = [(t_2 - t_3) + (receiveTimestamp of Delay_Resp message – originTimestamp of Sync message) – correctionField of Sync message – correctionField of Delay_Resp message]/2.

2) If the received Sync message indicated that a Follow_Up message will be received, the meanPathDelay shall be computed as: meanPathDelay = [(t_2 - t_3) + (receiveTimestamp of Delay_Resp message – preciseOriginTimestamp of Follow_Up message) – correctionField of Sync message– correctionField of Follow_Up message – correctionField of Delay_Resp message]/2.

NOTE— The delay request-response path length measurement normally uses the timestamps and correction fields of the most recent Sync and corresponding Follow_Up message prior to the Delay_Req message. However, the delay request-response measurement may use any Sync and corresponding Follow_Up message, although this will reduce the accuracy of the computed meanPathDelay.

11.4 Peer delay mechanism

11.4.1 Peer delay mechanism general requirements

The peer delay mechanism measures the port-to-port propagation time, i.e. the link delay, between two communicating ports supporting the peer delay mechanism.

This measurement should be made on all ports of a device including those that are blocked by lower level protocols. The addressing or other mechanisms to support this requirement are network specific and are specified in the relevant Annexes to this standard.

The link delay measurement shall be made independently by each port implementing the peer delay mechanism.

NOTE—This requirement means that the link delay is known by ports on both ends of a link. This allows path length corrections to be made immediately upon reconfiguration of the network.

In ordinary and boundary clocks, the peer delay mechanism shall be independent of whether the port is a master or a slave.

The peer delay mechanism uses the messages Pdelay_Req, Pdelay_Resp, and possibly Pdelay_Resp_Follow_Up as shown in the timing diagram of Figure 35.

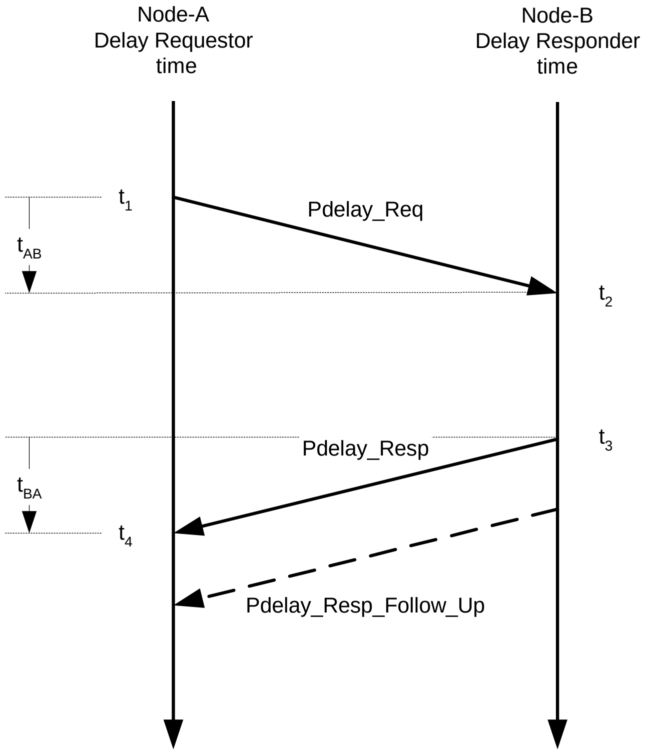


Figure 35: Peer delay link measurement

The timestamps t_1 and t_2 for the Pdelay_Req message and t_3 and t_4 for the Pdelay_Resp message of Figure 35 are measured as defined in 7.3.4. If the delayAsymmetry, see 7.4.2, of the links connected to the ingress and egress ports are known, the corrections of 11.6 shall be implemented.

The timestamps t_1 and t_4 shall be measured by Node-A as follows:

- If Node-A is a peer-to-peer transparent clock the timescale used is specified in 10.1.
- If Node-A is an ordinary or boundary clock the timescale used is that of the domain of the clock.

Timestamps t_2 and t_3 shall be measured by Node-B as follows:

- If Node-B is a peer-to-peer transparent clock the timescale used is specified in 10.1.
- If Node-B is an ordinary or boundary clock the timescale used is that of the domain of the clock.

NOTE—The nominal value of the meanPathDelay is computed as meanPathDelay = $[(t_2 - t_1) + (t_4 - t_3)]/2 = [(t_2 - t_3) + (t_4 - t_1)]/2$ The actual value is specified in 11.4.3.

11.4.2 Peer delay message timing

The transmission of a Pdelay_Req message from a requesting port shall be limited as follows:

- The initial Pdelay_Req message may be transmitted when required.
- Subsequent Pdelay_Req messages shall be transmitted such that the logarithm to the base 2 of the mean value of the interval in seconds between message transmissions is not less than the value of the logMinPdelayReqInterval member of the portDS data set.

Pdelay_Resp messages should be transmitted as soon as possible after the receipt of the associated Pdelay_Req message.

Pdelay_Resp_Follow_Up messages should be transmitted as soon as possible after the transmission of the associated Pdelay_Resp message.

11.4.3 Peer delay mechanism operational specifications

The actual value of the meanPathDelay shall be measured and computed as follows for each instance of a peer delay request-response measurement:

a) The delay requestor, Node-A:

   1) Prepares a Pdelay_Req message. The correctionField, see 13.3.2.7, shall be set to 0.

      i) If Node-A is an ordinary or boundary clock then the domainNumber field of the header shall be set to the domain of Node-A.

      ii) If Node-A is a syntonized peer-to-peer transparent clock, the domainNumber field of the header should be set to the domain being measured, either the primary syntonization domain, or one of the alternate domains if syntonization to multiple domains is implemented on Node-A.

      iii) If Node-A is not a syntonized peer-to-peer transparent clock the domainNumber field of the header shall be set to 0.

   2) If asymmetry corrections are required, shall modify the correctionField per 11.6.4.

   3) Shall set the originTimestamp to 0 or an estimate no worse than ± 1 second of the egress timestamp, t_1, of the Pdelay_Req message.

   4) Shall send the Pdelay_Req message and generate and save timestamp t_1

b) If the delay responder, Node-B, is a one-step clock it shall:

   1) Generate timestamp t_2 upon receipt of the Pdelay_Req message.

   2) Prepare a Pdelay_Resp message,

   3) Copy the sequenceId field from the Pdelay_Req message to the sequenceId field of the Pdelay_Resp message.

   4) Copy the sourcePortIdentity field from the Pdelay_Req message to the requestingPortIdentity field of the Pdelay_Resp message.

   5) Copy the domainNumber field from the Pdelay_Req message to the domainNumber field of the Pdelay_Resp message

   6) Copy the correctionField from the Pdelay_Req message to the correctionField of the Pdelay_Resp message.

   7) Then:

      i) Set to 0 the requestReceiptTimestamp field of the Pdelay_Resp message.

      ii) Issue the Pdelay_Resp message and generate timestamp t_3 upon sending.

      iii) After t_3 is generated but while the Pdelay_Resp message is leaving the responder, shall add the turnaround time t_3 - t_2 to the correctionField of the Pdelay_Resp message and make any needed corrections to checksums or other content dependent fields of the Pdelay_Resp message.

NOTE—The data type of the correctionField allows the time interval t3 - t2 to be expressed to a fraction of a nanosecond if needed and this accuracy is supported by the Responder.

c) If the delay responder is a two-step clock it shall:

1) Generate timestamp t_2 upon receipt of the Pdelay_Req message.

2) Prepare a Pdelay_Resp and a Pdelay_Resp_Follow_Up message,

3) Copy the correctionField from the Pdelay_Req message to the correctionField of the Pdelay_Resp_Follow_Up message and set correctionField of the Pdelay_Resp message to 0.

4) Copy the sequenceId field from the Pdelay_Req message to the sequenceId field of the Pdelay_Resp and the Pdelay_Resp_Follow_Up messages.

5) Copy the sourcePortIdentity field from the Pdelay_Req message to the requestingPortIdentity field of the Pdelay_Resp and the Pdelay_Resp_Follow_Up messages.

6) Copy the domainNumber field from the Pdelay_Req message to the domainNumber field of the Pdelay_Resp and Pdelay_Resp_Follow_Up messages.

7) Then either:

   i) Set to 0 the requestReceiptTimestamp fields of the Pdelay_Resp message.

   ii) Issue the Pdelay_Resp message and generates timestamp t_3 upon sending.

   iii) In the Pdelay_Resp_Follow_Up message, set the responseOriginTimestamp field to 0, and add the turnaround time t_3 - t_2 to the correctionField.

   iv) Issue the Pdelay_Resp_Follow_Up message.

8) Or,

   i) In the Pdelay_Resp message, set the requestReceiptTimestamp field to the seconds and nanoseconds portion of the time t_2, and subtract any fractional nanosecond portion of t_2 from the correctionField.

   ii) Issue the Pdelay_Resp message and generate timestamp t_3 upon sending.

   iii) In the Pdelay_Resp_Follow_Up message, set the responseOriginTimestamp field to the seconds and nanoseconds portion of the time t_3, and add any fractional nanosecond portion of t_3 to the correctionField.

   iv) Issue the Pdelay_Resp_Follow_Up message.

d) The delay requestor, Node-A, shall:

1) Generate timestamp t_4 upon receipt of the Pdelay_Resp message.

2) If asymmetry corrections are required, modify the correctionField of the Pdelay_Resp message per 11.6.5.

3) If the twoStepFlag of the received Pdelay_Resp message is FALSE, indicating that no Pdelay_Resp_Follow_Up message will be received, compute the meanPathDelay as: meanPathDelay = [(t_4 - t_1) ─ correctionField of Pdelay_Resp]/2.

4) If the twoStepFlag of the received Pdelay_Resp message is TRUE indicating that a Pdelay_Resp_Follow_Up will be received, compute the meanPathDelay as: meanPathDelay = [(t_4 - t_1) ─ (responseOriginTimestamp – requestReceiptTimestamp) ─ correctionField of Pdelay_Resp ─ correctionField of Pdelay_Resp_Follow_Up]/2.

The delay responder, Node-B upon receipt of a Pdelay_Req message, shall issue the associated Pdelay_Resp message as quickly as possible, consistent with the other requirements of 11.4.3, to minimize the turnaround time t_3 - t_2.

NOTE—Errors introduced by excessive values of the turnaround time are reduced by syntonization per 10.1.

11.4.4 Restriction on the use of the peer delay mechanism

A delay requestor, Node-A, may receive 0, 1, or multiple Pdelay_Resp messages for each transmitted Pdelay_Req.

Multiple responses can be detected by observing that the sourcePortIdentity field of the Pdelay_Resp messages differ.

NOTE—Multiple responses can occur if there is an end-to-end transparent clock or an ordinary bridge or other similar multicast and multi-port devices between Node-A and multiple Node-B devices. While the multiple responses can be distinguished, there is no mechanism in this standard that allows the path length associated with each of the responses from the multiple Node-B devices to be correctly assigned to a received Sync message.

The following actionsshould be taken in these cases:

a) When no Pdelay_Resp is received, Node-A should periodically retransmit a Pdelay_Req message to check for the appearance of a Node-B. The retransmission rate in this case is implementation-specific.

b) When a single Pdelay_Resp is received the protocol of 11.4 should be executed as specified.

c) When multiple Pdelay_Resp messages are received, Node-A shall either:

   1) Enter the FAULTY state if an ordinary or boundary clock, or a fault condition if a peer-to-peer transparent clock. In this case the device may periodically retransmit a Pdelay_Req message to check for the resolution of this condition. The retransmission rate in this case is implementation-specific. In this case Sync and Follow_Up messages received on the port shall be disregarded.

   2) Take implementation-specific measures to resolve the issue.

11.4.5 Path delay correction in transparent clocks

11.4.5.1 Peer-to-peer transparent clocks

A port on a peer-to-peer transparent clock upon receiving a Sync message shall:

a) If the clock is a one-step peer-to-peer clock, add the value of the <meanPathDelay>, as measured by peer delay mechanism for the link connected to the ingress port on which the Sync message was received to the correctionField of the Sync message prior to the completion of the transmission of the Sync message on each of the egress ports.

b) If the clock is a two-step peer-to-peer clock, add the value of the <meanPathDelay>, as measured by peer delay mechanism, for the link connected to the ingress port on which the Sync message was received to the correctionField of the associated Follow_Up message prior to the transmission of the Follow_Up message on each of the egress ports. The <meanPathDelay> shall be added subsequent to any residence time corrections, see 11.5.2.2.

NOTE 1—To correctly associate Sync and Follow_Up messages requires that the transparent clock maintain a record of the sourcePortIdentity, and sequenceId fields of the Sync message for comparison with the sourcePortIdentity and sequenceId fields of Follow_Up messages.

NOTE 2—The data type of the correctionField allows the <meanPathDelay> to be expressed to a fraction of a nanosecond if this accuracy is supported by the transparent clock.

11.4.5.2 End-to-end transparent clocks

An end-to-end transparent clock shall not correct for path delay on either ingress or egress ports.

11.5 Transparent clock residence time correction for PTP version 2 or higher events

11.5.1 Residence time computation

A transparent clock shall generate an ingress timestamp for all version 2 or higher event messages, see 7.3.4.2, indicating the time of receipt of the event message on the ingress port.

A transparent clock shall generate an egress timestamp for all version 2 or higher event messages, see 7.3.4.2, indicating the time of transmission of the event message on the egress port.

NOTE—In general the egress timestamp has a different value on each egress port of the transparent clock.

All timestamps shall be measured in the domain as specified in 10.1.

If the delayAsymmetry, see 7.4.2, of the paths connected to the ingress and egress ports is known, the corrections of 11.6 shall be implemented.

The residence time for each such event message shall be computed for each egress port as:
<residenceTime> = egress timestamp – ingress timestamp.

11.5.2 Residence time correction for Sync messages

11.5.2.1 One-step transparent clocks

The <residenceTime> shall be added to the correctionField of the Sync event message by the egress port of the clock as the Sync event message is being transmitted. The egress port shall make any needed corrections to checksums or other content dependent fields of the message.

NOTE—The data type of the correctionField allows the <residenceTime> to be expressed to a fraction of a nanosecond if this accuracy is supported by the transparent clock.

No modification of any received Follow_Up messages shall be made.

11.5.2.2 Two-step transparent clocks

If the twoStepFlag of the received Sync message is FALSE indicating that no Follow_Up message will be received then:

a) The twoStepFlag of the received Sync message shall be set to TRUE to indicate that a Follow_Up message will follow. The Sync message should be transmitted on the egress port as soon as possible, with any needed corrections to checksums or other content dependent fields of the message. This modified Sync message shall be used to generate the egress timestamp for the computation of residence time for the Sync message.

b) A Follow_Up message shall be prepared for transmission on the egress port as follows:

    1) The originTimestamp of the received Sync message shall be copied into the preciseOriginTimestamp field of the Follow_Up message,

    2) The sequenceId of the received Sync message shall be copied into the sequenceId of the Follow_Up message.

    3) The sourcePortIdentity of the received Sync message shall be copied to the sourcePortIdentity of the Follow_Up message.

4) The domainNumber of the received Sync message shall be copied to the domainNumber of the Follow_Up message.

5) The logMessageInterval of the received Sync message shall be copied to the logMessageInterval of the Follow_Up message.

6) The header flags field of the received Sync message shall be copied to the flags field of the Follow_Up message but with the twoStepFlag set to TRUE.

7) The correctionField of the Follow_Up message shall be set to the <residenceTime>. For peer-to-peer transparent clocks this shall be done prior to any corrections for <meanPathDelay>, see 11.4.5.1.

NOTE—The data type of the correctionField allows the <residenceTime> to be expressed to a fraction of a nanosecond if this accuracy is supported by the transparent clock.

If the twoStepFlag of the received Sync message is TRUE, indicating that a Follow_Up message will be received, then:

a) The received Sync message shall be transmitted from the egress port. This Sync message shall be used to generate the egress timestamp for the computation of residence time for the Sync message.

b) The <residenceTime> shall be added to the correctionField of the Follow_Up message associated with the Sync message prior to transmission on the egress port.

NOTE—To correctly associate Sync and Follow_Up messages requires that the transparent clock maintain a record of the sourcePortIdentity, and sequenceId fields of the Sync message for comparison with the sourcePortIdentity and sequenceId fields of Follow_Up messages. The data type of the correctionField allows the <residenceTime> to be expressed to a fraction of a nanosecond if this accuracy is supported by the transparent clock.

11.5.3 Residence time correction for Delay_Req messages

11.5.3.1 General specification

Subclause 11.5.3 applies to end-to-end transparent clocks. Peer-to-peer transparent clocks do not support Delay_Req messages.

11.5.3.2 One-step end-to-end transparent clocks

The <residenceTime> shall be added to the correctionField of the Delay_Req message by the egress port of the clock as the Delay_Req message is being transmitted. The egress port shall make any needed corrections to checksums or other content dependent fields of the message.

NOTE—The data type of the correctionField allows the <residenceTime> to be expressed to a fraction of a nanosecond if this accuracy is supported by the transparent clock.

No modification of any received Delay_Resp messages shall be made.

11.5.3.3 Two-step end-to-end transparent clocks

The received Delay_Req message shall be transmitted from the egress port. This Delay_Req message shall be used to generate the egress timestamp for the computation of <residenceTime> for the Delay_Req message.

The <residenceTime> shall be added to the correctionField of the Delay_Resp message associated with the Delay_Req message prior to transmission of the Delay_Resp message on the egress port.

NOTE—To correctly associate Delay_Req and Delay_Resp messages requires that the transparent clock maintain a record of the sourcePortIdentity, and sequenceId fields of the Delay_Req message for comparison with the requestingPortIdentity and sequenceId fields of Delay_Resp messages. The data type of the correctionField allows the <residenceTime> to be expressed to a fraction of a nanosecond if this accuracy is supported by the transparent clock.

11.5.4 Residence time correction for Pdelay_Req and Pdelay_Resp messages

11.5.4.1 General

Subclause 11.5.4 applies to end-to-end transparent clocks. Pdelay_Req and Pdelay_Resp messages terminate at peer-to-peer transparent clocks.

NOTE—peer-to-peer clocks are normally used only in a homogeneous system of peer-to-peer clocks. The provisions of 11.5.4 allow the use of end-to-end transparent clocks in systems based on future versions of the standard that specify how to implement mixed systems with one-to-many connections between peer-to-peer clocks.

11.5.4.2 One-step end-to-end transparent clocks

The <residenceTime> of the Pdelay_Req message shall be added to the correctionField of the Pdelay_Req message by the egress port of the clock as the Pdelay_Req message is being transmitted. The egress port shall make any needed corrections to checksums or other content dependent fields of the message.

No modification of any received Pdelay_Resp_Follow_Up or Pdelay_Resp messages shall be made for the <residenceTime> of a Pdelay_Req message.

The <residenceTime> of a Pdelay_Resp shall be added to the correctionField of the Pdelay_Resp message by the egress port of the clock as the Pdelay_Resp message is being transmitted. The egress port shall make any needed corrections to checksums or other content dependent fields of the message.

No modification of any received Pdelay_Resp_Follow_Up messages shall be made for the <residenceTime> of a Pdelay_Resp message.

11.5.4.3 Two-step end-to-end transparent clocks

The <residenceTime> of the Pdelay_Req message shall be measured and saved for incorporation into the correctionField of a Pdelay_Resp_Follow_Up message associated with the Pdelay_Req message.

The <residenceTime> of a Pdelay_Resp message associated with the Pdelay_Req message shall be measured and saved for incorporation into the correctionField of the Pdelay_Resp_Follow_Up message associated with the Pdelay_Req message.

If the twoStepFlag of the received Pdelay_Resp message associated with the Pdelay_Req message is FALSE, indicating that no Pdelay_Resp_Follow_Up message will be received, then:

a) The twoStepFlag of the received Pdelay_Resp message shall be set to TRUE to indicate that a Pdelay_Resp_Follow_Up message will follow. The Pdelay_Resp message should be transmitted as soon as possible, with any needed corrections to checksums or other content dependent fields of the message. This modified Pdelay_Resp message shall be used to generate the egress timestamp for the computation of the <residenceTime> for the Pdelay_Resp message.

b) A Pdelay_Resp_Follow_Up message shall be prepared for transmission as follows:

   1) Copy the sequenceId of the received Pdelay_Resp message into the sequenceId of the Pdelay_Resp_Follow_Up message.

2) Copy the requestingPortIdentity field from the Pdelay_Resp message to the requestingPortIdentity field of the Pdelay_Resp_Follow_Up message.

3) Set the responseOriginTimestamp to 0.

4) Copy the domainNumber field from the Pdelay_Resp message to the domainNumber field of the Pdelay_Resp_Follow_Up message.

5) The correctionField of the Pdelay_Resp_Follow_Up message shall be set to the sum of the <residenceTime> of the Pdelay_Resp and the <residenceTime> of the Pdelay_Req message associated with the Pdelay_Resp message.

If the twoStepFlag of the received Pdelay_Resp message is TRUE, indicating that a Pdelay_Resp_Follow_Up message will be received, then:

a) The received Pdelay_Resp message shall be transmitted from the egress port. This Pdelay_Resp message shall be used to generate the egress timestamp for the computation of residence time for the Pdelay_Resp message.

b) The sum of the <residenceTime> of the Pdelay_Resp and the <residenceTime> of the Pdelay_Req message associated with the Pdelay_Resp message shall be added to the correctionField of the Pdelay_Resp_Follow_Up message associated with the Pdelay_Req message prior to transmission on the egress port.

NOTE 1—To correctly associate Pdelay_Req, Pdelay_Resp, and Pdelay_Resp_Follow_Up messages requires that the transparent clock maintain a record of the sourcePortIdentity, and sequenceId fields of the Pdelay_Req message for comparison with the requestingPortIdentity and sequenceId fields of Pdelay_Resp and Pdelay_Resp_Follow_Up messages.

NOTE 2—The data type of the correctionField allows the <residenceTime> to be expressed to a fraction of a nanosecond if this accuracy is supported by the transparent clock.

11.6 Asymmetry correction for PTP version 2 or higher event messages

11.6.1 General specification

If the delayAsymmetry, see 7.4.2, of the paths connected to the ingress and egress ports of a clock is known, then PTP messages shall be corrected as specified below.

11.6.2 Asymmetry correction for Sync messages

Sync messages:

a) Shall not be corrected for asymmetry for the path connected to an egress port.

b) For a boundary or ordinary clock, upon receipt on an ingress port shall be corrected for asymmetry of the path connected to the ingress port by adding the value of the ingress path delayAsymmetry to the correctionField of the received Sync message prior to any use of the correctionField in a computation.

c) For a transparent clock, upon receipt on an ingress port shall be corrected for asymmetry of the path connected to the ingress port prior to transmission of the Sync message on an egress port of the transparent clock.

1) If the transparent clock is a one-step clock, the correction shall be made by adding the value of the ingress path delayAsymmetry to the correctionField of the received Sync message prior to transmission on an egress port.

2) If the transparent clock is a two-step clock, the correction shall be made by adding the value of the ingress path delayAsymmetry to the correctionField of the Follow_Up message associated with the received Sync message prior to transmission of the Follow_Up message on an egress port.

NOTE— This Follow_Up message may be generated by a two-step clock upstream from the end-to-end transparent clock or it may be generated by the end-to-end transparent clock itself. This requires that the transparent clock maintain a record of the sourcePortIdentity, and sequenceId fields of the Sync message for comparison with the sourcePortIdentity and sequenceId fields of Follow_Up messages.

11.6.3 Asymmetry correction for Delay_req messages

Delay_Req messages:

a) Shall not be corrected for asymmetry for the path connected to an ingress port.

b) For a boundary or ordinary clock, prior to transmission on an egress port the correctionField of the transmitted Delay_Req message shall be modified by subtracting the value of the egress path delayAsymmetry from the correctionField of the transmitted Delay_Req message.

c) For an end-to-end transparent clock, upon receipt on an ingress port and prior to the subsequent transmission of the Delay_Req message on an egress port:

   1) If the end-to-end transparent clock is an one-step clock, the correction shall be made by subtracting the value of the egress path delayAsymmetry from the correctionField of the received Delay_Req message prior to transmission on an egress port.

   2) If the end-to-end transparent clock is a two-step clock, the correction shall be made by subtracting the value of the egress path delayAsymmetry of the path connected from the egress port of the Delay_Req message from the correctionField of the Delay_Resp message associated with the original Delay_Req message prior to transmission of the Delay_Resp message.

11.6.4 Asymmetry correction for Pdelay_req messages

Pdelay_Req messages:

a) Shall not be corrected for asymmetry for the path connected to an ingress port.

b) For a boundary clock, ordinary clock, or P2P transparent clock, prior to transmission on an egress port the correctionField of the transmitted Pdelay_Req message shall be modified by subtracting the value of the egress path delayAsymmetry from the correctionField of the transmitted Pdelay_Req message.

c) For an end-to-end transparent clock, upon receipt on an ingress port and prior to the subsequent transmission of the Pdelay_Req message on an egress port:

   1) If the end-to-end transparent clock is an one-step clock, the correction shall be made by subtracting the value of the egress path delayAsymmetry from the correctionField of the received Pdelay_Req message prior to transmission on an egress port.

   2) If the end-to-end transparent clock is a two-step clock, the correction shall be made by subtracting the value of the egress path delayAsymmetry of the path connected to the egress port of the Pdelay_Req message from the correctionField of the Pdelay_Resp_Follow_Up message associated with the original Pdelay_Req message prior to transmission of the Pdelay_Resp_Follow_Up message.

NOTE—This Pdelay_Resp_Follow_Up message may be generated by a two-step clock upstream from the end-to-end transparent clock or it may be generated by the end-to-end transparent clock itself. This requires that the transparent clock maintain a record of the sourcePortIdentity, and sequenceId fields of the Delay_Req message for comparison with the requestingSourcePortIdentity and sequenceId fields of Delay_Resp messages.

11.6.5 Asymmetry correction for Pdelay_Resp messages

Pdelay_Resp messages:

a) Shall not be corrected for asymmetry for the path connected to an egress port.

b) For a boundary, ordinary, or peer-to-peer clock, upon receipt on an ingress port shall be corrected for asymmetry of the path connected to the ingress port by adding the value of the ingress path delayAsymmetry to the correctionField of the received Pdelay_Resp message prior to any use of the correctionField in a computation.

c) For an end-to-end transparent clock, upon receipt on an ingress port shall be corrected for asymmetry of the path connected to the ingress port prior to transmission of the Pdelay_Resp message on an egress port of the end-to-end transparent clock.

    1) If the end-to-end transparent clock is an one-step clock, the correction shall be made by adding the value of the ingress path delayAsymmetry to the correctionField of the received Pdelay_Resp message prior to transmission on an egress port.

    2) If the end-to-end transparent clock is a two-step clock, the correction shall be made by adding the value of the ingress path delayAsymmetry to the correctionField of the Pdelay_Resp_Follow_Up message associated with the transmitted Pdelay_Resp message prior to transmission of the Pdelay_Resp_Follow_Up message on an egress port.

NOTE—This Pdelay_Resp_Follow_Up message may be generated by a two-step clock upstream from the end-to-end transparent clock or it may be generated by the end-to-end transparent clock itself. This requires that the transparent clock maintain a record of the requestingPortIdentity, and sequenceId fields of the Pdelay_Resp message for comparison with the requestingPortIdentity and sequenceId fields of Pdelay_Resp_Follow_Up messages.

12. Synchronization and syntonization of clocks

12.1 Syntonization

12.1.1 General specification

Within a domain, any clock with a port in the SLAVE state, and any transparent clock should syntonize to the grandmaster.

12.1.2 Syntonization based on Sync messages

A clock, A, may syntonize to another clock, B, as follows.

For a sequence of Sync, and possibly Follow_Up messages from clock B, clock A computes <correctedMasterEventTimestamp> and <syncEventIngressTimestamp> as follows:

a) Upon receipt of a Sync message, clock A generates and records a timestamp <syncEventIngressTimestamp> corrected for latency per 7.3.4. If the delayAsymmetry, see 7.4.2, of the path connected to the ingress port is known, the corrections of 11.6 shall be made.

b) If the twoStepFlag bit of the flags field of the of the Sync message is FALSE indicating that a Follow_Up message will not be received, then the <correctedMasterEventTimestamp> = <originTimestamp> + <meanPathDelay> + correctionField of Sync message.

c) If the twoStepFlag bit of the flags field of the of the Sync message is TRUE indicating that a Follow_Up message will be received, then the <correctedMasterEventTimestamp> = <preciseOriginTimestamp> + <meanPathDelay> + correctionField of Sync message + correctionField of Follow_Up message.

Where:

a) The <originTimestamp> is the value of the originTimestamp field in the received Sync message,

b) The <preciseOriginTimestamp> is the value of the preciseOriginTimestamp field in the received Follow_Up message,

c) If the port is configured to use the delay request-response mechanism, then the <meanPathDelay> shall be that specified in 11.3,

d) If the port is configured to use the peer delay mechanism, then the <meanPathDelay> is that specified in 11.4.

The rate of change of clock A's time is then adjusted using the sequence of <syncEventIngressTimestamps> and the sequence of <correctedMasterEventTimestamps> to align it to the rate of change of the grandmaster's time.

NOTE—As one example, the ratio of the master frequency to the frequency of clock A can be estimated as the ratio of the elapsed time of clock A to the elapsed time of the master clock between a received timestamp and a second received timestamp some number of sync intervals later , i.e.,

$$\frac{< \text{sync} - \text{event} - \text{ingress} - \text{timestamp} >_N - < \text{sync} - \text{event} - \text{ingress} - \text{timestamp} >_0}{< \text{corrected} - \text{master} - \text{event} - \text{timestamp} >_N - < \text{corrected} - \text{master} - \text{event} - \text{timestamp} >_0},$$

where *N* is the number of sync intervals separating the timestamps (*N* > 0). The frequency of clock A can then be adjusted by this factor.

12.1.3 Syntonization based on other mechanisms

In some networks there may be physical signals accessible to clocks that can be used to syntonize two clocks. Such signals may be used provided the syntonization of clock A to clock B matches the synchronization hierarchy established by the best master clock algorithm, see 9.3, which means that Sync messages proceed from clock B to clock A. These means are out of scope of this standard.

12.2 Synchronization

Within a domain, an ordinary or boundary clock with a port in the Slave state shall synchronize to its master in the synchronization hierarchy established by the best master clock algorithm. The specific means for synchronization are out of scope of this standard but shall result in the minimization of the offsetFromMaster value computed by the slave per 11.2.

13. PTP message formats

13.1 General

The formats and definitions in Clause 13 define how the originator fills in these fields. What the receiver does with them is defined elsewhere in this standard.

In the tables in Clause 13, the ‘octets’ column indicates the size of the field in octets. The ‘offset’ column indicates the offset of the first octet of the field from the start of the PTP defined fields of the message.

13.2 General message format requirements

All messages shall have a header, body and suffix. The suffix may have zero length.

Reserved fields shall be transmitted with the all bits of the field 0 and ignored by the receiver.

Unless otherwise specified, all field values either:

- Shall be an inherent characteristic of the message and specified in Clause 13, or
- Shall be instantiated by the device originating the message, the originating node, based on the data sets or protocol operation of the originating device, or
- In the case of two-step transparent clocks, any generated Follow_Up or Pdelay_Resp_Follow_Up messages shall, except for adjustments required for the operation of the transparent clock, appear as though the messages were generated by the originator of the Sync and Pdelay_Resp messages respectively, see Clause 11.

The data type of the field shall be the type indicated in brackets in the title of each subclause.

13.3 Header

13.3.1 General header specifications

The common header for all PTP messages shall be as specified in Table 18.

Table 18: Common message header

Bits								**Octets**	**Offset**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
transportSpecific				messageType				1	0
reserved				versionPTP				1	1
messageLength								2	2
domainNumber								1	4
reserved								1	5
flags								2	6
correctionField								8	8
reserved								4	16
sourcePortIdentity								10	20
sequenceId								2	30
controlField								1	32
logMessageInterval								1	33

13.3.2 header field specifications

13.3.2.1 transportSpecific (Nibble)

The transportSpecific field may be used by a lower layer transport protocol and is defined by the mapping specification of that protocol in an annex of this standard.

13.3.2.2 messageType (Enumeration4)

The value shall indicate the type of the message as defined in Table 19.

Table 19: Values of messageType field

Message type	Message class	Value(hex)
Sync	Event	0
Delay_Req	Event	1
Pdelay_Req	Event	2
Pdelay_Resp	Event	3
Reserved	—	4-7
Follow_Up	General	8
Delay_Resp	General	9
Pdelay_Resp_Follow_Up	General	A
Announce	General	B
Signaling	General	C
Management	General	D
Reserved	—	E-F

The most significant bit of the message identification (ID) field divides this field in half between event and general messages.

NOTE— The reserved nibble immediately following messageType is reserved for future expansion of the messageType field.

13.3.2.3 versionPTP (UInteger4)

The value of the versionPTP field shall be the value of the versionNumber member of the portDS data set.

13.3.2.4 messageLength (UInteger16)

The value of the messageLength shall be the total number of octets that form the PTP message. The counted octets start with the first octet of the header and include and terminate with the last octet of any Suffix or, if there are no Suffix members with the last octet of the message as defined in Clause 13.

NOTE—The message length does not include any padding bits specified in Annex D.

13.3.2.5 domainNumber (UInteger8)

For ordinary or boundary clocks, the value shall be the value of the domainNumber member of the defaultDS data set of the originating ordinary or boundary clock.

For peer delay mechanism messages originating from a peer-to-peer transparent clock, the value shall be the value defined in 11.4.3.

For management messages, the value shall be the value defined in 15.4.1.1.

13.3.2.6 Flags (Octet[2])

The value of the bits of the array shall be as defined in Table 20. For message types where the bit is not defined in Table 20, the values shall be FALSE.

Table 20: Values of flags field

Octet	Bit	Message types	Name	Description
0	0	Announce, Sync, Follow_Up, Delay_Resp	alternateMasterFlag	FALSE if the port of the originator is in the MASTER state. Conditions to set the flag to TRUE are specified in subclauses 17.3 and 17.4.
0	1	Sync, Pdelay_Resp	twoStepFlag	For a one-step clock, the value of twoStepFlag shall be FALSE. For a two-step clock, the value of twoStepFlag shall be TRUE.
0	2	ALL	unicastFlag	TRUE, if the transport layer protocol address to which this message was sent is a unicast address. FALSE, if the transport layer protocol address to which this message was sent is a multicast address.
0	5	ALL	PTP profile Specific 1	As defined by an alternate PTP profile; otherwise FALSE
0	6	ALL	PTP profile Specific 2	As defined by an alternate PTP profile; otherwise FALSE
0	7	ALL	Reserved	See Note
1	0	Announce	LI_61	The value of leap61 of timePropertiesDS data set
1	1	Announce	LI_59	The value of leap59 of timePropertiesDS data set
1	2	Announce	currentUtcOffsetValid	The value of currentUtcOffsetValid of the timePropertiesDS data set.
1	3	Announce	ptpTimescale	The value of ptpTimescale of the timePropertiesDS data set.
1	4	Announce	timeTraceable	The value of timeTraceable of the timePropertiesDS data set.
1	5	Announce	frequencyTraceable	The value of frequencyTraceable of the timePropertiesDS data set.
NOTE—This bit is reserved for the experimental security mechanism of Annex K				

All unused flags are reserved.

13.3.2.7 correctionField (Integer64)

The correctionField is the value of the correction measured in nanoseconds and multiplied by 2^{16}. For example, 2.5 ns is represented as 0000000000028000_{16}.

A value of one in all bits, except the most significant, of the field, shall indicate that the correction is too big to be represented.

The value of the correctionField depends on the message type as described in Table 21.

Table 21: Correction field semantics

Message type	correctionField description
Sync	Corrections for fractional nanoseconds, residence time in transparent clocks, see 11.5.2, path delay in peer-to-peer clocks, see 11.4.5.1, and asymmetry corrections, see 11.6.2
Delay_Req	Corrections for fractional nanoseconds, residence time in transparent clocks, see 11.5.3, and asymmetry corrections, see 11.6.3
Pdelay_Req	Corrections for fractional nanoseconds, residence time in transparent clocks, see 11.5.4, and asymmetry corrections, see 11.6.4
Pdelay_Resp	Corrections for fractional nanoseconds, residence time in transparent clocks, see 11.5.4, and asymmetry corrections, see 11.6.5
Follow_Up	Corrections for fractional nanoseconds, residence time in transparent clocks, see 11.5.2, path delay in peer-to-peer clocks, see 11.4.5.1, and asymmetry corrections- see 11.6.2
Delay_Resp	Corrections for fractional nanoseconds, residence time in transparent clocks, see 11.5.3, and asymmetry corrections, see 11.6.3
Pdelay_Resp_Follow_Up	Corrections for fractional nanoseconds, residence time in transparent clocks, see 11.5.4, and asymmetry corrections, see 11.6.4 and 11.6.5
Announce	Zero
Signaling	Zero
Management	Zero

13.3.2.8 sourcePortIdentity (PortIdentity)

The value of the sourcePortIdentity field shall be the value of the portIdentity member of the portDS data set of the port that originated this message.

13.3.2.9 sequenceId (UInteger16)

The value of the sequenceId field shall be assigned by the originator of the message in conformance with 7.3.7 except in the case of Follow_Up, Delay_Resp, Pdelay_Resp, and Pdelay_Resp_Follow_Up messages and management messages that are a response to another management message. The sequenceId field values for these exceptions are defined in the references listed in Table 22.

Table 22: References for sequenceId value exceptions

Message type	Reference
Follow_Up	9.5.10, 11.5.2.2
Delay_Resp	11.3.2
Pdelay_Resp	11.4.3
Pdelay_Resp_Follow_Up	11.4.3
Management	15.4.1.2

13.3.2.10 controlField (UInteger8)

The value of controlField depends on the message type defined in the messageType field, see 13.3.2.2, and shall have the value specified in Table 23. The use of this field by the receiver is deprecated.

NOTE—This field is provided for compatibility with hardware designed to conform to version 1 of this standard.

Table 23: controlField enumeration

Message type	controlField value$_{16}$
Sync	0

Delay_Req	1
Follow_Up	2
Delay_Resp	3
Management	4
All others	5
reserved	6-FF

13.3.2.11 logMessageInterval (Integer8)

The value of the logMessageInterval field is determined by the type of the message and shall be as defined in Table 24.

Table 24: Values of logMessageInterval field

Message type	**Value of logMessageInterval**
Announce	The value of the logAnnounceInterval member of the portDS data set
Sync, Follow_Up	The value of the logSyncInterval member of the portDS data set in a multicast message, and $7F_{16}$ in a unicast message
Delay_Resp	the value of the logMinDelayReqInterval member of the portDS data set In a multicast message, and $7F_{16}$ in a unicast message
Delay_Req	$7F_{16}$
Signaling	$7F_{16}$
Management	$7F_{16}$
Pdelay_Req	$7F_{16}$
Pdelay_Resp,	$7F_{16}$
Pdelay_Resp_Follow_Up	$7F_{16}$

13.4 Suffix

An application layer message is suffixed by a contiguous sequence of zero or more TLV entities. The meaning of the entities is described in Clause 14. The first octet of TLV entity ‘n+1’ shall immediately follow the final octet of TLV entity ‘n’.

NOTE 1—The interpretation of a TLV should not depend on its position in the message.

NOTE 2—Nodes should append no TLV entity to event messages.

NOTE 3—Appending TLV entities to an event message is likely to change the transmission delay suffered by the messages in passing through non-PTP bridges.

13.5 Announce message

13.5.1 General Announce message specifications

The fields of Announce messages shall be as specified in Table 25.

Table 25: Announce message fields

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header (13.3)								34	0
originTimestamp								10	34
currentUTCOffset								2	44
reserved								1	46
grandmasterPriority1								1	47
grandmasterClockQuality								4	48
grandmasterPriority2								1	52
grandmasterIdentity								8	53
stepsRemoved								2	61
timeSource								1	63

13.5.2 Announce message field specifications

13.5.2.1 originTimestamp (Timestamp)

The value of originTimestamp shall be 0 or an estimate no worse than ± 1 second of the local time of the originating clock when the Announce message was transmitted.

13.5.2.2 currentUTCOffset (Integer16)

The value of currentUTCOffset shall be the value of the currentUtcOffset member of the timePropertiesDS data set.

13.5.2.3 grandmasterPriority1 (UInteger8)

The value of grandmasterPriority1 shall be the value of the grandmasterPriority1 member of the parentDS data set.

13.5.2.4 grandmasterClockQuality (ClockQuality)

The value of grandmasterClockQuality shall be the value of the grandmasterClockQuality member of the parentDS data set.

13.5.2.5 grandmasterPriority2 (UInteger8)

The value of grandmasterPriority2 shall be the value of the grandmasterPriority2 member of the parentDS data set.

13.5.2.6 grandmasterIdentity (ClockIdentity)

The value of grandmasterIdentity shall be the value of the grandmasterIdentity member of the parentDS data set.

13.5.2.7 stepsRemoved (UInteger16)

The value of stepsRemoved shall be the value of stepsRemoved of the currentDS data set of the clock issuing this message.

13.5.2.8 timeSource (Enumeration8)

The value of timeSource shall be the value of the timeSource member of the timePropertiesDS data set.

13.6 Sync and Delay_Req messages

13.6.1 General Sync and Delay_Req message specifications

The fields of Sync and Delay_Req messages shall be as specified in Table 26.

Table 26: Sync and Delay_Req message fields

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header (13.3)								34	0
originTimestamp								10	34

13.6.2 Sync and Delay_Req message field specifications

13.6.2.1 originTimestamp (Timestamp)

The value of the originTimestamp field shall be as specified in 9.5.9 and 11.3.

13.7 Follow_Up message

13.7.1 General Follow_Up message specifications

The fields of the Follow_Up message shall be as specified in Table 27.

Table 27: Follow_Up message fields

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header (13.3)								34	0
preciseOriginTimestamp								10	34

13.7.2 Follow_Up message field specifications

13.7.2.1 preciseOriginTimestamp (Timestamp)

The value of the preciseOriginTimestamp shall be as specified in 9.5.10 and 11.3.

13.8 Delay_Resp message

13.8.1 General Delay_Resp message specifications

The fields of the Delay_Resp message shall be as specified in Table 28.

Table 28: Delay_Resp message fields

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header (13.3)								34	0
receiveTimestamp								10	34
requestingPortIdentity								10	44

13.8.2 Delay_Resp message field specifications

13.8.2.1 receiveTimestamp (Timestamp)

The value of the receiveTimestamp shall be as specified in 9.5.12 and 11.3.

13.8.2.2 requestingPortIdentity (PortIdentity)

The value of the requestingPortIdentity shall be as specified in 9.5.12 and 11.3.

13.9 Pdelay_Req message

13.9.1 General Pdelay_Req message specifications

The fields of the Pdelay_Req message shall be as specified in Table 29.

Table 29: Pdelay_Req message fields

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header (13.3)								34	0
originTimestamp								10	34
reserved								10	44

Note- The reserved field in the Pdelay_Req message is to make the message length match the length of the Pdelay_Resp message. In some networks and bridges messages with unequal lengths have different transit times which introduce asymmetry errors

13.9.2 Pdelay_Req message field specifications

13.9.2.1 originTimestamp (Timestamp)

The value of the originTimestamp shall be as specified in 11.4.3.

13.10 Pdelay_Resp message

13.10.1 General Pdelay_Resp message specifications

The fields of the Pdelay_Resp message shall be as specified in Table 30.

Table 30: Pdelay_Resp message fields

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header (13.3)								34	0
requestReceiptTimestamp								10	34
requestingPortIdentity								10	44

13.10.2 Pdelay_Resp message field specifications

13.10.2.1 requestReceiptTimestamp (Timestamp)

The value of the requestReceiptTimestamp shall be as specified in 11.4.3.

13.10.2.2 requestingPortIdentity (PortIdentity)

The value of the requestingPortIdentity shall be as specified in 11.4.3.

13.11 Pdelay_Resp_Follow_Up message

13.11.1 General Pdelay_Resp_Follow_Up message specifications

The fields of the Pdelay_Resp_Follow_Up message shall be as specified in Table 31.

Table 31: Pdelay_Resp_Follow_Up message fields

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header (13.3)								34	0
responseOriginTimestamp								10	34
requestingPortIdentity								10	44

13.11.2 Pdelay_Resp_Follow_Up message field specifications

13.11.2.1 responseOriginTimestamp (Timestamp)

The value of the responseOriginTimestamp shall be as specified in 11.4.3.

13.11.2.2 requestingPortIdentity (PortIdentity)

The value of the requestingPortIdentity shall be as specified in 11.4.3.

13.12 Signaling message

13.12.1 Receipt of a signaling message from another node

Based on the indicated fields of a received signaling message, the message shall be accepted as indicated in Table 32.

Table 32 Acceptance of signalling messages

targetPortIdentity.clockIdentity equal to	targetPortIdentity.portNumber equal to	Accept Action
defaultDS.clockIdentity	All 1's	Yes
defaultDS.clockIdentity	portDS.portNumber	Yes but only apply to port with portNumber portDS.portNumber
All 1's	All 1's	Yes, but only if the TLV is specified to apply to all ports of the entire clock
All 1's	portDS.portNumber	Yes

TLVs defined as acting on a node shall be applied to the node if the signaling message is accepted. Messages that are not accepted shall be ignored.

13.12.2 Transmission of a signaling message

A port shall issue a signaling message when:

- Required by a TLV on a received signaling message, or
- Required by an optional or mandatory feature of this standard, or
- Required by implementation-specific considerations outside the scope of this standard.

The signaling message is used to transport a sequence of one or more TLV entities. Signaling messages are transmitted from one clock to one or more other clocks.

The common fields of a signaling message shall be as specified in Table 33.

Table 33: Signaling message fields

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header (13.3)								34	0
targetPortIdentity								10	34
One or more TLVs								M	44

13.12.2.1 targetPortIdentity (PortIdentity)

The targetPortIdentity field shall be of type PortIdentity. The value of the targetPortIdentity shall be the portIdentity, see 7.5.2, of the port to which this message is addressed.

NOTE—See 13.12.1.

13.13 Management message

Management messages are defined in Clause 15.

14. TLV entity specifications

14.1 General requirements

All TLV extensions shall have the data type, TLV, see 5.3.8.

PTP nodes that cannot parse a TLV extension shall ignore it and shall attempt to parse the next TLV in the message.

In the tables in Clause 14, the 'octets' column indicates the size of the field in octets. The 'TLV offset' column indicates the offset of the first octet of the field from the start of the TLV.

14.1.1 tlvType (Enumeration16)

The tlvType shall identify the TLV.

The values shall be as specified in Table 34.

Table 34: tlvType values

tlvType values	**Value$_{16}$**	**Defined in clause**
Reserved	0000	—
Standard TLVs		
MANAGEMENT	0001	15.5.3
MANAGEMENT_ERROR_STATUS	0002	15.5.4
ORGANIZATION_EXTENSION	0003	14.3
Optional unicast message negotiation TLVs		16.1
REQUEST_UNICAST_TRANSMISSION	0004	—
GRANT_UNICAST_TRANSMISSION	0005	—
CANCEL_UNICAST_TRANSMISSION	0006	—
ACKNOWLEDGE_CANCEL_UNICAST_TRANSMISSION	0007	—
Optional path trace mechanism TLV		**16.2**
PATH_TRACE	0008	—
Optional alternate timescale TLV		16.3
ALTERNATE_TIME_OFFSET_INDICATOR	0009	—
Reserved for standard TLVs	000A – 1FFF	—
Experimental TLVs		14.2
Security TLVs		Annex K
AUTHENTICATION	2000	—
AUTHENTICATION_CHALLENGE	2001	—
SECURITY_ASSOCIATION_UPDATE	2002	—
Cumulative frequency scale_factor offset		Annex L
CUM_FREQ_SCALE_FACTOR_OFFSET	2003	—
Reserved for Experimental TLVs	2004 – 3FFF	—
Reserved	4000 – FFFF	—

Experimental TLV values shall be reserved for assignment by the Precise Networked Clock Working Group of the IM/ST Committee, see 14.2.

14.1.2 lengthField (UInteger16)

The value is the length of the value field of the TLV in octets, see 5.3.8.

14.1.3 value (tlvType specific)

The format and meaning of the value member of the TLV is defined in subsequent clauses for each tlvType defined in this standard.

14.2 Experimental TLVs

Experimental TLVs are intended to facilitate operational experience with extensions that are likely to evolve into future standard extensions. Organizations or companies may apply to the Precise Networked Clock Working Group of the IM/ST Committee for an experimental tlvType value. Experimental tlvType values, the proposed format and semantics of the TLV, and contact information for the party responsible for the TLV will be public information.

Experimental TLV values are not permanent. They may be reassigned:

— If the TLV is made a standard TLV,

— If it is clear to all parties that it is no longer needed, or

— After a period of 5 years from the date of assignment.

14.3 Vendor and standard organization Extension TLVs

14.3.1 General

Vendor and standard organization extension TLVs can be used by vendors and standard organizations respectively to extend the protocol for their specific needs.

14.3.2 TLV member specifications

All organization specific TLV extensions shall have the format specified in Table 35.

Table 35: Organization specific TLV fields

Bits								Octets	TLV Offset
7	6	5	4	3	2	1	0		
tlvType								2	0
lengthField								2	2
organizationID								3	4
organizationSubType								3	7
data								N	10

14.3.2.1 tlvType (Enumeration16)

The tlvType shall be ORGANIZATION_EXTENSION for extensions defined by vendors and standards organization.

14.3.2.2 lengthField (UInteger16)

The value of the lengthField is 5+N where N is an odd number, see 5.3.8.

14.3.2.3 organizationID (Octet[3])

The value shall be the value of the OUI assigned to the vendor or standards organization by the IEEE [M5]. The octets shall be assigned in order to the 3-octet array with the most significant octet of the OUI assigned to the octet array member with index 0. The organization identified by the OUI shall ensure that organizationalSubType fields, 14.3.2.4, are unique within the scope defined by the organizationId value.

PTP nodes that do not recognize a particular organizationID or organizationSubType shall disregard the contents of the TLV except for the lengthField field.

14.3.2.4 organizationSubType (Enumeration24)

The organizationSubType field defines a sub-type of the tlvType. The organizationSubType values are assigned by the vendor or standards organization identified by the organizationID.

14.3.2.5 data (organizationSubType specific)

The format and meaning of the data field shall be defined by the owner of the concatenated {organizationID,organizationSubType} value.

15. Management

15.1 General

Management messages are used to access attributes and to generate certain events defined in this standard.

15.1.1 Selection of management mechanisms

One of the following two PTP management mechanisms shall be used:

- By default, the mechanism specified in 15.2, or
- An alternate management mechanism providing equivalent funtionality as specified in a PTP profile, or
- No specified management mechanism. The PTP profile shall specify the defined fixed values or/and state that there is an implementation-specific means to address all configurable variables, see 8.1.2.1.3.

15.2 PTP management mechanism

This management mechanism is defined by the remainder of Clause 15 and any management TLVs defined in optional clauses of this standard.

15.3 Processing of management messages

15.3.1 Receipt of a management message from another node

Based on the indicated fields of a received management message, the message shall be accepted as indicated in Table 36.

Table 36 Acceptance of management messages

targetPortIdentity. clockIdentity equal to	**targetPortIdentity. portNumber equal to**	**Action field equal to**	**managementID identified in Table 40 as applying only to entire clock**	**Accept Action**
defaultDS.clockIdentity	All 1’s	SET or COMMAND	o	Yes
defaultDS.clockIdentity	All 1’s	SET or COMMAND	Yes	Yes
defaultDS.clockIdentity	All 1’s	GET	Yes	Yes
defaultDS.clockIdentity	All 1’s	GET	No	No, except as specified in 15.5.3.1.2
defaultDS.clockIdentity	portDS.portNumber	GET or SET or COMMAND or RESPONSE or ACKNOWLEDGE	No	Yes
defaultDS.clockIdentity	portDS.portNumber	GET or SET or COMMAND	Yes	No
defaultDS.clockIdentity	portDS.portNumber	RESPONSE or ACKNOWLEDGE	Yes	Yes
All 1’s	All 1’s	SET or COMMAND	No	Yes
All 1’s	All 1’s	SET or COMMAND	Yes	Yes
All 1’s	All 1’s	GET	Yes	No
All 1’s	All 1’s	GET	No	No, except as specified in 15.5.3.1.2
All 1’s	portDS.portNumber	GET or SET or COMMAND	No	Yes
All 1’s	portDS.portNumber	GET or SET or COMMAND	Yes	No
All 1’s	portDS.portNumber	RESPONSE or ACKNOWLEDGE	No	No
All 1’s	portDS.portNumber	RESPONSE or ACKNOWLEDGE	Yes	No

The processing of an accepted management message shall be as specified for the TLV corresponding to the managementID field and as specified in the actionField, see 15.4.1.6. Management messages not accepted shall be ignored.

NOTE—The CLOCK_DESCRIPTION TLV can be used to discover any clock in the system supporting PTP management messages.

15.3.2 Transmission of a management message

The management message is used to transport a single management TLV entity. Management messages are used to transmit information from a clock to a node manager and from a node manager to one or more clocks, see 15.3.1.

A management message, identified in Table 40 as applying to the entire clock, shall be transmitted with the portNumber of the targetPortIdentity field set to all 1's.

15.3.3 Boundary clock forwarding of management messages

A boundary clock shall forward multicast management messages received on one port via other ports according to the following rules based on the state of the ports and the value of the boundaryHops field of the received management message:

a) Only multicast management messages received on a port in the MASTER, SLAVE, UNCALIBRATED, or PRE_MASTER states shall be forwarded.

b) If the received boundaryHops field value is 0, the management message shall not be retransmitted. Otherwise, the boundary clock shall decrement the value of the boundaryHops field of the management message by 1 before retransmitting the message

c) If the received boundaryHops field value is greater than 0, the management message shall be retransmitted only via ports in the MASTER, SLAVE, UNCALIBRATED, or PRE_MASTER states.

15.4 Management message format

15.4.1 Common fields

The common fields of a management message shall be as specified in Table 37.

Table 37: Management message fields

Bits								Octets	Offset
7	6	5	4	3	2	1	0		
header (see 13.3)								34	0
targetPortIdentity								10	34
startingBoundaryHops								1	44
boundaryHops								1	45
reserved				actionField				1	46
reserved								1	47
managementTLV								M	48

15.4.1.1 domainNumber of the header

The domainNumber of the message common header, see 13.3, of a management message shall be used for all TLV messages to specify a specific target domain.

15.4.1.2 sequenceId of the header

The sequenceId of the message common header, see 13.3, of a response management message shall be set to the sequenceId of the received management message causing the response. Otherwise the sequenceId shall be as specified in 7.3.7.

15.4.1.3 targetPortIdentity (PortIdentity)

The targetPortIdentity field shall be the identity of the port or node on which the management message acts.

NOTE—The port identified by targetPortIdentity is not necessarily the port on which the management message was received.

In the case of a management message transmitted by a clock to a manager, the targetPortIdentity field shall be set to the sourcePortIdentity of the management message to which it is a response.

15.4.1.4 startingBoundaryHops (UInteger8)

The value of the startingBoundaryHops field is implementation-dependent for messages that are not issued in response to a request from another management message. For management messages that are issued in response to a request from another management message, the value of startingBoundaryHops shall be the value computed from the startingBoundaryHops and boundaryHops fields of the requesting message as (startingBoundaryHops minus boundaryHops).

NOTE—when a management message is received, the absolute value of this difference indicates the number of retransmissions by boundary clocks that the message experienced.

15.4.1.5 boundaryHops (UInteger8)

The value of the boundaryHops field indicates the remaining number of successive retransmissions of the management message by boundary clocks receiving the message per 15.3.1. The value of boundaryHops shall be identical to the value of the field startingBoundaryHops when first transmitted by the issuing clock.

15.4.1.6 actionField (Enumeration4)

The value of the actionField shall indicate the action to be taken on receipt of the message as defined in Table 38.

Table 38: Values of the actionField

Action	**Action taken**	**Value$_{16}$**
GET	The management message shall carry a single management TLV. The managementID field of the TLV indicates the specific information that needs to be retrieved. The current values of the data identified by the managementID shall be returned in a management TLV with the actionField value set to RESPONSE. If an error occurs, a management error status TLV shall be returned with the actionField value set to RESPONSE.	0
SET	The management message shall carry a single management TLV. The data in the TLV shall be used to update the current value of the data identified by the managementID field. Attempts to set a static or non-configurable value shall return a management error status TLV, see 15.5.4. If the update is successful, a management message with the actionField value set to RESPONSE shall be returned. If an error occurs, a management error status TLV shall be returned with the actionField value set to RESPONSE. If the data identified by the managementID consists of several fields, the update shall be considered as an atomic actionField and the failure to update any item shall be considered an error in the execution of the SET. TLVs with data definitions that mix configurable and non-configurable data are not permitted.	1
RESPONSE	The data in the TLV shall be the current values of the data identified by the managementID field of the management message with the GET or SET actionField. The value of the managementID shall be identical to that in the requesting message. If the actionField required by the GET or SET actionFields could not be fully executed, the response shall be a management error status TLV, see 15.5.4.	2
COMMAND	The event indicated by the managementID field shall be initiated. The results of this command shall be acknowledged by a management message with actionField set to ACKNOWLEDGE.	3
ACKNOWLEDGE	An acknowledge management message is a response to a command management message. The value of the managementID shall be identical to that in the command message. If the command could not be executed the acknowledge message shall be a management error status TLV.	4
Reserved		5-F

15.4.1.7 managementTLV

Management messages shall be suffixed with zero or one TLV.

15.5 Management TLVs

15.5.1 Management TLV introduction

15.5.1.1 General

Subclause 15.5 details the structure of the management TLV and the management error status TLV.

There are two forms of management TLVs: those that manipulate data sets or individual data set members and those that initiate events.

15.5.1.1.1 Management of data sets

PTP defined configurable attributes, whether maintained in the data sets of Clause 8 or in implementation-specific form, are read and updated by management messages with the actionField value GET and SET respectively. The TLV data structures for these messages contain only configurable variables.

PTP defined static, dynamic, and configurable attributes maintained in data sets of Clause 8 or in implementation-specific form are read by management messages with the actionField value GET. SET may not be used with these messages.

15.5.1.1.2 Management of events

For TLVs that initiate events, the actionField value of the management message is COMMAND. The event and initiation semantics are defined or referenced for each TLV managementID.

15.5.2 Management TLV field format

Management TLV shall have the format specified in Table 39.

Table 39: Management TLV fields

Bits								Octets	TLV Offset
7	6	5	4	3	2	1	0		
tlvType								2	0
lengthField								2	2
managementID								2	4
dataField								N	6

15.5.2.1 tlvType (Enumeration16)

The tlvType shall be MANAGEMENT.

15.5.2.2 lengthField (UInteger16)

The value of the lengthField is 2+N where N is an even number, see 5.3.8.

15.5.2.3 managementID (Enumeration16)

The values of the managementID field are defined in Table 40. TLV semantics for each managementID value shall be as defined in the following subclauses and in optional clauses of this standard.

The entries in the allowed actions column of Table 40 indicate the permissible values of the actionField field in the management message common fields, see 15.4.1.6. The receipt of a management with a disallowed actionField value shall:

— Cause the contents of the management TLV to be disregarded

— Return a management error status TLV, see 15.5.4, NOT_SUPPORTED.

Table 40: managementID values

managementID name	**managementID value** (hex)	**Allowed actions**	**Applies only to entire**

			clock
Applicable to all node types	0000 – 1FFF		
NULL_MANAGEMENT	0000	GET, SET, COMMAND	—
CLOCK_DESCRIPTION	0001	GET,SET	—
USER_DESCRIPTION	0002	GET, SET	yes
SAVE_IN_NON_VOLATILE_STORAGE	0003	COMMAND	yes
RESET_NON_VOLATILE_STORAGE	0004	COMMAND	yes
INITIALIZE	0005	COMMAND	yes
FAULT_LOG	0006	GET	yes
FAULT_LOG_RESET	0007	COMMAND	yes
Reserved	0008 – 1FFF	—	—
Applicable to ordinary and boundary clocks	2000 – 3FFF	—	—
DEFAULT_DATA_SET	2000	GET	yes
CURRENT_DATA_SET	2001	GET	yes
PARENT_DATA_SET	2002	GET	yes
TIME_PROPERTIES_DATA_SET	2003	GET	yes
PORT_DATA_SET	2004	GET	
PRIORITY1	2005	GET, SET	yes
PRIORITY2	2006	GET, SET	yes
DOMAIN	2007	GET, SET	yes
SLAVE_ONLY	2008	GET, SET	yes
LOG_MEAN_ANNOUNCE_INTERVAL	2009	GET, SET	—
ANNOUNCE_RECEIPT_TIMEOUT	200A	GET, SET	—
LOG_MEAN_SYNC_INTERVAL	200B	GET, SET	—
VERSION_NUMBER	200C	GET, SET	—
ENABLE_PORT	200D	COMMAND	—
DISABLE_PORT	200E	COMMAND	—
TIME	200F	GET, SET	yes
CLOCK_ACCURACY	2010	GET, SET	yes
UTC_PROPERTIES	2011	GET, SET	yes
TRACEABILITY_PROPERTIES	2012	GET, SET	yes
TIMESCALE_PROPERTIES	2013	GET, SET	yes
UNICAST_NEGOTIATION_ENABLE	2014	GET, SET	—
PATH_TRACE_LIST	2015	GET	yes
PATH_TRACE_ENABLE	2016	GET, SET	yes
MASTER_CLUSTER_TABLE	2017	GET, SET	yes
UNICAST_MASTER_TABLE	2018	GET, SET	—
UNICAST_MASTER_MAX_TABLE_SIZE	2019	GET	—
ACCEPTABLE_MASTER_TABLE	201A	GET, SET	yes
ACCEPTABLE_MASTER_TABLE_ENABLED	201B	GET, SET	—
ACCEPTABLE_MASTER_MAX_TABLE_SIZE	201C	GET	yes
ALTERNATE_MASTER	201D	GET, SET	—
ALTERNATE_TIME_OFFSET_ENABLE	201E	GET, SET	yes
ALTERNATE_TIME_OFFSET_NAME	201F	GET, SET	yes
ALTERNATE_TIME_OFFSET_MAX_KEY	2020	GET	yes
ALTERNATE_TIME_OFFSET_PROPERTIES	2021	GET, SET	yes
Reserved	2022 – 3FFF	—	—
Applicable to transparent clocks	4000 to 5FFF	—	—
TRANSPARENT_CLOCK_DEFAULT_DATA_SET	4000	GET	yes
TRANSPARENT_CLOCK_CURRENT_DATA_SET	4001	GET	yes
TRANSPARENT_CLOCK_PORT_DATA_SET	4002	GET	
PRIMARY_DOMAIN	4003	GET, SET	yes
Reserved	4004 – 5FFF		

Applicable to ordinary, boundary and transparent clocks	6000 – 7FFF		
DELAY_MECHANISM	6000	GET, SET	
LOG_MIN_MEAN_PDELAY_REQ_INTERVAL	6001	GET, SET	
Reserved	6002 – BFFF		
This range is to be used for implementation-specific identifiers.	C000 – DFFF		
This range is to be assigned by an alternate PTP profile	E000 – FFFE		
Reserved	FFFF		

NOTE—The implementation-specific range of managementIds are assigned by manufacturers to define management functions unique to their own devices. There is no expectation of interoperability and users must ensure that such TLVs are directed to the appropriate device.

15.5.3 Management TLV data field specifications for each managementID

15.5.3.1 TLV data fields applicable to all clocks

15.5.3.1.1 NULL_MANAGEMENT

The management TLV data field is of zero length. No action affecting data sets or state shall result from receiving this TLV. The receipt of a NULL_MANAGEMENT message shall adhere to the requirements of the actionField, see 15.4.1.6.

NOTE—Null management messages are typically used to test implementations by exercising the management handlers without producing any change in protocol operation. For example such a message can be sent to test whether received management messages are being recorded in an implementation-specific event log.

15.5.3.1.2 CLOCK_DESCRIPTION

The portNumber member of the targetPortIdentity field, see 15.4.1.3, of the management message carrying this TLV shall indicate the port to which the physicalAddress, protocolAddress, and profileIdentity apply. If the portNumber is all 1's, then:

— The fields returned in this message shall be those for port 1 of the node regardless of the number of PTP ports on the node, and

— No information shall be returned for any of the other ports on a multiple PTP port node.

NOTE—The requirement that a portNumber of all 1's result in returning only the description for port 1 of the node is not in conflict with 15.3.2, which asserts that a portNumber of all 1's indicates that the received TLV applies to all ports on a node.

All other fields of this TLV apply to the entire node and shall be returned by a query directed at any port on the node.

The data field shall be as specified in Table 41.

Table 41: CLOCK_DESCRIPTION management TLV data field

Bits								**Octets**	**TLV data offset**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
clockType								2	0
physicalLayerProtocol								L	2

physicalAddressLength	2	2+L
physicalAddress	M	4+L
protocolAddress	N	4+L+M
manufacturerIdentity	3	4+L+M+N
Reserved	1	4+L+M+N+3
productDescription	P	4+L+M+N+4
revisionData	Q	4+L+M+N+4+P
userDescription	R	4+L+M+N+4+P+Q
profileIdentity	6	4+L+M+N+4+P+Q+R
pad	0 or 1	4+L+M+N+4+P+Q+R+6

15.5.3.1.2.1 clockType (Boolean[16])

The value shall indicate the type of PTP node as defined in Table 42. A TRUE value indicates that the description applies to the node.

NOTE—Several elements may be TRUE for example an ordinary clock combined with an end-to-end transparent clock. This is a static value not part of the clock data sets of Clause 8.

Table 42: clockType specification

Array index	Description
0	The node implements an ordinary clock
1	The node implements a boundary clock
2	The node implements a peer-to-peer transparent clock
3	The node implements an end-to-end transparent clock
4	The node implements a management node
5 – 15	Reserved

15.5.3.1.2.2 physicalLayerProtocol (PTPText)

The value shall indicate the physical layer protocol defining the physicalAddress member. This is a static value not part of the clock data sets of Clause 8.

The maximum number of characters in this field shall be 32.

15.5.3.1.2.3 physicalAddressLength (UInteger16)

The value is the number of octets in the physicalAddress field. The range shall be 1 to 16 octets. This is a static value not part of the clock data sets of Clause 8.

15.5.3.1.2.4 physicalAddress (Octet[physicalAddressLength])

The value shall be the physical address of the port indicated by the portNumber member of the targetPortIdentity field, for example for the MAC address for an IEEE 802.3 end station. If no physical address exists for a specific network technology, the value shall be of zero length.

This is a static value not part of the clock data sets of Clause 8.

15.5.3.1.2.5 protocolAddress (PortAddress)

The value shall be the protocol address of the port indicated by the portNumber member of the targetPortIdentity field. This is a static value (it can not be modified using the protocol) and is not part of the clock data sets of Clause 8.

15.5.3.1.2.6 manufacturerIdentity (Octet[4])

The 3 most significant octets of the manufacturerIdentity shall be an OUI owned by the manufacturer of the node. The remaining octet shall be all zeros.
For example:
The OUI for Company X is ACDE48 (hex). The byte and bit representations of the manufacturerIdentity of Company X are illustrated below.

OUI			
addr+0	addr+1	addr+2	order
AC	DE	48	hex
10101100	11011110	01001000	bits

```
|   |   |                                      |   |
|   |   group address bit                      |   |
|   most significant byte      least significant byte   |
```

This is a static value not part of the clock data sets of Clause 8**.**

15.5.3.1.2.7 productDescription (PTPText)

The productDescription field shall indicate, in order:
The name of the manufacturer of the node, manufacturerName, followed by a semicolon (;)
The model number of the node, modelNumber, followed by a semicolon(;)
An identifier of the instance of this mode, instanceIdentifierl, such as the MAC address or the serial number
The maximum number of characters in this field shall be 64.
This is a static value not part of the clock data sets of Clause 8**.**

The content and meaning of the manufacturerName, modelNumber, and the instanceIdentifier (the instance of this model, for example the serial number) strings are determined by the manufacturer.

15.5.3.1.2.8 revisionData (PTPText)

The value shall indicate the revisions for node hardware (HW), firmware (FW), and software (SW). This information shall be semicolon (;) separated text fields in the order HW;FW;SW. Non-applicable elements shall be indicated by a text fields of zero length. This is a static value not part of the clock data sets of Clause 8.

The maximum number of characters in this field shall be 32.

15.5.3.1.2.9 userDescription (PTPText)

The userDescription field shall indicate, in order:

a) A user defined name of the device, e.g. Sensor-1, followed by a semicolon (;)

b) A user defined physical location of the device, e.g. Rack-2 Shelf-3.

Either field may be absent, e.g. (;Rack-2 Shelf-3) or (Sensor-1). By default no text is required. This is a configurable value not part of the clock data sets of Clause 8.

The maximum number of characters in this field shall be 128.

15.5.3.1.2.10 profileIdentity (Octet[6])

The value of profileIdentity shall identify the PTP profile implemented by the port indicated by the portNumber member of the targetPortIdentity field.
The value of the profileIdentity shall be assigned by the creator of the profile as defined in subclause 19.3.3.
This is a static value not part of the clock data sets of Clause 8.

For example:
Annex J.4.1 defines the profileIdentity of the 'Default PTP profile for use with the peer delay mechanism' as the 6 octet field 001B19000200$_{16}$. The bit and byte representation of this profileIdentity are shown below:

OUI			**profileIndex**			**field**
addr+0	**addr+1**	**addr+2**	**addr+3**	**addr+4**	**addr+5**	**order**
00	1B	19	00	00	00	**hex**
00000000	00011011	00011001	00000000	00000000	00000000	**bits**

```
|   |   |                                                     |   |
|   |   group address bit                                     |   |
|   most significant byte                  least significant byte |
```

15.5.3.1.2.11 pad (Octet[M])

The pad field either shall be an octet array of length M where M is 1 with all bits 0 or shall be of length 0, whichever is required to make the size of the data field an even number of octets, see 15.5.2.2.

15.5.3.1.3 USER_DESCRIPTION

The data field shall be as specified in Table 43.

Table 43: USER_DESCRIPTION management TLV data field

Bits								**Octets**	**TLV data offset**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
userDescription								L	0
pad								0 or 1	L

15.5.3.1.3.1 userDescription (PTPText)

This TLV is used to configure the value of the user description returned by the CLOCK_DESCRIPTION TLV.

The userDescription field shall indicate, in order:

a) A user defined name of the device, e.g. Sensor-1, followed by a semicolon (;)

b) A user defined physical location of the device, e.g. Rack-2 Shelf-3.

Either field may be absent, e.g. (;Rack-2 Shelf-3) or (Sensor-1). No text is required.

The maximum number of characters in this field shall be 128.

15.5.3.1.3.2 pad (Octet[M])

The pad field either shall be an octet array of length M where M is 1 with all bits 0 or shall be of length 0, whichever is required to make the size of the data field an even number of octets, see 15.5.2.2.

15.5.3.1.4 SAVE_IN_NON_VOLATILE_STORAGE

The data field is of zero length. The receipt of this TLV shall cause the current values of the applicable dynamic and configurable data set members to be copied into non-volatile read-write memory as specified in 8.1.3.5.

15.5.3.1.5 RESET_NON_VOLATILE_STORAGE

The data field is of zero length. The receipt of this TLV shall cause the contents of non-volatile read-write memory to be reset to the applicable dynamic or configurable data set initialization values as specified in 8.1.3.5.

15.5.3.1.6 INITIALIZE

The data field shall be as specified in Table 44.

Table 44: INITIALIZE management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
initializationKey								2	0

15.5.3.1.6.1 initializationKey (Enumeration16)

The value of the initializationKey field of this message shall be as defined in Table 45.

Table 45: INITIALIZATION_KEY enumeration

INITIALIZATION_KEY	Value$_{16}$	Definition
INITIALIZE_EVENT	0000	In an ordinary and boundary clock the receipt of an INITIALIZE message with this key shall cause the INITIALIZE event, see 9.2.6.3, to occur. In other nodes this shall cause any implementation-specific initialization procedures to execute.

Reserved	0001-7FFF	Reserved. No action shall occur as a result of receiving an INITIALIZE message with this initializationKey.
Implementation-specific	8000 – FFFF	The result is implementation-specific.

15.5.3.1.7 FAULT_LOG

The data field shall be as specified in Table 47. The FAULT_LOG TLV returns a list of fault records. Each fault record is specified by the FaultRecord structure. The faultName, faultValue, and faultDescription members are implementation-specific and may be zero length, i.e. the length member of the PTPText struct is 0. The value of the severity member shall be selected from the enumeration in Table 46:

Table 46 Fault log severity enumeration

Value	Severity description
0	Emergency: system is unusable
1	Alert: immediate action needed
2	Critical: critical conditions
3	Error: error conditions
4	Warning: warning conditions
5	Notice: normal but significant condition
6	Informational: informational messages
7	Debug: debug-level messages

The faultName shall be a name for the fault unique within the implementation.

The faultValue shall be any value that may be associated with the fault that is necessary for fault diagnosis.

The faultDescription shall be any supplementary description of the fault.

The size of the fault log is implementation-specific. The fault log is maintained by the clock until a fault reset command is issued.

Table 47: FAULT_LOG management TLV data field

Bits								Octets	TLV Data Offset
7	6	5	4	3	2	1	0		
numberOfFaultRecords								2	0
faultRecord								N	2
pad								0 or 1	2+N

15.5.3.1.7.1 numberOfFaultRecords (UInteger16)

The value of the numberOfFaultRecords field shall be the number of fault records to be returned.

15.5.3.1.7.2 faultRecord (FaultRecord[numberOfFaultRecords])

The value of the faultRecord field shall be an array of fault records.

15.5.3.1.7.3 pad (Octet[M])

The pad field either shall be an octet array of length M where M is 1 with all bits 0 or shall be of length 0, whichever is required to make the size of the data field an even number of octets, see 15.5.2.2.

15.5.3.1.8 FAULT_LOG_RESET

The TLV carries no data.

The FAULT_LOG_RESET command shall cause the FAULT_LOG to be cleared.

15.5.3.2 TLV data fields applicable to ordinary and boundary clocks

15.5.3.2.1 TIME

This TLV may be used to set the time. Time originates in the grandmaster clock and is distributed by PTP to other clocks in the domain.

The time in the grandmaster clock is normally determined by interacting with a primary time source, e.g. GPS, by means outside the scope of this standard. When this TLV is sent to a node other than the grandmaster with an actionField of SET, the node should return a management error status TLV. When the actionField is GET, the node should return the current value of time.

NOTE—If the time is set in a clock other than the grandmaster, it will be overwritten upon receipt of the next Sync message and will therefore exist only as a transient.

When sent to a settable grandmaster clock, the normal rules for GET and SET apply, see 15.4.1.6.

The data field shall be as specified in Table 48.

Table 48: TIME management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
currentTime								10	0

15.5.3.2.1.1 currentTime (Timestamp)

The time to be set in the target clock.

NOTE—Since this TLV is transmitted over the network, the accuracy to which the time can be set is limited by network fluctuations. For this reason, no provision is made for setting the time to a precision of fractions of a nanosecond. In most cases the actual precision is on the order of milliseconds or worse depending on the source of the information used in populating the data field and on the characteristics of the network.

15.5.3.2.2 CLOCK_ACCURACY

This TLV may be used to set the accuracy of the target clock.

NOTE— The accuracy and the time in the grandmaster clock is normally determined by interacting with a primary or application specific time source, e.g. GPS, by means outside the scope of this standard. If the time is set in the grandmaster by means of the TIME TLV, then the accuracy should also be set. Since the clockAccuracy attribute is considered in the operation of the BMC algorithm, the setting of the clockAccuracy attribute in any clock by means of this TLV can result in a change of grandmaster the next time the BMC algorithm is performed.

When sent to a settable grandmaster clock, the normal rules for GET and SET apply, see 15.4.1.6.

The data field shall be as specified in Table 49.

Table 49: CLOCK_ACCURACY management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
clockAccuracy								1	0
reserved								1	1

15.5.3.2.2.1 clockAccuracy (Enumeration8)

The value shall be the value of the clockAccuracy member of the clockQuality member of the defaultDS data set. The value shall be selected from the clockAccuracy enumeration, see Table 6.

15.5.3.2.3 ENABLE_PORT

The TLV carries no data.

In an ordinary and boundary clock the receipt of an ENABLE_PORT message shall cause the DESIGNATED_ENABLED event, see 9.2.6.4, to occur.

15.5.3.2.4 DISABLE_PORT

The TLV carries no data.

In an ordinary and boundary clock the receipt of an DISABLE_PORT message shall cause the DESIGNATED_DISABLED event, see 9.2.6.5, to occur.

15.5.3.3 TLV data fields applicable to the defaultDS data set of ordinary and boundary clocks

15.5.3.3.1 DEFAULT_DATA_SET

The data field shall be as specified in Table 50.

Table 50: DEFAULT_DATA_SET management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
0	0	0	0	0	0	SO	TSC	1	0
reserved								1	1
clockIdentity								8	2
numberPorts								2	10
clockQuality								4	12

priority1	1	16
priority2	1	17
domainNumber	1	18
reserved	1	19

15.5.3.3.1.1 TSC (Boolean)

The value of TSC shall be the value of the twoStepFlag member of the defaultDS data set.

15.5.3.3.1.2 slaveOnly (Boolean)

The value of slaveOnly shall be the value of the slaveOnly member of the defaultDS data set.

15.5.3.3.1.3 clockIdentity (ClockIdentity)

The value of clockIdentity shall be the value of the clockIdentity member of the defaultDS data set.

15.5.3.3.1.4 numberPorts (UInteger16)

The value of numberPorts shall be the value of the numberPorts member of the defaultDS data set.

15.5.3.3.1.5 clockQuality (ClockQuality)

The value of clockQuality shall be the value of the clockQuality member of the defaultDS data set.

15.5.3.3.1.6 priority1 (UInteger8)

The value of priority1 shall be the value of the priority1 member of the defaultDS data set.

15.5.3.3.1.7 priority2 (UInteger8)

The value of priority2 shall be the value of the priority2 member of the defaultDS data set.

15.5.3.3.1.8 domainNumber (UInteger8)

The value of domainNumber shall be the value of the domainNumber member of the defaultDS data set.

15.5.3.3.2 PRIORITY1

The data field shall be as specified in Table 51.

Table 51: PRIORITY1 management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
priority1								1	0
reserved								1	1

15.5.3.3.2.1 priority1 (UInteger8)

The value of priority1 shall be the value of the priority1 member of the defaultDS data set.

15.5.3.3.3 PRIORITY2

The data field shall be as specified in Table 52.

Table 52: PRIORITY2 management TLV data field

Bits								**Octets**	**TLV data offset**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
priority2								1	0
reserved								1	1

15.5.3.3.3.1 priority2 (UInteger8)

The value of priority2 shall be the value of the priority2 member of the defaultDS data set.

15.5.3.3.4 DOMAIN

The data field shall be as specified in Table 53.

Table 53: DOMAIN management TLV data field

Bits								**Octets**	**TLV data offset**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
domainNumber								1	0
reserved								1	1

15.5.3.3.4.1 domainNumber (UInteger8)

The value of domainNumber shall be the value of the domainNumber member of the defaultDS data set.

15.5.3.3.5 SLAVE_ONLY

The data field shall be as specified in Table 54.

Table 54: SLAVE_ONLY management TLV data field

Bits								**Octets**	**TLV data offset**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
0	0	0	0	0	0	0	SO	1	0
reserved								1	1

15.5.3.3.5.1 SO (Boolean)

The value of SO shall be the value of the slaveOnly member of the defaultDS data set.

15.5.3.4 TLV data fields applicable to the currentDS data set of ordinary and boundary clocks

15.5.3.4.1 CURRENT_DATA_SET

The data field shall be as specified in Table 55.

Table 55: CURRENT_DATA_SET management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
stepsRemoved								2	0
offsetFromMaster								8	2
meanPathDelay								8	10

15.5.3.4.1.1 stepsRemoved (UInteger16)

The value of stepsRemoved shall be the value of the stepsRemoved member of the currentDS data set.

15.5.3.4.1.2 offsetFromMaster (TimeInterval)

The value of offsetFromMaster shall be the value of the offsetFromMaster member of the currentDS data set.

15.5.3.4.1.3 meanPathDelay (TimeInterval)

The value of meanPathDelay shall be the value of the meanPathDelay member of the currentDS data set.

15.5.3.5 TLV data fields applicable to the parentDS data set of ordinary and boundary clocks

15.5.3.5.1 PARENT_DATA_SET

The data field shall be as specified in Table 56.

Table 56: PARENT_DATA_SET management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
parentPortIdentity								10	0
0	0	0	0	0	0	0	PS	1	10
reserved								1	11
observedParentOffsetScaledLogVariance								2	12
observedParentClockPhaseChangeRate								4	14
grandmasterIdentity								8	18
grandmasterClockQuality								4	26
grandmasterPriority1								1	30
grandmasterPriority2								1	31

15.5.3.5.1.1 parentPortIdentity (PortIdentity)

The value of parentPortIdentity shall be the value of the parentPortIdentity member of the parentDS data set.

15.5.3.5.1.2 PS (Boolean)

The value of PS shall be the value of the parentStats member of the parentDS data set.

15.5.3.5.1.3 observedParentOffsetScaledLogVariance (UInteger16)

The value of observedParentOffsetScaledLogVariance shall be the value of the observedParentOffsetScaledLogVariance member of the parentDS data set.

15.5.3.5.1.4 observedParentClockPhaseChangeRate (Integer32)

The value of observedParentClockPhaseChangeRate shall be the value of the observedParentClockPhaseChangeRate member of the parentDS data set.

15.5.3.5.1.5 grandmasterIdentity (ClockIdentity)

The value of grandmasterIdentity shall be the value of the grandmasterIdentity member of the parentDS data set.

15.5.3.5.1.6 grandmasterClockQuality (ClockQuality)

The value of grandmasterClockQuality shall be the value of the grandmasterClockQuality member of the parentDS data set.

15.5.3.5.1.7 grandmasterPriority1 (UInteger8)

The value of grandmasterPriority1 shall be the value of the grandmasterPriority1 member of the parentDS data set.

15.5.3.5.1.8 grandmasterPriority2 (UInteger8)

The value of grandmasterPriority2 shall be the value of the grandmasterPriority2 member of the parentDS data set.

15.5.3.6 TLV data fields applicable to the timePropertiesDS data set of ordinary and boundary clocks

15.5.3.6.1 TIME_PROPERTIES_DATA_SET

The data field shall be as specified in Table 57.

Table 57: TIME_PROPERTIES_DATA_SET management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
currentUtcOffset								2	0
0	0	FTRA	TTRA	PTP	UTCV	LI-59	LI-61	1	2
timeSource								1	3

15.5.3.6.1.1 currentUtcOffset (Integer16)

The value of currentUtcOffset shall be the value of the currentUtcOffset member of the timePropertiesDS data set.

15.5.3.6.1.2 LI-61 (Boolean)

The value of LI-61 shall be the value of the leap61 member of the timePropertiesDS data set.

15.5.3.6.1.3 LI-59 (Boolean)

The value of LI-59 shall be the value of the leap59 member of the timePropertiesDS data set.

15.5.3.6.1.4 UTCV (Boolean)

The value of UTCV shall be the value of the currentUtcOffsetValid member of the timePropertiesDS data set.

15.5.3.6.1.5 PTP (Boolean)

The value of PTP shall be the value of the ptpTimescale member of the timePropertiesDS data set.

15.5.3.6.1.6 TTRA (Boolean)

The value of TTRA shall be the value of the timeTraceable member of the timePropertiesDS data set.

15.5.3.6.1.7 FTRA (Boolean)

The value of FTRA shall be the value of the frequencyTraceable member of the timePropertiesDS data set.

15.5.3.6.1.8 timeSource (Enumeration8)

The value of timeSource shall be the value of the timeSource member of the timePropertiesDS data set.

15.5.3.6.2 UTC_PROPERTIES

The data field shall be as specified in Table 58.

Table 58: UTC_PROPERTIES management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
currentUtcOffset								2	0
0	0	0	0	0	UTCV	LI-59	LI-61	1	2
reserved								1	3

15.5.3.6.2.1 currentUtcOffset (Integer16)

The value of currentUtcOffset shall be the value of the currentUtcOffset member of the timePropertiesDS data set.

15.5.3.6.2.2 LI-61 (Boolean)

The value of LI-61 shall be the value of the leap61 member of the timePropertiesDS data set.

15.5.3.6.2.3 LI-59 (Boolean)

The value of LI-59 shall be the value of the leap59 member of the timePropertiesDS data set.

15.5.3.6.2.4 UTCV (Boolean)

The value of UTCV shall be the value of the currentUtcOffsetValid member of the timePropertiesDS data set.

15.5.3.6.3 TRACEABILITY_PROPERTIES

The data field shall be as specified in Table 59

Table 59: TRACEABILITY_PROPERTIES management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
0	0	FTRA	TTRA	0	0	0	0	1	0
reserved								1	1

15.5.3.6.3.1 TTRA (Boolean)

The value of TTRA shall be the value of the timeTraceable member of the timePropertiesDS data set.

15.5.3.6.3.2 FTRA (Boolean)

The value of FTRA shall be the value of the frequencyTraceable member of the timePropertiesDS data set.

15.5.3.6.4 TIMESCALE_PROPERTIES

The data field shall be as specified in Table 60.

Table 60: TIMESCALE_PROPERTIES management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
0	0	0	0	PTP	0	0	0	1	0
timeSource								1	1

15.5.3.6.4.1 PTP (Boolean)

The value of PTP shall be the value of the ptpTimescale member of the timePropertiesDS data set.

15.5.3.6.4.2 timeSource (Enumeration8)

The value of timeSource shall be the value of the timeSource member of the timePropertiesDS data set.

15.5.3.7 TLV data fields applicable to the portDS data set of ordinary and boundary clocks

15.5.3.7.1 PORT_DATA_SET

The data field shall be as specified in Table 61.

Table 61: PORT_DATA_SET management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
portIdentity								10	0
portState								1	10
logMinDelayReqInterval								1	11
peerMeanPathDelay								8	12
logAnnounceInterval								1	20
announceReceiptTimeout								1	21
logSyncInterval								1	22
delayMechanism								1	23
logMinPdelayReqInterval								1	24
reserved				versionNumber				1	25

15.5.3.7.1.1 portIdentity (PortIdentity)

The value of portIdentity shall be the value of the portIdentity member of the portDS data set.

15.5.3.7.1.2 portState (Enumeration8)

The value of portState shall be the value of the portState member of the portDS data set.

15.5.3.7.1.3 logMinDelayReqInterval (Integer8)

The value of logMinDelayReqInterval shall be the value of the logMinDelayReqInterval member of the portDS data set.

15.5.3.7.1.4 peerMeanPathDelay (TimeInterval)

The value of peerMeanPathDelay shall be the value of the peerMeanPathDelay member of the portDS data set.

15.5.3.7.1.5 logAnnounceInterval (Integer8)

The value of logAnnounceInterval shall be the value of the logAnnounceInterval member of the portDS data set.

15.5.3.7.1.6 announceReceiptTimeout (UInteger8)

The value of announceReceiptTimeout shall be the value of the announceReceiptTimeout member of the portDS data set.

15.5.3.7.1.7 logSyncInterval (Integer8)

The value of logSyncInterval shall be the value of the logSyncInterval member of the portDS data set.

15.5.3.7.1.8 delayMechanism (Enumeration8)

The value of delayMechanism shall be the value of the value of the delayMechanism member of the portDS data set.

15.5.3.7.1.9 logMinPdelayReqInterval (Integer8)

The value of logMinPdelayReqInterval shall be the value of the value of the logMinPdelayReqInterval member of the portDS data set.

15.5.3.7.1.10 versionNumber (UInteger4)

The value of versionNumber shall be the value of the versionNumber member of the portDS data set.

15.5.3.7.2 LOG_MEAN_ANNOUNCE_INTERVAL

The data field shall be as specified in Table 62.

Table 62: LOG_MEAN_ANNOUNCE_INTERVAL management TLV data field

Bits								**Octets**	**TLV data offset**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
logAnnounceInterval								1	0
reserved								1	1

15.5.3.7.2.1 logAnnounceInterval (Integer8)

The value shall be the value of the logAnnounceInterval member of the portDS data set.

15.5.3.7.3 ANNOUNCE_RECEIPT_TIMEOUT

The data field shall be as specified in Table 63.

Table 63: ANNOUNCE_RECEIPT_TIMEOUT management TLV data field

Bits								**Octets**	**TLV data offset**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
announceReceiptTimeout								1	0
reserved								1	1

15.5.3.7.3.1 announceReceiptTimeout (UInteger8)

The value of announceReceiptTimeout shall be the value of the announceReceiptTimeout member of the portDS data set.

15.5.3.7.4 LOG_MEAN_SYNC_INTERVAL

The data field shall be as specified in Table 64.

Table 64: LOG_MEAN_SYNC_INTERVAL management TLV data field

Bits								**Octets**	**TLV data offset**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
logSyncInterval								1	0
reserved								1	1

15.5.3.7.4.1 logSyncInterval (Integer8)

The value of logSyncInterval shall be the value of the logSyncInterval member of the portDS data set.

15.5.3.7.5 DELAY_MECHANISM

The data field shall be as specified in Table 65.

Table 65: DELAY_MECHANISM management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
delayMechanism								1	0
reserved								1	1

15.5.3.7.5.1 delayMechanism (Enumeration8)

The value shall be the value of the delayMechanism member of the portDS data set for ordinary or boundary clocks. For transparent clocks, the value shall be the value of the delayMechanism member of the defaultDS data set if implemented; otherwise, the value shall be obtained from the implementation-specific storage of this value.

15.5.3.7.6 LOG_MIN_MEAN_PDELAY_REQ_INTERVAL

The data field shall be as specified in Table 66.

Table 66: LOG_MIN_MEAN_PDELAY_REQ_INTERVAL management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
logMinPdelayReqInterval								1	0
reserved								1	1

15.5.3.7.6.1 logMinPdelayReqInterval (Integer8)

The value of logMinPdelayReqInterval shall be the value of the logMinPdelayReqInterval member of the portDS data set.

15.5.3.7.7 VERSION_NUMBER

The data field shall be as specified in Table 67, see 7.5.5.

Table 67: VERSION_NUMBER management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
reserved				versionNumber				1	0
reserved								1	1

15.5.3.7.7.1 versionNumber (UInteger4)

The value of versionNumber shall be the value of the versionNumber member of the portDS data set.

15.5.3.8 TLV data fields applicable to the defaultDS data set of transparent clocks

15.5.3.8.1 TRANSPARENT_CLOCK_DEFAULT_DATA_SET

The data field shall be as specified in Table 68.

Table 68: TRANSPARENT_CLOCK_DEFAULT_DATA_SET management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
clockIdentity								8	0
numberPorts								2	8
delayMechanism								1	10
primaryDomain								1	11

15.5.3.8.1.1 clockIdentity (ClockIdentity)

The value of clockIdentity shall be the value of the clockIdentity member of the defaultDS data set.

15.5.3.8.1.2 numberPorts (UInteger16)

The value of numberPorts shall be the value of the numberPorts member of the defaultDS data set.

15.5.3.8.1.3 delayMechanism (Enumeration8)

The value of delayMechanism shall be the value of the delayMechanism member of the defaultDS data set.

15.5.3.8.1.4 primaryDomain (UInteger8)

The value of primaryDomain shall be the value of the primaryDomain member of the defaultDS data set.

15.5.3.9 DELAY_MECHANISM

The same TLV applicable to the portDS data set of ordinary or boundary clocks shall be used, see 15.5.3.7.5.

15.5.3.9.1 PRIMARY_DOMAIN

The data field shall be as specified in Table 69.

Table 69: PRIMARY_DOMAIN management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
primaryDomain								1	0
reserved								1	1

15.5.3.9.1.1 primaryDomain (UInteger8)

The value of primaryDomain shall be the value of the primaryDomain member of the defaultDS data set if implemented; otherwise, the value shall be obtained from the implementation-specific storage of this value.

15.5.3.10 TLV data fields applicable to the currentDS data set of transparent clocks

15.5.3.10.1 TRANSPARENT_CLOCK_CURRENT_DATA_SET

The data field shall be as specified in Table 70.

Table 70: TRANSPARENT_CLOCK_CURRENT_DATA_SET management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
0	0	0	0	0	0	DS	SY	1	0
scaledFractionalFreqencyOffset								1	1

15.5.3.10.1.1 SY (Boolean)

The value of SY shall be the value of the syntonizedFlag member of the currentDS data set.

15.5.3.10.1.2 DS (Boolean)

The value of DS shall be the value of the domainSyntonization member of the domain called out in the domain field in the common header.

15.5.3.10.1.3 scaledFractionalFreqencyOffset (Integer8)

The value of scaledFractionalFreqencyOffset shall be the value of the scaledFractionalFreqencyOffset member of the domain called out in the domain field in the common header.

15.5.3.11 TLV data fields applicable to the portDS data set of transparent clocks

15.5.3.11.1 TRANSPARENT_CLOCK_PORT_DATA_SET

The data field shall be as specified in Table 71.

Table 71: TRANSPARENT_CLOCK_PORT_DATA_SET management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
portIdentity								10	0
0	0	0	0	0	0	0	FLT	1	10
logMinPdelayReqInterval								1	11
peerMeanPathDelay								8	12

15.5.3.11.1.1 portIdentity (PortIdentity)

The value of portIdentity shall be the value of the portIdentity member of the portDS data set.

15.5.3.11.1.2 FLT (Boolean)

The value of FLT shall be the value of the faulty member of the portDS data set.

15.5.3.11.1.3 logMinPdelayReqInterval (Integer8)

The value of logMinPdelayReqInterval shall be the value of the logMinPdelayReqInterval member of the portDS data set.

15.5.3.11.1.4 peerMeanPathDelay (TimeInterval)

The value of peerMeanPathDelay shall be the value of the peerMeanPathDelay member of the portDS data set.

15.5.3.11.2 LOG_MIN_MEAN_PDELAY_REQ_INTERVAL

The same TLV applicable to the portDS data set of ordinary or boundary clocks shall be used, see 15.5.3.7.6.

15.5.4 MANAGEMENT_ERROR_STATUS TLV

15.5.4.1.1 General

This TLV is returned in either response or acknowledge management messages, see 15.4.1.6. The MANAGEMENT_ERROR_STATUS TLV format shall be as specified in Table 72.

Table 72: MANAGEMENT_ERROR_STATUS TLV format

Bits								Octets	TLV Offset
7	6	5	4	3	2	1	0		
tlvType								2	0
lengthField								2	2
managementErrorID								2	4
managementID								2	6
reserved								4	8
displayData								N	12

15.5.4.1.2 tlvType

The value of tlvType shall be MANAGEMENT_ERROR_STATUS.

15.5.4.1.3 lengthField

The lengthField shall be 8 + N where N is the length of the displayData field and shall be an even number of octets. N shall be no greater than 244 octets.

15.5.4.1.4 managementErrorID (Enumeration16)

The value of this enumeration shall be as defined in Table 73.

Table 73: managementErrorID enumeration

managementErrorID	Specification	value$_{16}$
Reserved	—	0000
RESPONSE_TOO_BIG	The requested operation could not fit in a single response message.	0001
NO_SUCH_ID	The managementID is not recognized	0002
WRONG_LENGTH	The managementID was identified but the length of the data was wrong	0003
WRONG_VALUE	The managementID and length were correct but one or more values were wrong.	0004
NOT_SETABLE	Some of the variables in the set command were not updated because they are not configurable.	0005
NOT_SUPPORTED	The requested operation is not supported in this node	0006

Reserved		0007 – BFFF
Implementation-specific	This range is to be used for implementation-specific errors.	C000 – DFFF
PTP profile defined	This range is to be assigned by an alternate PTP profile	E000 – FFFD
GENERAL_ERROR	An error occurred that is not covered by other managementErrorID values.	FFFE
Reserved	—	FFFF

15.5.4.1.5 managementID (Enumeration16)

The values of the managementID field are defined in Table 40. The managementID field shall contain the managementID corresponding to the managementID that was in error.

15.5.4.1.6 displayData (PTPText)

This is an optional text field to provide a human readable explanation of the error.

The maximum number of characters in this field shall be 50.

16. General optional features

16.1 Unicast message negotiation (optional)

16.1.1 General unicast negotiation port operation specifications

A port (the requestor) may request by transmitting a REQUEST_UNICAST_TRANSMISSION TLV entity, that another port (the grantor) transmit unicast Announce, Sync, Delay_Resp, or Pdelay_Resp messages.

A request for unicast transmissions may be made, granted, acknowledged, and cancelled irrespective of PTP port state, except that these operations shall not occur in any port of an ordinary or boundary clock in the INITIALIZING, FAULTY, or DISABLED states or in any port of a transparent clock that is in a fault condition.

A port that receives a REQUEST_UNICAST_TRANSMISSION TLV entity shall respond with a GRANT_UNICAST_TRANSMISSION TLV entity. The transmitted GRANT_UNICAST_TRANSMISSION TLV entity grants or denies the request.

A port to which a grant has been made (grantee) may inform the grantor that it no longer needs the granted service. It does this by transmitting a CANCEL_UNICAST_TRANSMISSION TLV entity. A grantor receiving a CANCEL_UNICAST_TRANSMISSION TLV shall always respond with an ACKNOWLEDGE_CANCEL_UNICAST_TRANSMISSION TLV and may immediately cease to provide the indicated service.

A grantor may inform the grantee that it is no longer able to provide the granted service. It does this by transmitting a CANCEL_UNICAST_TRANSMISSION TLV entity.

A grantee receiving a CANCEL_UNICAST_TRANSMISSION TLV shall always respond with an ACKNOWLEDGE_CANCEL_UNICAST_TRANSMISSION TLV and should immediately cease to use the indicated service. The grantor should continue to provide the granted service until either an ACKNOWLEDGE_CANCEL_UNICAST_TRANSMISSION TLV has been received or an implementation-specific number of CANCEL_UNICAST_TRANSMISSION TLVs have been transmitted.

When the grant is of Announce or Sync messages, the grantor shall transmit the messages with a mean inter-message period approximately equal to the granted inter message period. Unless either the grantor or grantee cancels the grant, the transmission shall continue for at least the duration of the grant and start at the time of the transmission of the grant message.

When the grant is of Delay_Resp messages and Delay_Req messages are received from the grantee with a mean inter-message period no smaller than the granted inter message period, the grantor shall respond to each received Delay_Req message with a Delay_Resp message. Unless either the grantor or grantee cancels the grant, this operation shall continue for at least the duration of the grant and start at the time of the transmission of the grant message. If the mean period between reception of Delay_Req messages is less than the granted inter message period, the grantor may ignore the excess Delay_Req messages.

In transmitting messages the inter-message period shall, with 90% confidence, be within +/- 30% of the inter-message period granted in the grant.

For each message type only one grant is active at any time. The reception of a grant message granting transmissions of a particular messageId cancels any previous agreement made with the grantor.

If a unicast contract is negotiated for a particular message type between two ports then any multicast messages of this type between the two ports should be ignored.

When a unicast contract is negotiated for transmitting Delay_Resp messages then the Delay_Req messages associated with the Delay_Resp messages shall also be unicast.

16.1.2 Unicast negotiation enable

The unicast negotiation mechanism can be enabled or disabled by means of the management message UNICAST_NEGOTIATION_ENABLE. By default this mechanism shall be disabled unless otherwise specified in a PTP profile.

If disabled the node shall respond to a UNICAST_NEGOTIATION_ENABLE TLV entity. A disabled node shall not:

- Respond to a REQUEST_UNICAST_TRANSMISSION TLV entity
- Transmit a REQUEST_UNICAST_TRANSMISSION TLV entity
- Transmit a GRANT_UNICAST_TRANSMISSION TLV entity.

16.1.3 Granting port operations

If the requestor issues a new transmission request before the current agreement expires, the grantor should, if resources permit, respond to that request with a grant that is at least as generous as the unexpired portion of the previous grant.

If the granting port considers the combination of inter message period and duration to be unreasonable, the port should reduce the duration of its grant in preference to reducing the rate.

16.1.4 Unicast TLVs

16.1.4.1 REQUEST_UNICAST_TRANSMISSION TLV specification

The REQUEST_UNICAST_TRANSMISSION TLV format shall be as specified in Table 74.

Table 74: REQUEST_UNICAST_TRANSMISSION TLV format

Bits								Octets	TLV offset
7	6	5	4	3	2	1	0		
tlvType								2	0
lengthField								2	2
messageType				reserved				1	4
logInterMessagePeriod								1	5
durationField								4	6

16.1.4.1.1 tlvType

The value of tlvType shall be REQUEST_UNICAST_TRANSMISSION.

16.1.4.1.2 lengthField

The value of the lengthField is 6.

16.1.4.1.3 messageType (Enumeration4)

The value shall indicate the message type for the unicast message transmission requested. The coding of the enumeration is identical to that used in the messageType field of message headers, see 13.3.2.2. Requests for unicast messages other than Announce, Sync, Delay_Resp, or Pdelay_Resp messages shall always be denied. If unicast transmission is granted for Sync or Pdelay_Resp messages by a two-step clock, then unicast transmission shall also be used for the corresponding Follow_Up and Pdelay_Resp_Follow_Up messages.

16.1.4.1.4 logInterMessagePeriod (Integer8)

The value shall be the logarithm, to base 2, of the requested mean period, in seconds, between the requested unicast messages.

16.1.4.1.5 durationField (UInteger32)

The value shall be the requested number of seconds for which the requested messages shall be transmitted.

16.1.4.2 GRANT_UNICAST_TRANSMISSION TLV specification

The GRANT_UNICAST_TRANSMISSION TLV format shall be as specified in Table 75.

Table 75: GRANT_UNICAST_TRANSMISSION TLV format

Bits								Octets	TLV offset
7	6	5	4	3	2	1	0		
tlvType								2	0
lengthField								2	2
messageType				reserved				1	4
logInterMessagePeriod								1	5
durationField								4	6
reserved								1	10
0	0	0	0	0	0	0	R	1	11

16.1.4.2.1 tlvType

The value of tlvType shall be GRANT_UNICAST_TRANSMISSION.

16.1.4.2.2 lengthField

The value of the lengthField is 8.

16.1.4.2.3 messageType (Enumeration4)

The value shall indicate the message type for the unicast message transmission granted. The coding of the enumeration is identical to that used in the messageType field of message headers, see 13.3.2.2. The value shall be identical to the messageType field of the REQUEST_UNICAST_TRANSMISSION TLV request.

16.1.4.2.4 logInterMessagePeriod (Integer8)

The value shall be the logarithm, to base 2, of the granted mean period, in seconds, between the requested unicast messages.

16.1.4.2.5 durationField (UInteger32)

The value shall be the number of seconds for which the messages shall be transmitted. A value of zero shall indicate that the request has been denied.

16.1.4.2.6 R (Renewal Invited) (Boolean)

The value shall be TRUE when the granting port considers that the grant is likely to be renewed when the requesting port repeats its request.

16.1.4.3 CANCEL_UNICAST_TRANSMISSION TLV specification

The CANCEL_UNICAST_TRANSMISSION TLV format shall be as specified in Table 76.

Table 76: CANCEL_UNICAST_TRANSMISSION TLV format

Bits								Octets	TLV offset
7	6	5	4	3	2	1	0		
tlvType								2	0
lengthField								2	2
messageType				reserved				1	4
reserved								1	5

16.1.4.3.1 tlvType

The value of tlvType shall be CANCEL_UNICAST_TRANSMISSION.

16.1.4.3.2 lengthField

The value of the lengthField is 2.

16.1.4.3.3 messageType (Enumeration4)

The value shall indicate the type of unicast message transmission to be cancelled. The coding of the enumeration is identical to that used in the messageType field of message headers, see 13.3.2.2.

16.1.4.4 ACKNOWLEDGE_CANCEL_UNICAST_TRANSMISSION TLV specification

The ACKNOWLEDGE_CANCEL_UNICAST_TRANSMISSION TLV format shall be as specified in Table 77.

Table 77: ACKNOWLEDGE_CANCEL_UNICAST_TRANSMISSION TLV format

Bits								Octets	TLV offset
7	6	5	4	3	2	1	0		
tlvType								2	0
lengthField								2	2
messageType				reserved				1	4
reserved								1	5

16.1.4.4.1 tlvType

The value of tlvType shall be ACKNOWLEDGE_CANCEL_UNICAST_TRANSMISSION.

16.1.4.4.2 lengthField

The value of the lengthField is 2.

16.1.4.4.3 messageType (Enumeration4)

The value shall indicate the type of unicast message cancellation transmission being acknowledged. The value shall be identical to the messageType field in the CANCEL_UNICAST_TRANSMISSION TLV to which this message is the acknowledgement.

16.1.4.5 UNICAST_NEGOTIATION_ENABLE

This message may be used to enable or disable the unicast negotiation mechanism.

A node with the unicast negotiation mechanism enabled, upon receiving a UNICAST_NEGOTIATION_ENABLE TLV with the EN field value FALSE shall cancel all negotiated grants using the CANCEL_UNICAST_TRANSMISSION TLV as defined in 16.1.4.3. Until a CANCEL_UNICAST_TRANSMISSION TLV has been transmitted to all grantees, the node shall report that the negotiation mechanism is enabled in the response to any UNICAST_NEGOTIATION_ENABLE TLV.

The UNICAST_NEGOTIATION_ENABLE management TLV format shall be as specified in Table 78.

Table 78: UNICAST_NEGOTIATION_ENABLE management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
0	0	0	0	0	0	0	EN	1	0
reserved								1	1

16.1.4.5.1 EN (Boolean)

A value of EN of TRUE shall indicate that the unicast transmission mechanism is operational: otherwise it indicates that the mechanism is not operational.

16.2 Path trace (optional)

16.2.1 General

Subclause 16.2 specifies a mechanism, using a TLV, for tracing the route of a PTP Announce message through the timing system. It is an optional mechanism that may be implemented in boundary clocks.

Upon receipt, a boundary clock scans the pathSequence member of the TLV, to see if its own clockIdentity is present, which would indicate that a loop is present. The boundary clock appends its clockIdentity to the tail of the pathSequence member of the TLV, and appends the TLV to outgoing Announce messages.

One of the principal uses of this mechanism is to detect Announce messages endlessly circulating in loops of boundary clocks, so-called rogue frames. If such a loop is detected the received Announce message shall be disregarded. Such loops are eliminated by spanning tree protocols executing on the underlying network, see 6.2.

NOTE—The mechanism of subclause 9.3.2.5 provides a safeguard agains rogue frames introduced by failure or transients in the operation of spanning tree protocols.

In general a boundary clock may receive Announce messages from multiple sources: the current parent clock and any number of foreign master clocks. The <pathTraceList>, 16.2.3, should be maintained only for Announce messages from the current parent. Application of this mechanism to messages other than Announce messages from the current parent is outside the scope of this standard.

16.2.2 Path trace enable

An implementation-specific enable control shall be maintained. If enabled, the path trace mechanism shall be operational. If disabled, the path trace mechanism shall be inactive except for the processing of the PATH_TRACE_ENABLE management TLV. By default, the path trace mechanism shall be disabled unless otherwise specified in a PTP profile.

16.2.3 Path trace list

An implementation-specific list <pathTraceList> of members of type ClockIdentity shall be maintained. The initialization value shall be the empty list.

16.2.4 Change of state

The <pathTraceList>, see 16.2.3, shall be initialized to the empty list whenever the clock updates data sets based on decision code M1 or M2, see 9.3.5.

16.2.5 Receipt of an Announce message

The following additional specifications shall apply to the processing of received Announce messages, see 9.5.3. A port of a boundary clock receiving an Announce message shall:

a) Scan any PATH_TRACE TLV present for a clockIdentity field equal to the clockIdentity field of the defaultDS data set.

b) If the TLV is present and a match is found, the message shall be disregarded.

c) If the TLV is present and no match is found, the clock shall copy the pathSequence member of the TLV to the <pathTraceList>, see 16.2.3.

16.2.6 Transmission of an Announce message

The following additional specifications shall apply to the transmission of Announce messages, see 9.5.8.

A port sending an Announce message shall append a PATH_TRACE TLV to the message. The value of the data field of the PATH_TRACE TLV shall be the <pathTraceList>, see 16.2.3 with the clock's clockIdentity appended to the tail of the list. If the resulting Announce message size exceeds the maximum frame size permitted by the network technology, the PATH_TRACE TLV shall not be appended.

16.2.7 PATH_TRACE TLV specification

The PATH_TRACE TLV format shall be as specified in Table 79.

Table 79: PATH_TRACE TLV format

Bits								Octets	TLV offset
7	6	5	4	3	2	1	0		
tlvType								2	0
lengthField								2	2
pathSequence								8N	4

16.2.7.1 tlvType

The value of tlvType shall be PATH_TRACE.

16.2.7.2 lengthField

The value of the lengthField is 8N.

16.2.7.3 pathSequence (ClockIdentity[N])

The value of pathSequence is a list of clock identities.

16.2.8 PATH_TRACE_LIST management message

This management message TLV may be used to retrieve the current <pathTraceList>, see 16.2.3 from an ordinary or boundary clock. The data field shall be as specified in Table 80.

Table 80: PATH_TRACE_LIST management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
pathSequence								8N	0

16.2.8.1 pathSequence (ClockIdentity[N])

The value of pathSequence is the list of clock identities in the <pathTraceList>, see 16.2.3.

16.2.9 PATH_TRACE_ENABLE management message

This management message may be used to enable or disable the path trace mechanism. The PATH_TRACE_ENABLE TLV data field shall be as specified in Table 80.

Table 81: PATH_TRACE_ENABLE management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
0	0	0	0	0	0	0	EN	1	0
reserved								1	1

16.2.9.1 EN (Boolean)

A value of EN of TRUE shall indicate that the path trace mechanism is operational: otherwise it indicates that the mechanism is not operational.

16.3 Alternate timescales (optional)

16.3.1 General

A node may indicate the offset of an alternate time from its node time by transmitting an ALTERNATE_TIME_OFFSET_INDICATOR TLV entity.

Multiple alternate timescales may be maintained. Each alternate timescale shall be identified by a key and described by a textual display name. The key values shall be consecutive beginning with 0. Each alternate timescale may be enabled or disabled.

NOTE—Alternate timescales require a grandmaster to monitor node time to determine when to transmit the ALTERNATE_TIME_OFFSET_INDICATOR TLV and in most cases require access to external sources for the needed information to determine the time and magnitude of offset discontinuities. For this reason, no provision is made for creating or deleting alternate timescales. Provision is made for modification of alternate timescale attributes.

For each enabled alternate timescale a node shall transmit an ALTERNATE_TIME_OFFSET_INDICATOR TLV entity in all announce messages. If an alternate timescale is disabled, the node shall not transmit this TLV entity.

An alternate timescale may have discontinuities (for example, at the beginning and end of daylight saving time).

The alternate time offset indicator shall not be used to indicate the offset or pending changes in the offset of UTC from the PTP timescale.

If a discontinuity (jump) is about to occur, the node shall indicate this in a contiguous sequence of at least portDataSet.announceReceiptTimeout+1 announce messages transmitted immediately before the discontinuity. The time and magnitude of this discontinuity shall be indicated using the jumpSeconds and timeOfNextJump fields specified in Table 82.

A node shall ignore received ALTERNATE_TIME_OFFSET_INDICATOR TLV entities when jumpSeconds is non zero, indicating a forthcoming discontinuity, and the node's time is after the time contained in timeOfNextJump.

The properties of the alternate timescale mechanism are managed using the ALTERNATE_TIME_OFFSET_ENABLE, ALTERNATE_TIME_OFFSET_NAME, and the ALTERNATE_TIME_OFFSET_PROPERTIES management TLVs.

16.3.2 Forwarding by boundary clocks

Boundary clocks shall forward the information in the ALTERNATE_TIME_OFFSET_INDICATOR TLV.

16.3.3 ALTERNATE_TIME_OFFSET_INDICATOR TLV specification

16.3.3.1 General

The ALTERNATE_TIME_OFFSET_INDICATOR TLV format shall be as specified in Table 82

Table 82: ALTERNATE_TIME_OFFSET_INDICATOR TLV format

Bits								Octets	TLV offset
7	6	5	4	3	2	1	0		
tlvType								2	0
lengthField								2	2
keyField								1	4
currentOffset								4	5
jumpSeconds								4	9
timeOfNextJump								6	13
displayName								L	19
pad								M	19+L

16.3.3.2 tlvType

The value of tlvType shall be ALTERNATE_TIME_OFFSET_INDICATOR.

16.3.3.3 keyField (UInteger8)

The value of keyField shall indicate the alternate timescale reported in this TLV entity.

16.3.3.4 currentOffset (Integer32)

The value of currentOffset shall be the offset of the alternate time, in seconds, from the node's time. The alternate time is the sum of this value and the node's time.

16.3.3.5 jumpSeconds (Integer32)

The value of jumpSeconds shall be the size of the next discontinuity, in seconds, of the alternate time. A value of zero indicates that no discontinuity is expected. A positive value indicates that the discontinuity will cause the currentOffset of the alternate time to increase.

16.3.3.6 timeOfNextJump (UInteger48)

The value of timeOfNextJump shall be the of the seconds portion of the transmitting node's time at the time that the next discontinuity will occur. The discontinuity occurs at the start of the second indicated by the value of timeOfNextJump.

16.3.3.7 displayName (PTPText)

The value of displayName shall be the text name of the alternate timescale.

NOTE—Commonly used acronyms should be used, e.g. NTP, PST, PDT for Network Time Protocol, Pacific Standard Time, and Pacific Daylight Savings Time respectively.

The maximum number of characters in this field shall be 10.

16.3.3.8 pad (Octet[M])

The pad field either shall be an octet array of length M where M is 1 with all bits 0 or shall be of length 0, whichever is required to make the size of the data field an even number of octets, see 15.5.2.2.

16.3.4 ALTERNATE_TIME_OFFSET_ENABLE management message

This management TLV allows the indicated alternate timescale to be enabled or disabled in the grandmaster clock. If this TLV is received by a clock other than the grandmaster, the contents shall be disregarded and a MANAGEMENT_ERROR_STATUS TLV shall be returned.

The maintenance of this data is implementation-specific..

The ALTERNATE_TIME_OFFSET_ENABLE management TLV data format shall be as specified in Table 83.

Table 83: ALTERNATE_TIME_OFFSET_ENABLE management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
keyField								1	0
0	0	0	0	0	0	0	EN	1	1

16.3.4.1 keyField (UInteger8)

The value shall indicate the alternate timescale enabled or disabled by this TLV entity. A value of FF_{16} shall indicate that all alternate timescales maintained by the grandmaster clock are to be enabled or disabled. If the value is not associated with a maintained alternate timescale, the contents shall be disregarded and a MANAGEMENT_ERROR_STATUS TLV shall be returned.

16.3.4.2 EN (Boolean)

If TRUE, the ALTERNATE_TIMESCALE_OFFSET_INDICATOR for the timescale indicated by the keyField value shall be attached to Announce messages. If FALSE, the TLV shall not be attached.

16.3.5 ALTERNATE_TIME_OFFSET_NAME TLV specification (optional)

This management TLV allows the grandmaster clock to be configured with the timescale offset description attributes for each selected alternate timescale. If this TLV is received by a clock other than the grandmaster, the contents shall be disregarded and a MANAGEMENT_ERROR_STATUS TLV shall be returned.

The maintenance of this data is implementation-specific.

The ALTERNATE_TIME_OFFSET_NAME management TLV data format shall be as specified in Table 84.

Table 84: ALTERNATE_TIME_OFFSET_NAME management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
keyField								1	0
displayName								L	1
pad								M	1+L

16.3.5.1 keyField (UInteger8)

The value of keyField shall indicate the alternate timescale updated or queried by this TLV entity.

If the value is FF_{16} or is not associated with any maintained alternate timescale, the TLV shall be ignored and a MANAGEMENT_ERROR_STATUS TLV returned.

16.3.5.2 displayName (PTPText)

The value of displayName shall be the text name of the alternate timescale, see 16.3.3.7 .

16.3.5.3 pad (Octet[M])

The pad field either shall be an octet array of length M where M is 1 with all bits 0 or shall be of length 0, whichever is required to make the size of the data field an even number of octets, see 15.5.2.2.

16.3.6 ALTERNATE_TIME_OFFSET_MAX_KEY management TLV

This management TLV allows the grandmaster clock to report the number of alternate timescales maintained. If this TLV is received by a clock other than the grandmaster, the contents shall be disregarded and a MANAGEMENT_ERROR_STATUS TLV shall be returned.

The maintenance of this data is implementation-specific.

The ALTERNATE_TIME_OFFSET_MAX_KEY management TLV data format shall be as specified in Table 85.

Table 85: ALTERNATE_TIME_OFFSET_MAX_KEY management TLV data field

Bits								Octets	TLV offset
7	6	5	4	3	2	1	0		
maxKey								1	0
reserved								1	1

16.3.6.1 maxKey (UInteger8)

The value of maxKey shall indicate the value of the largest key.

16.3.7 ALTERNATE_TIME_OFFSET_PROPERTIES management TLV (optional)

This management TLV allows the grandmaster clock to be configured with the timescale offset attributes for each selected alternate timescale. If this TLV is received by a clock other than the grandmaster, the contents shall be disregarded and a MANAGEMENT_ERROR_STATUS TLV shall be returned.

If this TLV is received with and action value of SET, the update of currentOffset, jumpSeconds, and timeOfNextJump shall be atomic. If any of these values fail to update a MANAGEMENT_ERROR_STATUS TLV shall be returned.

The maintenance of this data is implementation-specific.

The ALTERNATE_TIME_OFFSET_PROPERTIES management TLV data format shall be as specified in Table 86.

Table 86: ALTERNATE_TIME_OFFSET_PROPERTIES management TLV data field

Bits								Octets	TLV offset
7	6	5	4	3	2	1	0		
keyField								1	0
currentOffset								4	1
jumpSeconds								4	5
timeOfNextJump								6	9
reserved								1	15

16.3.7.1 keyField (UInteger8)

The value of keyField shall indicate the alternate timescale updated or queried by this TLV entity.

If the value is FF_{16} or is not associated with any maintained alternate timescale, the TLV shall be ignored and a MANAGEMENT_ERROR_STATUS TLV returned.

16.3.7.2 currentOffset (Integer32)

The value of currentOffset shall be the offset of the alternate time, see 16.3.3.4.

16.3.7.3 jumpSeconds (Integer32)

The value of jumpSeconds shall be the size of the next discontinuity, see 16.3.3.5.

16.3.7.4 timeOfNextJump (UInteger48)

The value of timeOfNextJump shall be the time that the next discontinuity will occur, see 16.3.3.6.

17. State configuration options

17.1 General

Many applications require one or more of the following capabilities:

- Automatic recovery from a break in the communication network,
- Automatic recovery from the failure of a clock,
- Explicit control over the selection of port state.

This standard provides several features designed to meet these requirements.

The operation of the best master clock algorithm and state machine in ordinary and boundary clocks, Clause 9, ensures that a master-slave hierarchy is established with the best clock present in the system being the grandmaster. This provides automatic recovery from both network failure and failure of individual clocks. The rate of recovery is dependent on the announceInterval and the topology of the network.

Peer-to-peer transparent clocks, Clause 10, provide for rapid recovery in the event of network reconfiguration. Since the peer delay mechanism, see 11.4, measures the path delays on all links in the event of a reconfiguration the needed path delay correction information is immediately available.

Clause 17 specifies additional optional features that may be used in conjunction with a best master clock algorithm to enhance performance or to exert more control over the selection of port state.

NOTE—These options should be used with care. For example, if some clocks connected to a communication path are configured to use the Acceptable Master Table and some are not, it is possible that more than one port will consider itself to be the best master. If clocks connected to a communication path are configured with incompatible Acceptable Master Tables, it is possible that more than one port will consider itself to be the best master. Similar misconfiguration inconsistencies can occur with the configuration mechanism of any of these options.

17.2 Data types for options

17.2.1 General

The data types in 17.2 are used in one or more of the options of Clause 17. These data type specifications of 17.2 shall be used in addition to those in Clause 5 for any implemented option of Clause 17 that references one of the following:

- PortAddressQueryTable
- AcceptableMaster
- AcceptableMasterTable

All specifications of Clause 5 apply to the data types defined in 17.2.

17.2.2 Port address query table

The PortAddressQueryTable type represents a list of port addresses along with the query interval.

struct PortAddressQueryTable

```
{
                UInteger16 maxTableSize;
                Integer8 logQueryInterval;
                UInteger16 actualTableSize;
                PortAddress[actualTableSize] portAddresses;
};
```

The value of maxTableSize is implementation-specific.

17.2.3 Acceptable master

```
struct AcceptableMaster
{
                PortAddress address;
                UInteger8 alternatePriority1;
};
```

17.2.4 Acceptable Master Table

```
struct AcceptableMasterTable
{
                UInteger16 maxTableSize;
                UInteger16 actualTableSize;
                AcceptableMaster[actualTableSize] acceptableMaster;
};
```

17.3 Master clusters (optional)

17.3.1 General specification

When the normal operation of the best master clock algorithm is used, the time to recover from the failure of a grandmaster clock depends on the announceInterval. If the required time to change from one grandmaster to another is incompatible with the multicast announceInterval, the mechanism of 17.3 may be used to decrease the time required to select a new grandmaster. If this option is implemented, the unicast negotiation option, see 16.1, shall also be implemented.

Two or more ordinary or boundary clocks may be designated as a master cluster. For correct operation each clock in the cluster should be configured with values of priority1 such that the priority1 values of all members of the cluster are less than the priority1 values of all other clocks in the domain.

NOTE—Although designed for use in improving the change over of grandmaster clocks, in some topologies this mechanism may be useful for designating clusters of master clocks below the grandmaster in the master-slave hierarchy. The configuring of master clusters for such use is out of scope of this standard.

17.3.2 Operation of the master cluster

Each clock in the cluster shall:

a) Maintain a configured table of Masters, the Master Cluster Table, with data type PortAddressQueryTable, see 17.2.2. The portAddress members of the table shall each hold the protocol address of another member of the master cluster.

b) Use the unicast message negotiation option, see 16.1, to periodically request unicast Announce messages from all the ports listed in the portAddress members of the Master Cluster Table.

c) set alternateMasterFlag to FALSE if the port is transmitting a unicast Announce message under the terms of this subclause and is in the MASTER state, and set to TRUE otherwise.

d) Insert the value of the logQueryInterval member of the Master Cluster Table into the logInterMessageInterval field of the REQUEST_UNICAST_TRANSMISSION TLV.

e) Request a renewal prior to the expiration of each negotiated unicast transmission.

f) Use the received unicast Announce messages from the cluster members, irrespective of the value of alternateMasterFlag , to exercise the best master clock algorithm to determine the portState of each of its ports.

If the Master Cluster Table is empty, i.e. actualTableSize is 0, this option shall be inactive except for the processing of the MASTER_CLUSTER_TABLE management TLV.

The maintenance of this information is implementation-specific and is not part of the node's data sets of Clause 8.

17.3.3 MASTER_CLUSTER_TABLE management TLV data field specification

17.3.3.1 General specifications

Upon receipt of a management message with managementID MASTER_CLUSTER_TABLE and action field value of Set, the clock shall replace the current portAddress members of the Master Cluster Table with the masterClusterMembers of the management message. The member identifying the recipient clock shall not be entered into the master cluster table. If any member fails to update, and MANAGEMENT_ERROR_STATUS TLV shall be returned with the managementErrorID GENERAL_ERROR.

If the tableSize member of the TLV is non-zero and port address of the receiving node does not appear in the list of masterClusterMembers of the TLV, the management message shall be rejected and the Master Cluster Table shall not be updated. For this case, the managementErrorID shall be WRONG_VALUE.

If the PortAddress array in the masterClusterMemeber of the TLV cannot be fully stored, the Master Cluster Table shall not be altered and the management message shall be rejected. For this case the managementErrorID shall be WRONG_LENGTH if the failure is due to insufficient space in the master cluster table and GENERAL_ERROR for other failures.

The receipt of a TLV with tableSize 0 shall cause the receiving clock to clear all members from the array of port addresses in the Master Cluster Table.

If the Master Cluster Table is empty (actualTableSize is 0), this option shall be inactive except for the processing of the MASTER_CLUSTER_TABLE management TLV. The default value of actualTableSize shall be 0 unless otherwise specified in a PTP profile.

17.3.4 MASTER_CLUSTER_TABLE management TLV

The management TLV data field shall be as specified in Table 87.

Table 87: MASTER_CLUSTER_TABLE management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
logQueryInterval								1	0
tableSize								1	1

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
masterClusterMembers								M	2
pad								0 or 1	2+M

17.3.4.1 logQueryInterval (Integer8)

The value shall be the logarithm to the base 2 of the mean interval in seconds between unicast Announce messages from cluster members.

17.3.4.2 tableSize (UInteger8)

The value shall be the number of entries in the masterClusterMembers array. The maximum number of entries shall be 5.

17.3.4.3 masterClusterMembers (PortAddress[tableSize])

The PortAddress elements of the masterClusterMembers array shall carry the respective protocol addresses of the members of the Master Cluster Table of 17.3.2.

17.3.4.4 pad (Octet[M])

The pad field either shall be an octet array of length M where M is 1 with all bits 0 or shall be of length 0, whichever is required to make the size of the data field an even number of octets, see 15.5.2.2.

17.4 Alternate master (optional)

17.4.1 General

This option allows alternate masters that are not currently the best master to be visible to slave ports, and for a slave port to acquire knowledge of the characteristics of the transmission path between itself and each alternate master. This will allow a slave switch over to an alternate master with a small phase excursion when the best master fails.

17.4.2 Transmission of messages by alternate masters

A port shall transmit multicast Announce messages subject to the restrictions in Table 88. A port transmitting Announce message under the terms of 17.4 shall set alternateMasterFlag , see 13.3.2.6, to TRUE. These messages shall be transmitted at the interval defined by logAnnounceInterval, see 8.2.5.4.1.

A port shall transmit multicast Sync, and, if a two-step clock, Follow_Up messages subject to the restrictions in Table 88. A port transmitting Sync or Follow_Up message under the terms of 17.4 shall set alternateMasterFlag to TRUE. These messages shall be transmitted at the interval defined by log_alternate_multicast_sync_interval in Table 88.

NOTE—A slave node that does not want to use information from alternate masters merely ignores all messages with alternateMasterFlag TRUE.

A port shall maintain the configurable attributes specified in Table 88. The maintenance of this information is implementation-specific and is not part of the node’s data sets of Clause 8.

Table 88: Alternate master attributes

Name	Type	Description
<numberOfAlternateMasters>	UInteger8	The port, if not in the MASTER state, shall transmit multicast Announce messages when the number of visible ports that: • Are currently transmitting Announce messages with alternateMasterFlag set, and • Would be chosen using the best master algorithm as best master in preference to this port is less than <numberOfAlternateMasters>. The default value for <numberOfAlternateMasters> shall be 0.
<transmitAlternateMulticastSync>	Boolean	If TRUE and the port is currently transmitting multicast Announce messages with alternateMasterFlag TRUE, the port shall also transmit multicast Sync and, if a two-step clock, Follow_Up messages.
<logAlternateMulticastSync_interval>	Integer8	The logarithm to the base 2 of the mean period in seconds between Sync messages transmitted under the terms of 17.4.

NOTE 1—A port is visible to port-A if it is transmitting Announce messages that are received by port-A.

NOTE 2—The default value of <numberOf_alternateMasters> causes multicast Announce messages to be transmitted only when the port is in the MASTER state.

17.4.3 ALTERNATE_MASTER management TLV data field

The alternate master attributes in Table 88 may be updated using a management message with managementID ALTERNATE_MASTER.

If this TLV is received with and action value of SET, the updates of logAlternateMulticastSyncInterval, and numberOfAlternateMasters shall be atomic. If either of these values fails to update a MANAGEMENT_ERROR_STATUS TLV shall be returned.

The ALTERNATE_MASTER management TLV data format shall be as specified in Table 89.

Table 89: ALTERNATE_MASTER management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
0	0	0	0	0	0	0	S	1	0
logAlternateMulticastSyncInterval								1	1
numberOfAlternateMasters								1	2
reserved								1	3

17.4.3.1.1 S (Boolean)

The value of S shall be the value of <transmitAlternateMulticastSync>.

17.4.3.1.2 logAlternateMulticastSyncInterval (Integer8)

The value of logAlternateMulticastSyncInterval shall be the value of <logAlternateMulticastSyncInterval>.

17.4.3.1.3 numberOfAlternateMasters (UInteger8)

The value of numberOfAlternateMasters shall be the value of number_of_alternate_masters.

17.5 Unicast discovery (optional)

17.5.1 General

This option allows PTP to be used over a network that does not provide multicast (for example, many IP networks). A slave port is configured with the addresses of potential masters. The slave may request that these masters transmit unicast Announce, Sync and Delay_Resp messages to it. If this option is implemented, the unicast negotiation option, see 16.1, shall also be implemented.

17.5.2 Operation of Unicast discovery

An ordinary or boundary clock shall maintain a configured table of Masters, the Unicast Master Table. This table shall have the data type PortAddressQueryTable, see 17.2.2. The portAddress members of the table each hold the protocol address of a remote port with which this node shall attempt to establish communication.

The node shall use the unicast message negotiation option, see 16.1, to periodically request unicast Announce messages from all the ports listed in the Unicast Master Table. If a request is not granted by a port, the request shall be repeated after the delay indicated by the logQueryInterval member of the table.

If the Unicast Master Table is empty (actualTableSize is 0), this option shall be inactive except for the processing of the UNICAST_MASTER_TABLE and UNICAST_MASTER_MAX_TABLE_SIZE management TLVs.

A port shall maintain the configurable attributes specified in Table 90. The maintenance of this information is implementation-specific and is not part of the node's data sets of Clause 8.

The default value of actualTableSize is 0 unless otherwise specified in a PTP profile.

17.5.3 UNICAST_MASTER_TABLE management TLV data field

The management TLV data field shall be as specified in Table 90.

If this TLV is received with and action value of SET, the update of logQueryInterval, tableSize, and unicastMasterTable shall be atomic. If any of these values fail to update a MANAGEMENT_ERROR_STATUS TLV shall be returned.

Table 90: UNICAST_MASTER_TABLE management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
logQueryInterval								1	0
tableSize								2	1
unicastMasterTable								M	3
pad								0 or 1	3+M

17.5.3.1 logQueryInterval (Integer8)

The value shall be the logarithm to the base 2 of the mean interval in seconds between request from a node for a unicast Announce message.

17.5.3.2 tableSize (UInteger16)

The value shall be the number of entries in the unicastMasterTable.

17.5.3.3 unicastMasterTable (PortAddress[tableSize])

The PortAddress members of the unicastMasterTable array shall carry the respective protocol address of each member of the unicast Master Table of 17.5.2.

17.5.3.4 pad (Octet[M])

The pad field either shall be an octet array of length M where M is 1 with all bits 0 or shall be of length 0, whichever is required to make the size of the data field an even number of octets, see 15.5.2.2.

17.5.4 UNICAST_MASTER_MAX_TABLE_SIZE management TLV data field

The management TLV data field shall be as specified in Table 90.

Table 91: UNICAST_MASTER_MAX_TABLE_SIZE management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
maxTableSize								2	0

17.5.4.1 maxTableSize (UInteger16)

The value shall be the value of the maximum number of records of data type PortAddress that can be maintained in the Unicast Master Table.

17.6 Acceptable master table (optional)

17.6.1 General

This option allows slave ports to be configured to refuse to synchronize to clocks not on the acceptable master list.

NOTE—This may be used to rule out synchronization to suspected rogue or spurious masters.

17.6.2 Operation of the acceptable master table

An ordinary or boundary clock shall maintain a configured table, the Acceptable Master Table of type AcceptableMasterTable, and a per port configurable Boolean value acceptable_master_table_enabled.

The default value of acceptable_master_table_enabled shall be FALSE unless otherwise specified in a PTP profile.

The maintenance of this information is implementation-specific and is not part of the node's data sets of Clause 8.

The operation of the acceptable master table option on each port shall be as specified in Table 92

Table 92: Operation of acceptable master table option

acceptable_master_table_enabled	Operational specification
FALSE	The Acceptable Master Table is not used on this port. The normal operation of the protocol is in effect. The port shall process ACCEPTABLE_MASTER_TABLE_ENABLED management TLVs.
TRUE	The port indicated by the value of E_{rbest} determined by the best master clock algorithm, see 9.3, shall be a member of the Acceptable Master Table. If more than one member of the Acceptable Master Table is visible to this port, the normal data set comparison algorithm of 9.3.4 shall be used to select E_{rbest} from the visible members of this table. If the alternatePriority1 member of the AcceptableMaster member of the table for a port is 0 the alternatePriority1 member shall have no effect on the computation of E_{rbest}. If the value of the alternatePriority1 member is greater than 0, the value of priority1 in the Announce message from the remote port shall be replaced by the value of the alternatePriority1 member of this table for purposes of computing E_{rbest}.

The term visible in the context of the operation of the acceptable master table option shall be as follows:

A node is visible to this node when the time since the receipt of the most recent qualified Announce message is not greater than defined by the announceReceiptTimeout member of the portDS data set of the port executing the acceptable master table option.

17.6.3 ACCEPTABLE_MASTER_TABLE management TLV data field

The management TLV data field shall be as specified in Table 90.

If this TLV is received with and action value of SET, the update of tableSize, and acceptableMasterTable shall be atomic. If either of these values fail to update a MANAGEMENT_ERROR_STATUS TLV shall be returned.

Table 93: ACCEPTABLE_MASTER_TABLE management TLV data field

Bits								Octets	TLV data offset
7	6	5	4	3	2	1	0		
tableSize								2	0
acceptableMasterTable								M	2
pad								0 or 1	2+M

17.6.3.1 tableSize (Integer16)

The value of tableSize shall be the number of entries in the acceptableMasterTable.

17.6.3.2 acceptableMasterTable (AcceptableMaster[tableSize])

The AcceptableMaster members of the acceptableMasterTable array shall carry the respective protocol address and alternatePriority1 values of each member of the Acceptable Master Table of 17.6.2.

17.6.3.3 pad (Octet[M])

The pad field either shall be an octet array of length M where M is 1 with all bits 0 or shall be of length 0, whichever is required to make the size of the data field an even number of octets, see 15.5.2.2.

17.6.4 ACCEPTABLE_MASTER_MAX_TABLE_SIZE management TLV data field

The management TLV data field shall be as specified in Table 94.

Table 94: ACCEPTABLE_MASTER_MAX_TABLE_SIZE management TLV data field

Bits								**Octets**	**TLV data offset**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
maxTableSize								2	0

17.6.4.1 maxTableSize (UInteger16)

The value shall be the value of the maximum number of records of data type AcceptableMaster that can be maintained in the Acceptable Master Table.

17.6.5 ACCEPTABLE_MASTER_TABLE_ENABLED management TLV data field

The management TLV data field shall be as specified in Table 95.

Table 95: ACCEPTABLE_MASTER_TABLE_ENABLED management TLV data field

Bits								**Octets**	**TLV data offset**
7	**6**	**5**	**4**	**3**	**2**	**1**	**0**		
0	0	0	0	0	0	0	EN	1	0
reserved								1	1

17.6.5.1 EN (Boolean)

The value of EN shall be the value of acceptableMasterTableEnabled.

18. Compatibility requirements

18.1 Compatibility between version 2 and future versions

A PTP node A implemented using version VA, upon receiving a PTP message from a PTP node B with a versionPTP field value, VB, greater than VA, shall:

a) Disregard the received version VB message if it is not defined in version VA, otherwise

b) Parse and execute the version VB messages that are defined in version VA. Disregard TLV extensions of the version VB message that are not defined in version VA.

c) If during the parsing of the VB message inconsistencies are detected, for example an incorrect length, the message shall be disregarded.

18.2 Compatibility between version 1 and version 2

A node is not required to support translation between version 1 and version 2. If translation is supported it shall be implemented per 18.3 and 18.4.

PTP nodes implemented under version 1 of this standard should communicate with nodes implemented under version 2 or higher of this standard via a boundary clock designed for this purpose. Such a boundary clock :

a) Communicates with the version 1 node via a port that implements the messaging and protocol requirements of version 1, and

b) Communicates with the version 2 or higher node via a port that implements the messaging and protocol requirements of version 2 or higher, and

c) Internally resolves differences in the operation of the two protocol versions, including translating message formats and resolving differences in the values of attributes as specified in the following clauses.

Transparent clock translation devices are out of scope of this standard.

Translation devices shall send version 1 messages only on ports for which the versionNumber attribute of the port's data set is set to 1. Version 1 messages received on a port for which the versionNumber attribute is set to 2 shall be ignored.

NOTE 1— Automatic detection of the version used within a communication path is out of scope of this standard.

NOTE 2— Multicast management message forwarding between version 1 and version 2 ports is outside the scope of this standard.

Specific requirements and restrictions on translating between version 1 and version 2 or higher boundary or ordinary clock nodes are covered in the following subclauses.

18.3 Message formats and data types

18.3.1 Domains

The translating device shall:

a) Map the version 1 subdomain value _DFLT to version 2 domain value 0 and vice versa,

b) Map the version 1 subdomain value _ALT1 to version 2 domain value 1 and vice versa,

c) Map the version 1 subdomain value _ALT2 to version 2 domain value 2 and vice versa,

d) Map the version 1 subdomain value _ALT3 to version 2 domain value 3 and vice versa,

The mappings of domains other than those specified above is outside the scope of this standard.

18.3.2 Version 1 Stratum and version 2 or higher clockClass

The translating device shall map version 1 stratum values to version 2 clockClass values as specified in Table 96.

Table 96: Version 1 stratum to version 2 class

Version 1 stratum	Version 2 clockClass
0	6
1	9
2	10
3	248
4	251
255	255

Except as provided in Table 99, the translating device shall map version 2 clockClass value to version 1 stratum values as specified in Table 97.

Table 97: Version 2 clockClass to version 1 stratum

Version 2 clockClass	Version 1 stratum
6	0
7	0
9	1
10	2
13-248	3
251	4
255	255

18.3.3 Version 1 preferred and version 2 priority1

The translating device shall map the version 1 grandmasterIsPreferred field into the version 2 or higher priority1 field as shown in Table 98.

Table 98: Version 1 to version 2 or higher translation of grandmasterIsPreferred field

grandmasterIsPreferred	priority1
0	128
1	127

The translating device shall map the version 2 or higher priority1 field into the version 1 grandmasterIsPreferred, and grandmasterClockStratum fields as shown in Table 99.

Table 99: Version 2 or higher to version 1 translation of the priority1 field

V2	V1	
priority1	grandmasterIsPreferred	grandmasterClockStratum

0-126	1	0
127	1	Per Table Table 97
128	0	Per Table Table 97
>128	0	255

18.3.4 Version 1 clock identifier and Version 2 clockAccuracy and timeSource

There is no mapping that preserves the semantics of these attributes.

The translating device shall map version 1 clock identifier to version 2 clockAccuracy values as specified in Table 100. The values of the version 2 timeSource attribute shall be set as shown in Table 100.

Table 100: Version 1 clock identifier to version 2 clockAccuracy

Version 1 clock identifier	Version 2 clockAccuracy	Version 2 timeSource
ATOM	22_{16}	ATOMIC_CLOCK
GPS	22_{16}	GPS
NTP	$2F_{16}$	NTP
HAND	30_{16}	HAND_SET
INIT	FD_{16}	OTHER
DFLT	FE_{16}	INTERNAL_OSCILLATOR

The translating device shall map version 2 clockAccuracy to version 1 clock identifier values as specified in Table 101. Version 2 timeSource values shall be disregarded when translating to version 1.

Table 101: Version 2 clockAccuracy to version 1 clock identifier

Version 2 clockAccuracy	Version 1 clock identifier
20 - 22_{16}	ATOM
23 - $2F_{16}$	NTP
30_{16}	HAND
31 - FD_{16}	INIT
FE_{16}	DFLT

18.3.5 Version 1 grandmasterIsboundaryClock fields and version 2 or higher priority2 fields

The translating device shall map the version 1 grandmasterIsboundaryClock field into the version 2 or higher priority2 fields as shown in Table 102.

Table 102: Version 1 to version 2 or higher translation of grandmasterIsboundaryClock field

grandmasterIsboundaryClock	priority2
FALSE	128
TRUE	127

The translating device shall map the version 2 or higher priority2 field into the version 1 grandmasterIsboundaryClock field as shown in Table 103.

Table 103: Version 2 or higher to version 1 translation of the priority2 field

V2 priority2	V1 grandmasterIsBoundaryClock
0-127	TRUE

128-255	FALSE

18.3.6 Version 1 control and version 2 or higher messageType fields

The translating device mapping between the version 1 control field and the version 2 or higher messageType field shall be as shown in Table 104.

Table 104: Version 1 control field and version 2 or higher messageType field mappings

Message	Message class	Version 2 or higher messageType value	Version 1 control value	Version 1 messageType
Sync	Event	0_{16}	00_{16}	01_{16}
Delay_Req	Event	1_{16}	01_{16}	01_{16}
Pdelay_Req	Event	2_{16}	N/A	N/A
Pdelay_Resp	Event	3_{16}	N/A	N/A
Reserved		$4\text{-}7_{16}$	N/A	N/A
Followup	General	8_{16}	02_{16}	02_{16}
Delay_Resp	General	9_{16}	03_{16}	02_{16}
Pdelay_Resp_Follow_Up	General	A_{16}	N/A	N/A
Announce	General	B_{16}	N/A	N/A
Signaling	General	C_{16}	N/A	N/A
Management	General	D_{16}	04_{16}	02_{16}
Reserved		$E\text{-}F_{16}$	N/A	N/A
Reserved		N/A	$05\text{ - }FF_{16}$	All other values

18.3.7 Version 2 sourcePortIdentity and version 1 sourceCommunicationTechnology, sourceUuid, and sourcePortId

Version 1 sourceCommunicationTechnology, which had a value 1 for all conformant devices, maps into version 2 Communication Technology “version 1 devices” in Table 4 and vice versa.

NOTE—While version 1 Table 2 enumerated many values of sourceCommunicationTechnology, the only transport mapping defined in version 1 was to Ethernet (version 1 Annex D), which corresponds to the sourceCommunicationTechnology value 1.

Version 1 sourceUuid maps into Octets 2-7 of the clockIdentity member of the sourcePortIdentity field, per 7.5.2.2.3.

Version 2 clockIdentity member of the sourcePortIdentity:

— If from 7.5.2.2.3: Octets 2-7 map into the clockUuid, and Communication Protocol maps into the version 1 defined communicationId for the same protocol

— If from 7.5.2.2.2: the version 1 communicationId shall be 0. The six least significant octets of the EUI-64 version 2 clockIdentity shall be mapped into the 6 octets of the version 1 clockUuid field.

Version 1 sourcePortId maps into version 2 portNumber member of the sourcePortIdentity field and vice versa.

18.3.8 Version 2 grandmasterIdentity and version 1 grandmasterCommunicationTechnology, grandmasterClockUUID, and grandmasterPortId

The mapping rules are identical to the mapping rules of 18.3.7.

18.3.9 Version 2 parentPortIdentity and version 1 parentCommunicationTechnology, parentClockUuid, and parentPortId

The mapping rules are identical to the mapping rules of 18.3.7.

18.3.10 Flags fields of common headers

The translation of common header flags from version 1 to version 2 shall be as specified in Table 105.

Table 105: Translation of flags field from version 1 to version 2

Version 2 or higher flag	**Set to value of version 1 attribute**
currentUtcOffsetValid	TRUE if stratum = 1 or 2 and identifier is not INIT or DFLT. Otherwise FALSE
LI_61	PTP_LI_61
LI_59	PTP_LI_59
alternateMasterFlag	FALSE
twoStepFlag	PTP_ASSIST
timeTraceable	TRUE if stratum = 1 or 2 and identifier is not INIT or DFLT. Otherwise FALSE
frequencyTraceable	TRUE if stratum = 1 or 2 and identifier is not INIT or DFLT. Otherwise FALSE
ptpTimescale	TRUE if identifier is not INIT or DFLT. Otherwise FALSE
unicastFlag	FALSE
All other flag fields	Set to FALSE

The translation of common header flags version 2 to version 1 shall be as specified in Table 106.

Table 106: Translation of flags field from version 2 to version 1

Version 1 flags	**Set to value of version 2 or higher attribute**
Discard the information	currentUtcOffsetValid
PTP_LI_61	LI_61
PTP_LI_59	LI_59
If ALTERNATE_MASTER is TRUE, the version 2 message shall not be transmitted into the version 1 region.	alternateMasterFlag
PTP_ASSIST	twoStepFlag
If unicastFlag is TRUE the version 2 message shall not be transmitted into the version 1 region.	unicastFlag
PTP_SYNC_BURST	FALSE
PARENT_STATS	FALSE
PTP_EXT_SYNC	FALSE
PTP_BOUNDARY_CLOCK	FALSE
Discard the information	All other flags

18.3.11 Version 2 logMessageInterval and version 1 syncInterval of Sync messages

The logMessageInterval of version 2 Announce, Delay_Req and Delay_Resp messages are ignored in translating into version 1.

The logMessageInterval of a version 2 Sync or Follow_Up message is translated into the syncInterval of version 1 Sync messages.

The syncInterval of a version 1 Sync message is translated into the logMessageInterval for version 2 Announce, Sync and Follow_Up messages.

The syncInterval of a version 1 Sync message incremented by +5 is translated into the logMessageInterval for version 2 Delay_Req and Delay_Resp messages.

NOTE—It is necessary to increment the version 1 syncInterval by +5 to obtain the logMessageInterval for version 2 Delay_Req and Delay_Resp messages to meet the requirements of 7.7.2.4.

18.3.12 Version 2 fields of type ClockQuality and version 1 stratum, identifier and variance fields

Version 2 field grandmasterClockQuality is mapped between the version 1 counterparts as follows:

— Version 2 clockClass member to grandmasterClockStratum mapping is per 18.3.2

— Version 2 clockAccuracy member to grandmasterClockIdentifier mapping is per 18.3.4

— Version 2 offsetScaledLogVariance member maps directly to the grandmasterClockVariance after correcting for the offset of 8000_{16}, see 7.6.3.3.

18.3.13 Version 2 fields of type Timestamp and version 1 epochNumber and fields of type TimeRepresentation

Version 2 nanoseconds member UInteger32 maps to version 1 nanoseconds member Integer32.

Version 1 nanoseconds member if positive maps to version 2 nanoseconds member. If negative, an error should be generated since negative timestamps are not permitted in version 2.

The least significant 32 bits of the version 2 seconds field UInteger48 map directly to the version 1 seconds field and vice versa.

The most significant 16 bits of the version 2 seconds field UInteger48 map directly to the version 1 epochNumber and vice versa.

18.3.14 Version 2 or higher fields that have no version 1 counterpart

For each version 2 or higher field shown in Table 107, the translating device shall take the actions specified.

Table 107: Version 2 or higher fields with no version 1 counterpart

Version 2 or higher field	**Message**	**Version 2 to version 1**	**Version 1 to version 2**
transportSpecific	Common header	Discard information	Set per applicable Annex D - I
messageLength	Common header	Discard information	Set per 13.3.2.4
correctionField	Common header	Out of scope of this standard	Set to 0.

18.3.15 Version 1 fields that have no version 2 or higher counterpart

For each version 1 field shown in Table 108, the translating device shall take the actions specified in translating to version 2.

Table 108: Version 1 fields with no version 2 or higher counterpart

Version 1 field	Message	Action
versionNetwork	Common header	Discard information
messageType		
utcReasonable	Sync and Delay_Req	Discard information
localClockVariance		
localClockStratum		
localClockIdentifier		
estimatedMasterVariance		
estimatedMasterDrift		
associatedSequenceId	Follow_Up	Maps to sequenceId of common header in version 2
requestingSourceSequenceId	Delay_Resp	Maps to sequenceId of common header in version 2
managementMessageKey	Management	Map between version 1 and version 2 format and semantics for each management message.
parameterLength		
messageParameters		

18.4 Naming changes

Table 109 shows the correspondence between version 1 and version 2 names for the same quantity for cases in which the semantics remain unchanged.

NOTE—Quantities for which semantics have changed are covered in previous clauses.

Table 109: Name correspondence

Version 1 name	Version 2 name
estimatedMasterVariance	observedParentOffsetScaledLogVariance of portDS data set
estimatedMasterDrift	observedParentClockPhaseChangeRate of portDS data set

18.5 Restrictions on mixed version 1 and version 2 systems.

The translations specifications of Clause 18 permit mixed version 1 and 2 systems with the restrictions specified in this subclause.

Mixed version 1 and version 2 systems should be configured as shown in Figure 36 subject to the restrictions in any one of the rows of Table 110. Version 1 stratum 3 clocks shall not be used in the implementation of mixed version 1 and version 2 systems.

Other system configurations and limitations may be possible but are out of scope.

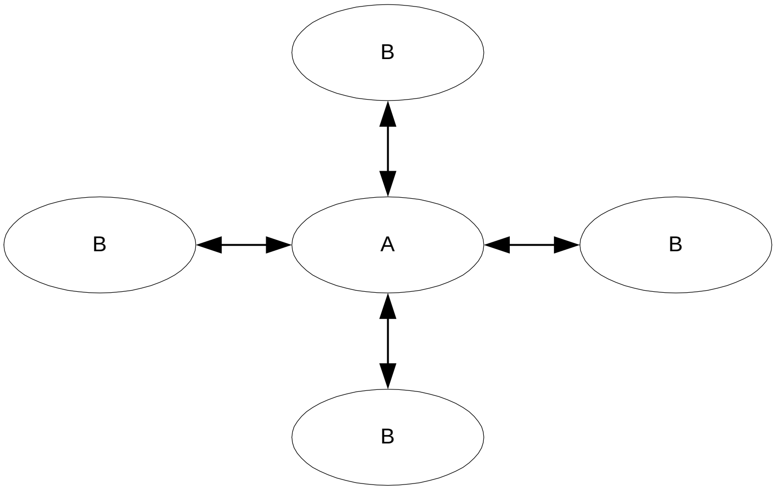


Figure 36: Permitted mixed system configuration

Shown in Figure 36 is a region A implementing one version of PTP connected to one or more regions implementing a second version of PTP.

Table 110: Mixed system restrictions

Region A characteristics	**Region B characteristics**	**Restrictions**
Version 2 clocks	Version 1 clocks	No class 6 or 7 or stratum 1 or 2 clocks in the system.
Version 2 clocks	Version 1 clocks	Region A contains at least one class 6 or 7 clock with priority1 <128.
Version 2 clocks	Version 1 clocks.	Any stratum 1 or 2 clocks are contained in a single B region.
Version 1 clocks.	Version 2 clocks	Region B clocks all have priority1>128
Version 1 clocks.	Version 2 clocks	Region B clocks all have priority1>127 and there is a region A clock with preferred = TRUE.
Version 1 clocks.	Version 2 clocks	Any class 6 or 7 clocks and any clocks with priority1<128 are contained in a single B region.

19. Conformance

19.1 Conformance objective

The philosophy underlying the conformance requirements of Clause 19 is to:

- Raise the level of interoperability of systems built to this standard,
- Encourage the manufacture of PTP components with the broadest possible range of applicability,
- Provide opportunity for continued technical improvement and differentiation.

19.2 PTP conformance requirements

19.2.1 General conformance specification

Conformance requirements are specified in terms of nodes.

Nodes shall conform to all clauses of this standard with the exception of:

- Clauses specifically marked Optional, and
- For applications that distribute only frequency and do not require the measurement of the path delays, an alternate PTP profile may specify that the path delay mechanisms of 11.3 and 11.4 shall not be implemented or activated.

For each option implemented, the node shall conform to the clause specifying the option.

For applications that distribute only frequency and do not require the measurement of the path delays, an alternate PTP profile may specify that the path delay mechanisms of 11.3 and 11.4 need not be implemented or activated.

19.2.2 Transport conformance specification

A node that uses a transport protocol for which the mapping is defined in an annex of this standard shall conform to that annex.

The transport of PTP packets using a transport protocol for which there is no mapping defined in this standard shall be defined by a mapping defined and published by the standards organization, or its designee, with jurisdiction over the transport. The publication specifying this mapping shall be referenced by a PTP profile.

NOTE— The organization defining a transport mapping has to secure an enumeration value for the transport, see 7.4.1.

19.2.3 PTP profile conformance specification

A node claiming compliance shall specify at least one PTP profile to which it complies. One of the two default PTP profiles shall be used in the absence of a suitable alternate PTP profile. The default PTP profiles are specified in Annex J.

If a particular attribute or option is not specified in the selected PTP profile then the node shall conform to the value or choice specified in the default PTP profile that specifies the same path delay mechanism.

All PTP devices should support one of the default PTP profiles.

19.3 PTP profiles

19.3.1.1 General

The purpose of a PTP profile is to allow organizations to specify specific selections of attribute values and optional features of PTP that, when using the same transport protocol, inter-works and achieve a performance that meets the requirements of a particular application.

A PTP profile is a set of required options, prohibited options, and the ranges and defaults of configurable attributes. Profiles specifications shall be consistent with the specifications in subclauses 19.2.1 and 19.2.2.

19.3.1.2 PTP profile recommendations

A PTP profile should define:

— Which of the best master clock algorithm options, see 9.3.1, is to be implemented

— Which of the configuration management options, see 15.1.1, is to be implemented

— Which of the path delay mechanisms, delay request-response, see 11.3, or peer delay, see 11.4, is to be implemented.

— The range and default values of all PTP configurable attributes and data set members

— The transport mechanisms required, permitted, or prohibited.

— The node types required, permitted, or prohibited.

— The options required, permitted, or prohibited.

A PTP profile shall extend the standard only by:

a) The use of the TLV mechanism of 14.3.

b) The specification of an optional best master clock algorithm ,see 9.3.1.

c) The specification of an optional management mechanism, see 15.1.1.

d) The provisions of 19.2.2.

e) The provisions of 7.3.1.

19.3.2 Specific PTP profiles

A PTP profile may be developed by external organizations including:

a) A recognized standards organization with jurisdiction over the industry, e.g. IEC, IEEE, IETF, ANSI, ITU, or

b) An industry trade association or other similar organization recognized within the industry as having standards authority for the industry, or

c) Other organizations as appropriate.

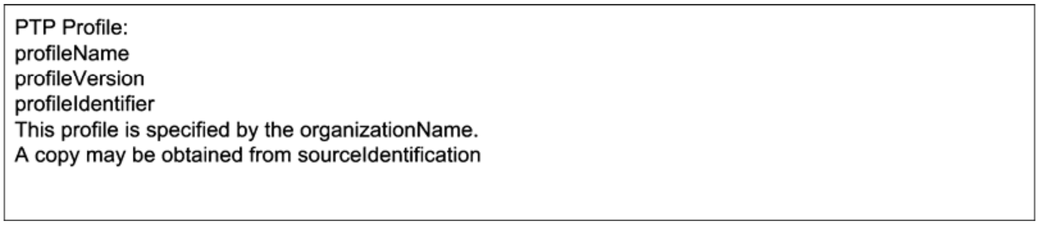
The PTP profile development organization should consult the Precise Networked Clock Synchronization Working Group of the IM/ST Committee for technical review.

19.3.3 PTP profile specifications

A PTP profile shall be identified by the following text printed at the beginning of the profile document as shown in Figure 37.

The items indicated in Figure 37 are defined as follows:

- profileName: This field shall be the text title of the profile as designated by the organization specifying the profile. See J.3.1 for an example.
- profileVersion: This field shall be the version of the profile as designated by the organization specifying the profile. The version designation shall consist of two fields: A primaryVersion (UInteger16) and a revisionNumber (UInteger8). The Profile Version shall be printed as "Version primaryVersion.revisionNumber"
- profileIdentifier: This field shall be a EUI-48. The OUI portion of the EUI-48 shall be owned by the organization specifying the profile. This organization shall ensure that the Profile Identifier is unique to each profile and version specified by the organization. The remaining octets of the EUI-48 shall be the primaryVersion and revisionNumber in that order. See J.3.1 for an example.
- organizationName: This field shall be the textual name of the organization specifying the profile and owning the OUI of the Profile Identifier.
- sourceIdentification: This field shall be a URL or regular mail address to which enquiries concerning the profile or requests for copies may be sent.



PTP Profile:
profileName
profileVersion
profileIdentifier
This profile is specified by the organizationName.
A copy may be obtained from sourceIdentification

Figure 37 Profile Print Form

Annex A
(informative)
Using PTP

A.1 Overview

PTP provides a simple methodology for accurately synchronizing clocks in a distributed system. When designing such a system the following questions need to be answered.

Physical layout issues:

— How physically dispersed are the clocks?

— What network technology is to be used?

Logical issues:

— Is the system a single collection of clocks, or are the clocks divided into logical groupings each with their own sense of time?

Component issues:

— How accurately do the clocks need to be synchronized?

— What is the source of time for the system? Should it be traceable to UTC?

Local implementation issues:

— How are timing requirements to be met?

— How do other applications sharing the communication network affect PTP?

— How do accuracy requirements affect the implementation?

— What are the design issues for local oscillators?

System implementation issues:

— How is the system partitioned?

— Which options are used?

— Which profiles are used?

Performance issues:

— How do network delays and fluctuations affect timing accuracy?

— How does clock oscillator stability affect timing accuracy?

Conformance testing issues:

— Features to aid in conformance and performance testing,

— Features to aid in calibrating device timing.

The following subclauses of this annex address each of these topics.

A.2 Physical layout

clocks communicate with each other over a network. Typically, the selection of the network technology is based on the primary application. PTP works on any packet-based system. PTP is designed to work in a multicast environment, although it is possible to design unicast PTP components and systems. Ethernet is an ideal network for implementing PTP, and the rest of this annex uses Ethernet as an example.

All networks have limitations on distance, number of allowed nodes, and traffic. If the clocks to be synchronized are dispersed beyond the range of the network technology, then the system should be designed as separate ‘islands of time’ with provision outside of PTP for synchronizing these islands.

For example, if the system consists of two compact sites separated by several miles, PTP can be used within each site, with site-to-site synchronization provided by another technology such as GPS.

Within a site, distance, traffic, and number of node issues are usually addressed by special network components. For Ethernet, localized nodes typically communicate via switches (i.e. bridges). For larger and more complex systems, routers are used to separate the system into regions using only bridges. In general, each level of separation using these devices introduces additional statistical delay and delay fluctuation in the message transmission times between nodes.

PTP is designed to minimize the effects of delay and delay fluctuation. To get the best PTP performance, the network topology should have as a constraint the minimization of the number of such separating devices between clocks with the most critical synchronization requirements.

Boundary clocks can be used to improve the performance across separations in the network defined by routers, or in place of ordinary bridges. Transparent clocks can also be used in place of ordinary bridges particularly in situations where many devices are connected in a linear topology.

Bridges not implementing PTP may introduce considerable timing jitter and path asymmetry. While such bridges may be included in a system implementing PTP, these bridges should not be used unless timing errors introduced by their jitter and path asymmetry are tolerable for the application, or can be reduced by an appropriate filtering algorithm.

A.3 Logical layout

Most applications consist of a single set of clocks to be synchronized. For this case, all the clocks can be placed in a single domain. If the default values specified in this standard and the applicable conformant PTP profile are used, then generally no configuration of the clocks is necessary.

If the application requires several groups of clocks, with each group maintaining a different self-consistent time base, then one of two solutions may be used:

- If the rest of the application is segmented into the same groups, it may be possible to use separate non-communicating networks in which case each group can use the default domain. Network routers are often used for this purpose.
- If the groups have to share a common network, then each group may be assigned to a different domain. This logically divides the clocks as desired. Depending on the mapping to the underlying physical addressing of the network, the processing load on each clock may or may not be affected.

With the exception of the assignment of PTP nodes to a domain, PTP defines an administration free system in the default case. Within a domain, PTP nodes may be added or removed without any requirement for modification of address tables, etc. provided components use the recommended multicast communication model. Addition or removal of PTP nodes may cause a different clock to become the grandmaster clock in the system. This may cause a transient in the time base as the system automatically recalibrates for the new delay patterns to the new grandmaster clock.

This standard provides several configuration options for users that require more control over the selection of master clocks, or over different timing and other attributes that govern system performance. For example, the use of the priority1 attribute allows system designers to designate up to 254 devices in a priority order for grandmaster clock selection.

A.4 Component issues

The primary issue in the selection of PTP system components is the required synchronization accuracy.

- clocks should be selected that are designed to support those features of the protocol required for a given accuracy. clocks with the highest inherent accuracy should support the use of the Follow_Up messages.
- Network components and physical design decisions also affect the accuracy as outlined in the previous clauses.

Properly designed Ethernet PTP systems can readily achieve sub-microsecond accuracy.

A second issue is the technique for establishing the PTP system epoch. In every domain, the epoch is defined by the grandmaster clock that is selected according to the best master clock algorithm.

If TAI-traceable time is a requirement, then the grandmaster clock must maintain a PTP time base.

If the lowest value of clockClass is 6, 7, 52 or 187 for all clocks in a domain, the time base is PTP. From the PTP timescale, UTC can be computed using the value of UTC Offset distributed by PTP. Such systems may or may not maintain the epoch after a power outage, see 7.6.2.4.

If the lowest value of clockClass is 13, 14, 58, 193, or 216 or greater for all clocks in a domain, the time base is either ARB or a time base established by the user, see 7.6.2.4. Such systems may or may not maintain the epoch after a power outage.

A master clock may fail in such a way that its time or frequency become incorrect. Detection of this problem and recovery from it are outside the scope of this standard. Some information such as the parent statistics maintained in the parentDS data set is available to aid in detecting a "false-ticking" master. Implementers are advised to consider information from as many clocks as possible, and to weigh the information from each clock according to that clock's inherent stability.

The DISABLE_PORT management message is available to aid in recovering from a false-ticking master. Note that disabling or demoting a master has side effects (especially if it is a boundary clock), so the decision to do that may depend on factors besides its timekeeping quality. That decision is outside the scope of this standard.

A.5 Local implementation issues

A.5.1 General

A.5 provides some guidelines for implementers of PTP ordinary, boundary and transparent clocks. While not in the scope of this standard, implementations should take care that services built on top of clocks synchronized via PTP (or any other protocol) do not degrade the accuracy.

A.5.2 Timing issues

Implementations must meet the message processing and timing requirements and must also meet whatever timing requirements are needed to operate any servomechanism that synchronizes the local clock based on information in PTP messages.

Implementations must ensure that adequate computing and memory resources are available to meet these requirements. Implementations must also ensure that the resources needed by the PTP implementation have adequate priority over other applications sharing these resources to meet the PTP and servomechanism timing requirements. PTP tasks should be assigned the highest priority in an implementation, similar to priorities assigned to the protocol stack and other operating system resources.

PTP implementations normally require resources for a short time in every sync interval. The selection of the sync interval for a system must be consistent with the available resources in all system components. The use of network resources by other applications can affect PTP accuracy as discussed in A.5.3.

A.5.3 Accuracy issues

A.5.3.1 General

The achievable accuracy of a PTP system is limited by the following:

— The delay fluctuation in the protocol stacks of clocks,

— The delay fluctuation in network components,

— Timestamping accuracy, and

— Stability issues.

A.5.3.2 Protocol stack delay fluctuation

The simplest implementations of PTP operate as ordinary applications at the top of the network protocol stack. Timestamps are generated at the application level. Protocol stack delay fluctuation cause errors in these timestamps. These errors are typically in the hundred microseconds to milliseconds range depending on the operating system.

Implementations may generate timestamps at the interrupt level rather than at the application level. In this case, delay fluctuation typically can be reduced to tens of microseconds depending on other use of interrupts by other applications.

The greatest reduction in errors due to protocol stack delay fluctuation is achieved with hardware assist techniques that generate timestamps at the physical layer of the protocol stack. Delay fluctuation at this point is typically in the nanoseconds range. For example, in an Ethernet system these errors result from the phase lock characteristics of the PHY chips that recover the clock and data synchronization from the incoming data streams. The effect of this delay fluctuation may be reduced by suitable design of the clock servo algorithms.

A.5.3.3 Network component delay fluctuation

Network components introduce fluctuation in the propagation time of messages. This directly affects the accuracy of the offsetFromMaster and meanPathDelay values of the currentDS data set.

Network bridges and routers are subject to store and forward delay fluctuation. Typical Ethernet bridges have input and output buffers communicating over a very high-speed back plane or switch fabric. Each port typically connects directly to an end device or another Ethernet bridge. The dominant contribution to delay fluctuation arises from the output buffering and queuing. If the output subnet is always available, this delay fluctuation is typically in the nanoseconds range and reducible by averaging techniques. Intensive traffic directed at a node containing a clock may cause increased delay fluctuation due to this output buffering. This increased delay fluctuation is much more difficult to reduce. The proper design of PTP systems must recognize this effect and take measures to reduce the impact.

Most bridges and routers support traffic prioritization. High priority traffic suffers less fluctuation in propagation time. PTP event messages should be sent with high priority compared to other data whenever possible. See Annexes D ─ I for specific priority recommendations for each transport protocol.

A.5.4 Timestamp accuracy

The resolution of the clock generating the timestamps required by PTP must be consistent with the desired accuracy. Note that this resolution contributes to the PTP variance, see 7.6.3.

A.5.5 Stability issues

As noted in previous subclauses of this annex, the delay fluctuation introduced into the computation of the offsetFromMaster and meanPathDelay members of the currentDS data set may be reduced by suitable design of any synchronization servo algorithms of the local clock. Engineering trade-offs must be made between the averaging times (number of samples) and the responsiveness to effects other than delay fluctuation, such as oscillator stability.

The fundamental time stability of the local clock must be consistent with the required sync interval and accuracy specifications. The algorithms used to reduce delay fluctuation do not correct for drifts of the local clocks during time intervals small compared with the averaging intervals of the algorithms. Servos cannot correct for random drifts occurring within a sync interval.

At high accuracy the specifications on the stability of the local oscillators driving the local clock can be quite difficult to meet. The trade-off is between cost and stability. Local oscillators typically are quartz crystals. The frequency of quartz crystals typically drift due to thermal, mechanical and aging effects. Of these, thermal effects are the most difficult to deal with in most applications.

For example, a typical thermal specification for uncompensated crystals is 1 PPM per degree Celsius. A 1 degree temperature rise over a sync interval of 2 seconds produces an error on the order of 2 microseconds. Accuracies in the tens of nanosecond range therefore imply that some combination of better thermal specifications on the crystal, reduced sync interval, and better thermal management be used to reduce the thermal drift by two orders of magnitude.

PTP allows sync intervals to be reduced to a fraction of a second depending on the PTP profile selected, with the corresponding increase in computation and network bandwidth requirements.

Thermal specifications on crystals become increasingly expensive below 1 PPM/degree. Control of the thermal environment must be carefully managed, particularly in high accuracy implementations. Very long averaging times typically require oven controlled crystals or the use of more stable oscillators. Thermal drift during the short intervals and averaging times typical of PTP systems can often be managed by attention to heat dissipation in surrounding devices, cooling patterns within the node, increasing the thermal mass of the oscillator, and similar techniques. See [M24] for a thorough discussion of clock characterization.

A.6 System Implementation Issues

A PTP system is the collection of PTP components that operate together to meet the requirements of an application. An interoperable PTP system is one where the protocol operates as specified in this standard, the selection and configuration of nodes is such that the protocol is successful in constructing a master-slave timing hierarchy, and the nodes with a port in the SLAVE state are able to synchronize to a node with a port in the MASTER state. An optimal PTP system is one that is interoperable, manageable, and meets the synchronization requirements of the application. Ensuring that a system built with conformant PTP nodes is optimal is an issue for the system integrator and no guidance for optimal systems is provided below. The following recommendations facilitate the construction of interoperable systems:

- Use a single transport throughout the domain, or divide the domain into regions each of which uses a single transport. Regions are connected using boundary clocks.
- Use a single management approach throughout the system. Either the management message mechanism of this standard, or an alternate management mechanism specified in a PTP profile are acceptable.
- Use the same choice of best master clock algorithm throughout the domain. There is no assurance that regions of a domain implementing different choices of best master clock algorithm can be made to interoperate, even when connected by a boundary clock. Use either the best master clock algorithm defined in this standard or an alternate specified in a PTP profile.

- Use the same selection of state configuration options, Clause 17, throughout the domain. If state configuration options are used, it is the responsibility of the system integrator to ensure that the selected configuration produces an interoperable system. There is no assurance that regions of a domain implementing different choices of configuration options and configuration can be made to interoperate, even when connected by a boundary clock.

- Use a single path delay mechanism (see 11.3 and 11.4) throughout the domain, or divide the domain into regions each of which uses a single path delay mechanism. Regions are connected using one or more boundary clocks.

- Use an interoperable set of attribute and configurable data set values throughout the domain or divide the domain into regions each of which uses a single such interoperable set. Regions are connected using one or more boundary clocks.

- Use the same default value for each attribute and configurable data set member on all nodes in the system.

- Use the same required maximum and required minimum range values for each attribute on all nodes in the system.

- Some options must be present and active on every node in a system for the option to work as designed and to avoid interoperability problems. An example is the experimental security option. Other options are effective on the subset of nodes implementing them, even if other nodes in the system do not support the option. Furthermore the presence of such nodes does not interfere with nodes not implementing the option. An example is the unicast option.

- Use only nodes implementing the same PTP profile throughout the domain, or divide the domain into regions each of which uses the same PTP profile. Regions are connected using a boundary clock capable of resolving the PTP profile differences. There is no assurance that the specifications of two PTP profiles admit to the design of a boundary clock that resolves the differences. For example, it is not possible to ensure that regions using self-configuration with the best master clock algorithm of this standard can interoperate with regions that use configuration of the master-slave hierarchy.

- Use only nodes implementing the same version of this standard throughout the domain, or divide the domain into regions each of which uses a same version (version 1, version 2, or a future version). Connect these regions using a boundary clock.

A.7 Performance

The following requirements should be met to achieve optimal clock synchronization performance:

a) Network delay between master and slave should be symmetric.

b) A clock may contain asymmetric delays in its timestamping mechanism or protocol path. If these asymmetries are not negligible, they should be correctly accounted for, see 11.6.

c) Network delay between master and slave should be constant over the time interval between Delay_Req messages.

d) Delay fluctuation due to network components and due to the protocol stack within clocks should be reduced by two techniques:

   1) The timestamps used in PTP should be generated as close to the physical layer as practical for a given clock implementation. In cases where the most accurate timestamps can be generated only after a message has actually been transmitted, the actual value is communicated in the Follow_Up message from the master or the Pdelay_Resp_Follow_Up message from the peer-to-peer transparent clock.

   NOTE—See [M7, M12, M4] for mechanisms to aid in generating these timestamps.

2) Remaining delay fluctuation introduced by the protocol stack and by network components not isolated by a boundary or transparent clock can be reduced by averaging. The averaging algorithms are outside the scope of this standard.

e) The computing power of clocks implementing the protocol must be great enough, and the number of clocks must be small enough, to meet the timing constraints. Implementers of boundary and ordinary clocks, for example, need to consider the resources required to process Delay_Req messages from slaves communicating with the node. The inability to process these messages due to resource limitations may lead to deterioration in the synchronization performance due to missed measurements of the path delays. Users need to be aware of this limitation when selecting nodes and designing their systems

f) The inherent stability and precision of a clock's oscillator must be adequate, see A.5.4.

A.8 Recommendations to aid in conformance testing

To aid in:

- Testing the performance of a PTP system,
- Calibrating PTP devices, and
- Verifying conformance,

all PTP ordinary and boundary clocks should provide a 1 pulse per second (PPS) signal with the rising edge of the pulse coincident with each increment in the seconds field of the clock. If implemented and not coincident then the device specifications should include the time offset of the 1 PPS signal from the seconds increment event time. This signal may be an accessible internal test point and need not be visible as an external output of the device in which the clock is embedded, e.g. a sensor.

A.9 Recommendation for implementations in unicast networks or networks with non-PTP bridges and routers.

A.9.1 General

PTP masters and slaves will be introduced into networks where bridges and routers do not support the PTP standard. Further, many networks do not support multicast. clocks can be introduced in these networks by using clocks that are able to communicate with each other over such networks.

The unicast communication model can be used to overcome many of these problems. Clause 7.3.1 allows the use of a unicast model provided that the behavior of the protocol is preserved.

A.9 describes issues that must be specified in an alternate PTP profile when using a unicast communication model, to produce an implementation that is likely to work in such networks while satisfying a wide range of timing requirements. Some wide-area network requirements, such as security and resilience are out of scope of this discussion.

A.9.2 Boundary clocks and transparent clocks in a unicast model

In the multicast model, ordinary and boundary clocks automatically create a synchronization hierarchy without prior knowledge of network topology. It is guaranteed that, except for management messages, boundary clocks terminate all PTP messages. Further, if only PTP bridges, routers, transparent clocks and boundary clocks are present, the messages used by the peer delay mechanism are guaranteed to terminate in the neighbor peer-to-peer clock thus ensuring correct operation of the mechanism. In the unicast model, however, the above conditions do not hold.

To preserve the protocol behavior, the following functions must be preserved when using the unicast model:

- The correct creation of synchronization hierarchy,
- The correct exchange of timing messages and associated general messages needed for synchronization,
- The correct operation of the peer delay or delay request-response mechanisms for determining path latency, and
- A management mechanism for configuring the clocks.

One way to achieve these functions is by requiring that all ordinary, boundary, and peer-to-peer transparent clocks are configured in advance, with the unicast protocol addresses of the neighboring clocks visible from each port. As one exception to the previous sentence, the addresses of slave-only clocks using the delay request-response mechanism do not need to be preconfigured in other clocks if unicast option 16.1 is used. If the peer delay mechanism is to be used, the configuration must ensure that only a single peer-to-peer clock is visible from each port, see 11.4.4.

Clock ports may be neighbors even when there are bridges, routers, or transparent clocks between the ports. Clock ports are not neighbors if a boundary clock is between the ports. In the event of a network reconfiguration, the neighbor relationships may change, in which case two ports may communicate in unicast across a boundary clock. If a mechanism for learning topology changes is available, clocks can stop all unicast communications between non-neighbors, leading to an optimized synchronization hierarchy and better utilization of the network resources. If such a learning mechanism is not available then, depending on the network topology, it may be advisable to use end-to-end transparent clocks instead of boundary clocks or peer-to-peer transparent clocks. In all cases the implemention has to provide a mechanism to break forwarding loops for achieving correct operation of the protocol.

A.9.3 Unicast options

The configuration options of Clause 17 can be used to configure each port with the needed unicast protocol addresses.

The unicast option of 16.1 can be used to establish unicast communications for Announce, Sync, Delay_Resp, and Pdelay_Resp messages, and any associated general messages.

Alternatively, unicast contracts between two nodes can be created using a management procedure. These contracts consist of the unicast address information and respective specified packet rates for Sync, Announce, and Delay_Req messages.

The path trace option of 16.2 can be used in defining a mechanism for breaking forwarding loops.

Since the management mechanism of 15.2 depends on the use of the multicast model and forwarding by boundary clocks, an alternative management mechanism for configuring the clocks must be specified as permitted in 15.1.1.

A.9.4 Unicast Conformance

A.9.4.1 General

Subclause 19.2.3 specifies that to claim conformance, a node must comply with a PTP profile in addition to conforming to the PTP standard. This profile must specify any differences from the specifications of the default PTP profiles of Annex J. A.9.4.2 contains examples of some of the specifications that are needed to implement a unicast model to meet the requirements discussed in A.9.1 and A.9.2. Not discussed are possible alternate best master clock algorithms and an alternate unicast-based management mechanism.

A.9.4.2 PTP options and attribute values

The unicast options defined in 16.1 and 17.5 need to be supported and operational by default. All other options of Clause 16 and Clause 17 are permitted but need to be inactive by default.

The unicast communication model is used by default as permitted by 7.3.1. If the multicast communication model is also implemented it must be inactive by default. Multicast communication is a recommended option for exploiting future multicast support in these networks and for allowing interoperability with equipment supporting the PTP default profiles.

The timing of unicast messages is determined by the values of the logInterMessagePeriod field in the unicast negotiation REQUEST_UNICAST_TRANSMISSION TLV.

Suggested values for the logInterMessagePeriod field of the REQUEST_UNICAST_TRANSMISSION TLV are:

— For requesting unicast Announce messages: The default initialization value of logInterMessagePeriod is 1 (once every two seconds). The configurable range is -3 (8 per second) to 3 (once every 8 seconds).

— For requesting unicast Sync messages: The default initialization value of logInterMessagePeriod is -4 (16 per second). The configurable range is -7 (128 per second) to 1 (once every 2 seconds).

— For requesting unicast Delay_Resp messages: The default initialization value of logInterMessagePeriod is -4 (16 per second). The configurable range is -7 (128 per second) to 6 (once every 64 seconds).

The durationField value in each REQUEST_UNICAST_TRANSMISSION TLV has a default initialization value of 300 (300 seconds) and a configurable range of 10 to 1000.

The maintenance and configuration of these default and configuration range values is implementation-specific.

In implementing the GRANT_UNICAST_TRANSMISSION TLV mechanism, the granted values should be the same as requested in the received REQUEST_UNICAST_TRANSMISSION TLV as long as the requests are in the configurable range.

NOTE—Since the transport may be unreliable, the requesting port should repeat the request after an implementation-specific timeout if no grant TLV has been received. For receiving continuous service, a requester should reissue a request in advance of the end of the grant period. The recommended advance should include sufficient margin for reissuing the request at least two more times if no grant is received.

The values of announceReceiptTimeout, priority1, priority2, slaveOnly, primaryDomain, and τ are identical to those specified in J.3.

The physical requirements are identical to those specified in J.3.

Annex B
(informative)
Timescales and epochs in PTP

B.1 General considerations

A more detailed discussion of many of the topics in this annex may be found in [M1], [M18], [M17], [M16] and [M6] and [M2].

Within a domain, the characteristics of the time available are determined by the grandmaster clock of the domain. The grandmaster determines:

— The rate at which time advances. The grandmaster frequency accuracy is measured by how well a time interval determined between any two events, as measured by the grandmaster, corresponds to the same interval measured using a clock consistent with the internationally defined second. The internationally defined second, SI, is the measure of time defining the TAI timescale maintained by the Bureau International des Poids et Mesures near Paris.

— The origin, or epoch, of the timescale.

The possible timescales and epochs available for use by the PTP grandmaster clock are:

— PTP timescale: Indicated by a ptpTimescale value of TRUE. The epoch is the PTP epoch.

— ARB timescale: Indicated by a ptpTimescale value of FALSE. The epoch is specific to the implementation.

B.2 UTC, TAI and the PTP epoch

TAI and UTC are international standards for time based on the SI second as realized on the rotating geoid. UTC is implemented by a suite of atomic clocks and forms the timekeeping basis for other timescales in common use.

UTC is the timescale of most engineering and commercial interest. The UTC representation is specified by ISO 8601 as YYYY-MM-DD for the date and hh:mm:ss for the time in each day. The rate at which UTC time advances is identical to the rate of TAI. UTC time differs from the TAI time by a constant offset. This offset is modified on occasion by adding or subtracting leap seconds. TAI advances continuously while UTC experiences a discontinuity with each leap second introduction.

Starting on 0 hours on 1 January 1972 UTC (Modified Julian Day (MJD) 41,317.0)[6], the world's standard time systems began the implementation of leap seconds to allow only integral second correction between UTC Seconds and TAI, both of which are expressed in days, hours, minutes and seconds. On this date TAI − UTC was 10 seconds. Prior to 1 January 1972, corrections to the offset between UTC and TAI were made in fractions of a second.

Leap second corrections, which are applied to UTC but not to TAI, are made preferably following second 23:59:59 of the last day of June or December. The first such correction, a single positive leap second correction, was made following 23:59:59 on 30 June 1972 UTC, and UTC was 11 seconds behind TAI following that instant.

[6] The Julian Date (JD) is the Julian Day number (JDN) followed by the fraction of the day elapsed since the preceding Greenwich mean noon. The JDN is a day count with the origin, JD = 0, at Greenwich mean noon on 1 January 4713 BC. The Modified Julian Date, MJD, is the Julian Date less 2 400 000.5 which shifts the origin to midnight on 17 November 1858. For example: at 0 hours on 1 January 1900, JD = 2 415 020.5, and MJD = 15020.

NOTE— As of 0 hours 1 January 2006 UTC, TAI ─ UTC = +33 seconds.

In computer systems the common POSIX based time conversion algorithms are typically used to produce the correct ISO 8601:2004 printed representations for both TAI and UTC. UTC is behind TAI by the number of leap seconds.

The PTP epoch is set such that a direct application of the POSIX algorithm to a PTP timescale timestamp converts the PTP timestamp to the ISO 8601:2004 printed representation of TAI. PTP also distributes the current number of leap seconds <currentLeapSeconds> in the utc_offset field of Announce messages. Subtracting <currentLeapSeconds> from a PTP timestamp prior to applying the POSIX algorithm results in the ISO 8601:2004 printed representation of UTC. Conversely, applying the inverse POSIX algorithm and adding <currentLeapSeconds> converts from the ISO 8601:2004 printed form of UTC to the form required to generate a PTP timestamp.

For example, the POSIX algorithm applied to a PTP timestamp value of 8 seconds yields 00:00:08 1970:01:01 (eight seconds after midnight on 1 January 1970 TAI). At this time the value of utc_offset was approximately 8 seconds. Subtracting 8 seconds from the PTP timestamp value 8, yields a value of 0. The POSIX algorithm applied to the value 0 yields 00:00:00 1970:01:01 (the beginning of the first second of 1 January 1970 UTC), which is the expected UTC value. Note that from 7.2.2 the PTP epoch is approximately 8 seconds before this time. Thus a direct application of the POSIX algorithms to a PTP timescale timestamp yields the print form of TAI for that time.

B.3 Standard time sources

There are two standard time sources of particular interest in implementing PTP systems for which UTC traceable time is required by the application.

The first time source is the set of systems implementing the NTP protocol, widely used in synchronizing computer systems within a campus and around the world. A set of NTP servers, to which NTP clients synchronize, is maintained. These servers themselves are synchronized to timeservers traceable to international standards. UTC time precision from NTP systems is usually in the millisecond range. NTP provides the current time, the current number of leap seconds (supported only in NTP version 4), and warning flags marking the introduction of a leap second correction, which is inserted at the end of the current UTC day. NTP does not correct the number of NTP seconds since the NTP epoch whenever a leap second correction is made. (In other words, the NTP clock effectively stops during a leap second, and the time interval occupied by a leap second is effectively “forgotten” once it has been inserted.) The NTP epoch is 0 hours on 1 January 1900. NTP was set at 0 hours on 1 January 1972 to 2 272 060 800.0, to agree with UTC. Currently, NTP represents seconds as a 32 bit unsigned integer. NTP therefore rolls over every 2^{32}seconds ≈ 136 years with the first such rollover occurring in approximately the year 2036.

The second system of interest is the global positioning satellite system, GPS, maintained by the U.S. Department of Defense. UTC time precision from the GPS system is usually in the 10-100 ns range. GPS system transmissions represent the time as {GPS Weeks, GPS SecondsInLastWeek}, i.e. the number of weeks since the GPS epoch and the number of seconds since the beginning of the current week. From this, GPS Seconds, i.e. the number of seconds since the GPS epoch can be computed. GPS provides the current time, the current number of leap seconds, and warning flags marking the introduction of a leap second correction. From GPS time, UTC, and TAI times may be computed using the information contained in the GPS transmissions. The GPS epoch began at 0 hours on 6 January 1980 (MJD 44 244). GPS weeks are represented in the satellite transmissions modulo 1024 weeks = 19.7 years. The first such rollover occurred between the weeks of 15 August and 22 August 1999. Many, but not all, commercial systems are believed to have correctly managed this rollover.

Either of these systems may be conveniently used to provide time to a clockClass 6 clock. Relationships between the timescales discussed and examples of times in each system for interesting instants are given in

Table 111. In Table 111, PTP Seconds refers to the seconds portion of the time distributed by the PTP timescale and as noted is referenced to 1 January 1970 TAI.

Table 111: Relationships between timescales

From	To	Formula
NTP Seconds	PTP Seconds	PTP Seconds = NTP Seconds − 2 208 988 800 + currentUTCOffset
PTP Seconds	NTP Seconds	NTP Seconds = PTP Seconds + 2 208 988 800 − currentUTCOffset
GPS Seconds = (GPS Weeks × 7 × 86400) + GPSSecondsInLastWeek (GPS week number must include 1024 × number of rollovers)	PTP Seconds	PTP Seconds = GPS Seconds + 315 964 819
PTP Seconds	GPS Seconds	GPS Seconds = PTP Seconds − 315 964 819

Annex C
(informative)
Examples of residence and asymmetry corrections.

C.1 General

Annex C provides several examples illustrating the exchange of timing messages and the application of the corrections for residence time, path delay and asymmetry in transparent clocks.

The transit times of event messages between clocks, and the residence time within transparent clocks are shown and are not assumed to be the same in both directions.

In each case the figures only include the critical fields of the timing messages. The clock at the top of the figure is always assumed to be the master, or in the case of measurement of link delay the peer-to-peer responder.

The times shown in the boxes representing each of the clocks are expressed as local time in the device. Times are represented in the figures as seconds:nanoseconds.fractional nanoseconds. For example, 144:7.3 is 144.0000000073 seconds. Bear in mind that the timestamp fields of messages cannot hold fractional nanoseconds. Fractional nanoseconds can only be carried in the correctionField, which in the figures is expressed as nanoseconds.fractional nanoseconds.

The relationship of the clocks in the devices is given so that times relative to the master can be computed if desired. This relation is expressed in the form time = Tm + offset, where Tm is the master time and offset is the offset between the respective clock and the respective master. The times shown in the message boxes are the times that would be entered by the clock transmitting the message. For example, in Figure 38 the Sync egress and ingress timestamps for the Sync message as it travels from the master, through the end-to-end transparent clock to the slave are given as 144:7.3, (144:7.3 + 0.65 + 100.3), (144:7.3 +0.65 +100.3 +207.4), and (144:7.3 + 0.65 +207.4 +0.5 +25.2) = 144:241.05. The first is the egress timestamp t_1 leaving the master clock. The second, the ingress at the transparent clock, is computed by summing t_1, 0.65, which is the transit time between the master and the transparent clock, and 100.3, which is the assumed offset between the local clocks in the master and the transparent clock. This computation results in the ingress timestamp as generated by the transparent clock. The computation of the other terms is similar. The ingress and egress timestamps for the Delay_Req message are computed in the same way. Note that the egress time from the slave clock is given as 144:651.1 relative to the slave clock, which is 144:651.1 -25.2 = 144:625.9 relative to the master clock.

The computation in the slave or requestor clocks illustrate the process of combining timestamps, correctionFields, and asymmetry corrections to compute offsets and path delays. These values are always compared to the assumed values used in the figures to illustrate the operation of the protocol.

The lettered call-outs in the figure are referenced to specific clauses in the standard to show the principal point being illustrated.

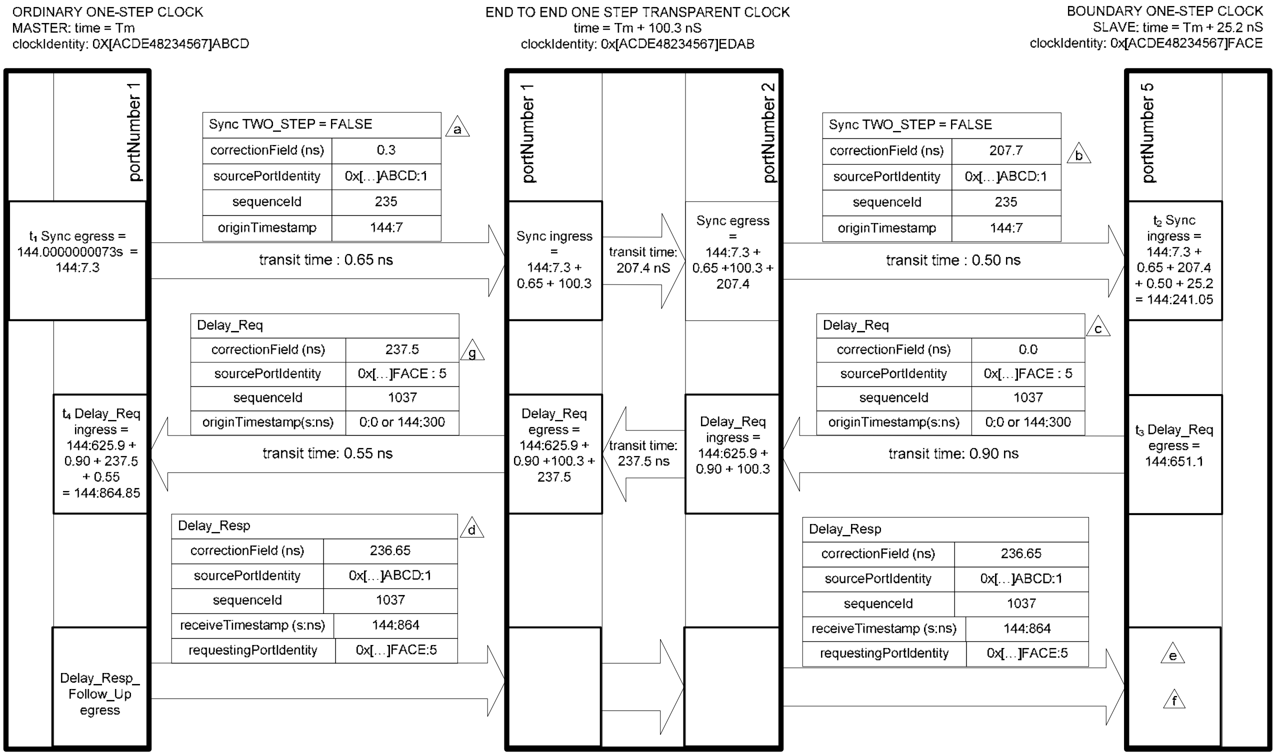
With the exception of Figure 38, all examples show the corrections for both residence time and asymmetry.

Note-Hex numbers are represented by 0x in the figures in Annex C for clarity.

C.2 Computations using the delay request-response mechanism

C.2.1 Master, end-to-end transparent, and slave all one-step clocks showing residence time corrections.

Figure 38 illustrates the measurement and computation of meanPathDelay and offsetFromMaster in a system composed of two ordinary clocks, one a master and one a slave, separated by an end-to-end transparent clock. All are one-step clocks. In Figure 38 no asymmetry corrections are illustrated.


ORDINARY ONE-STEP CLOCK
MASTER: time = Tm
clockIdentity: 0X[ACDE48234567]ABCD
END TO END ONE STEP TRANSPARENT CLOCK
time = Tm + 100.3 nS
clockIdentity: 0x[ACDE48234567]EDAB
BOUNDARY ONE-STEP CLOCK
SLAVE: time = Tm + 25.2 nS
clockIdentity: 0x[ACDE48234567]FACE
portNumber 1
portNumber 1
portNumber 2
portNumber 5
a
b
c
d
e
f
g
t₁ Sync egress = 144.0000000073s = 144:7.3
transit time : 0.65 ns
Sync ingress = 144:7.3 + 0.65 + 100.3
transit time: 207.4 nS
Sync egress = 144:7.3 + 0.65 +100.3 + 207.4
transit time : 0.50 ns
t₂ Sync ingress = 144:7.3 + 0.65 + 207.4 + 0.50 + 25.2 = 144:241.05
t₄ Delay_Req ingress = 144:625.9 + 0.90 + 237.5 + 0.55 = 144:864.85
transit time: 0.55 ns
Delay_Req egress = 144:625.9 + 0.90 +100.3 + 237.5
transit time: 237.5 ns
Delay_Req ingress = 144:625.9 + 0.90 + 100.3
transit time: 0.90 ns
t₃ Delay_Req egress = 144:651.1
Delay_Resp_ Follow_Up egress

Sync TWO_STEP = FALSE	
correctionField (ns)	0.3
sourcePortIdentity	0x[…]ABCD:1
sequenceId	235
originTimestamp	144:7

Sync TWO_STEP = FALSE	
correctionField (ns)	207.7
sourcePortIdentity	0x[…]ABCD:1
sequenceId	235
originTimestamp	144:7

Delay_Req	
correctionField (ns)	237.5
sourcePortIdentity	0x[…]FACE : 5
sequenceId	1037
originTimestamp(s:ns)	0:0 or 144:300

Delay_Req	
correctionField (ns)	0.0
sourcePortIdentity	0x[…]FACE : 5
sequenceId	1037
originTimestamp(s:ns)	0:0 or 144:300

Delay_Resp	
correctionField (ns)	236.65
sourcePortIdentity	0x[…]ABCD:1
sequenceId	1037
receiveTimestamp (s:ns)	144:864
requestingPortIdentity	0x[…]FACE:5

Delay_Resp	
correctionField (ns)	236.65
sourcePortIdentity	0x[…]ABCD:1
sequenceId	1037
receiveTimestamp (s:ns)	144:864
requestingPortIdentity	0x[…]FACE:5

e mean_path_delay = [(144:241.05 – 144:651.1) + (144:864 – 144:7) – 207.7 – 236.65]/2 = 1.3 = [0.5 + 0.9 + 0.65 + 0.55]/2

f offset_from_master = 144:241.05 – 144:7 – 1.3 – 207.7 = 25.05 vs actual 25.2. error -0.15 = 0.65 + 0.5 – 1.3 due to uncorrected path symmetry

Figure 38: Master, end-to-end, and slave one-step clocks- no asymmetry correction

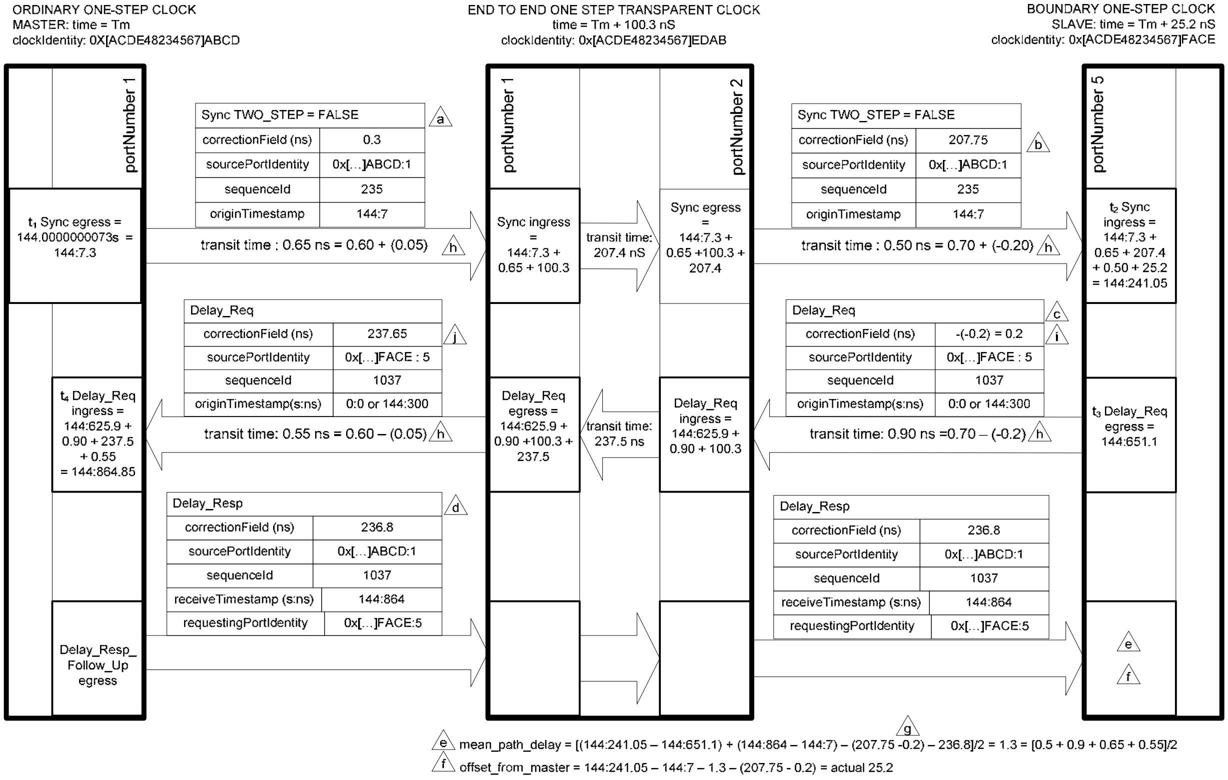
The interpretations of key values for Figure 38 are given in Table 112.

Table 112: Interpretation of Figure 38 key values

Key	Reference	Comments
a	9.5.9.3 & 11.3.2	Sum of timestamp and correctionField is t_1
b	11.5.2.1	Residence time, 207.4, has been added to the correctionField
c	11.3.2	Timestamp set to 0 or an estimate, 144:300, of the egress timestamp. The correctionField is set to 0.
d	11.3.2	Note that requestingPortIdentity and sequenceId are those of the slave clock. The receiveTimestamp is t_4 excluding fractional nanoseconds. The correctionField is the correctionField from the Delay_Req message (which was incremented in the transparent clock by the residence time, see 11.5.3.2) MINUS the fractional nanoseconds portion of t_4 (0.85)
e	11.3.2	The first term is the difference (t_2 – t_3). The second term is the difference in the receive and origin timestamps. The last two terms are the correction fields. Note that the computed meanPathDelay matches the actual assumed mean path delay from the figure.
f	11.2	The terms in order are t_2, originTimestamp meanPathDelay and Sync correctionField. Note that the computed offset is in error by -0.15 due to the uncorrected asymmetry in the transit times.
g	11.5.3.2	The residence time, 237.5, has been added to the correctionField by the transparent clock.

C.2.2 Master, end-to-end transparent, and slave all one-step clocks showing residence time and asymmetry computations

Figure 39 shows the same set of clocks but in this figure the asymmetry corrections are made.


ORDINARY ONE-STEP CLOCK
MASTER: time = Tm
clockIdentity: 0X[ACDE48234567]ABCD
END TO END ONE STEP TRANSPARENT CLOCK
time = Tm + 100.3 nS
clockIdentity: 0x[ACDE48234567]EDAB
BOUNDARY ONE-STEP CLOCK
SLAVE: time = Tm + 25.2 nS
clockIdentity: 0x[ACDE48234567]FACE
portNumber 1
portNumber 1
portNumber 2
portNumber 5
t₁ Sync egress = 144.0000000073s = 144:7.3
Sync ingress = 144:7.3 + 0.65 + 100.3
transit time: 207.4 nS
Sync egress = 144:7.3 + 0.65 +100.3 + 207.4
t₂ Sync ingress = 144:7.3 + 0.65 + 207.4 + 0.50 + 25.2 = 144:241.05
transit time : 0.65 ns = 0.60 + (0.05)
h
transit time : 0.50 ns = 0.70 + (-0.20)
h
t₄ Delay_Req ingress = 144:625.9 + 0.90 + 237.5 + 0.55 = 144:864.85
Delay_Req egress = 144:625.9 + 0.90 +100.3 + 237.5
transit time: 237.5 ns
Delay_Req ingress = 144:625.9 + 0.90 + 100.3
t₃ Delay_Req egress = 144:651.1
transit time: 0.55 ns = 0.60 – (0.05)
h
transit time: 0.90 ns =0.70 – (-0.2)
h
Delay_Resp_ Follow_Up egress
a
b
c
i
j
d
e
f
g

Sync TWO_STEP = FALSE	
correctionField (ns)	0.3
sourcePortIdentity	0x[…]ABCD:1
sequenceId	235
originTimestamp	144:7

Sync TWO_STEP = FALSE	
correctionField (ns)	207.75
sourcePortIdentity	0x[…]ABCD:1
sequenceId	235
originTimestamp	144:7

Delay_Req	
correctionField (ns)	237.65
sourcePortIdentity	0x[…]FACE : 5
sequenceId	1037
originTimestamp(s:ns)	0:0 or 144:300

Delay_Req	
correctionField (ns)	-(-0.2) = 0.2
sourcePortIdentity	0x[…]FACE : 5
sequenceId	1037
originTimestamp(s:ns)	0:0 or 144:300

Delay_Resp	
correctionField (ns)	236.8
sourcePortIdentity	0x[…]ABCD:1
sequenceId	1037
receiveTimestamp (s:ns)	144:864
requestingPortIdentity	0x[…]FACE:5

Delay_Resp	
correctionField (ns)	236.8
sourcePortIdentity	0x[…]ABCD:1
sequenceId	1037
receiveTimestamp (s:ns)	144:864
requestingPortIdentity	0x[…]FACE:5

e mean_path_delay = [(144:241.05 – 144:651.1) + (144:864 – 144:7) – (207.75 -0.2) – 236.8]/2 = 1.3 = [0.5 + 0.9 + 0.65 + 0.55]/2

f offset_from_master = 144:241.05 – 144:7 – 1.3 – (207.75 - 0.2) = actual 25.2

Figure 39: Master, end-to-end, and slave one-step clocks- with asymmetry correction

The interpretations of key values for Figure 39 are given in Table 113.

Table 113: Interpretation of Figure 39 key values

Key	Reference	Comments
a	9.5.9.3 & 11.3.2	Sum of timestamp and correctionField is t_1
b	11.5.2.1 & 11.6.2	Residence time, 207.4, and the ingress path asymmetry (0.05) have been added to the correctionField
c	11.3.2	Timestamp set to 0 or an estimate, 144:300, of the egress timestamp. See "i" for correctionField
d	11.3.2	Note that requestingPortIdentity and sequenceId are those of the slave clock. The receiveTimestamp is t_4 excluding fractional nanoseconds. The correctionField is the correctionField from the Delay_Req message MINUS the fractional nanoseconds portion of t_4 (0.85)
e	11.3.2	The first term is the difference (t_2 – t_3). The second term is the difference in the receive and origin timestamps. The last two terms are the correction fields modified as explained in "g". Note that the computed meanPathDelay matches the actual assumed mean path delay from the figure.
f	11.2	The terms in order are t_2, originTimestamp meanPathDelay and Sync correctionField (modified as explained in "g"). Note that the computed offset is exactly as assumed due to the application of asymmetry corrections.
g	11.6.2	The ingress asymmetry (-0.20) on the path for the port receiving the Sync is added to

		the Sync correctionField before it is used in any computation. Thus this term (-0.20) appears added to the Sync correctionField (207.75) in both computations.
h	7.4.2	The asymmetry in the transit times is modeled as defined in 7.4.2. Thus for the path between the master and the transparent clock the mean transit time is 0.60 ns. The master to slave direction the actual transit time is 0.65 ns = 0.60 + (0.05) indicating that the correct asymmetry value is (0.05). In contrast the lower path the master-slave direction is shorter (0.7 compared to 0.9) yielding a negative asymmetry value (-0.2)
i	11.6.3	The egress path asymmetry (-0.2) has been subtracted from the correctionField
j	11.6.3 & 11.5.3.2	The transparent clock added the residence time (237.5) to and subtracted the egress path asymmetry (0.05) from the original correctionField value (0.2) or 0.2 + 237.5 – 0.05 = 237.65

C.2.3 Master two-step and end-to-end transparent and slave one-step clocks showing residence time and asymmetry computations

Figure 40 illustrates a two-step master clock interacting with one-step end-to-end transparent and slave clocks. A comparison of the content of the Sync and Follow_Up messages and the computations of meanPathDelay and offsetFromMaster in Figure 40 and Figure 39 shows the effect of using the Follow_Up message.

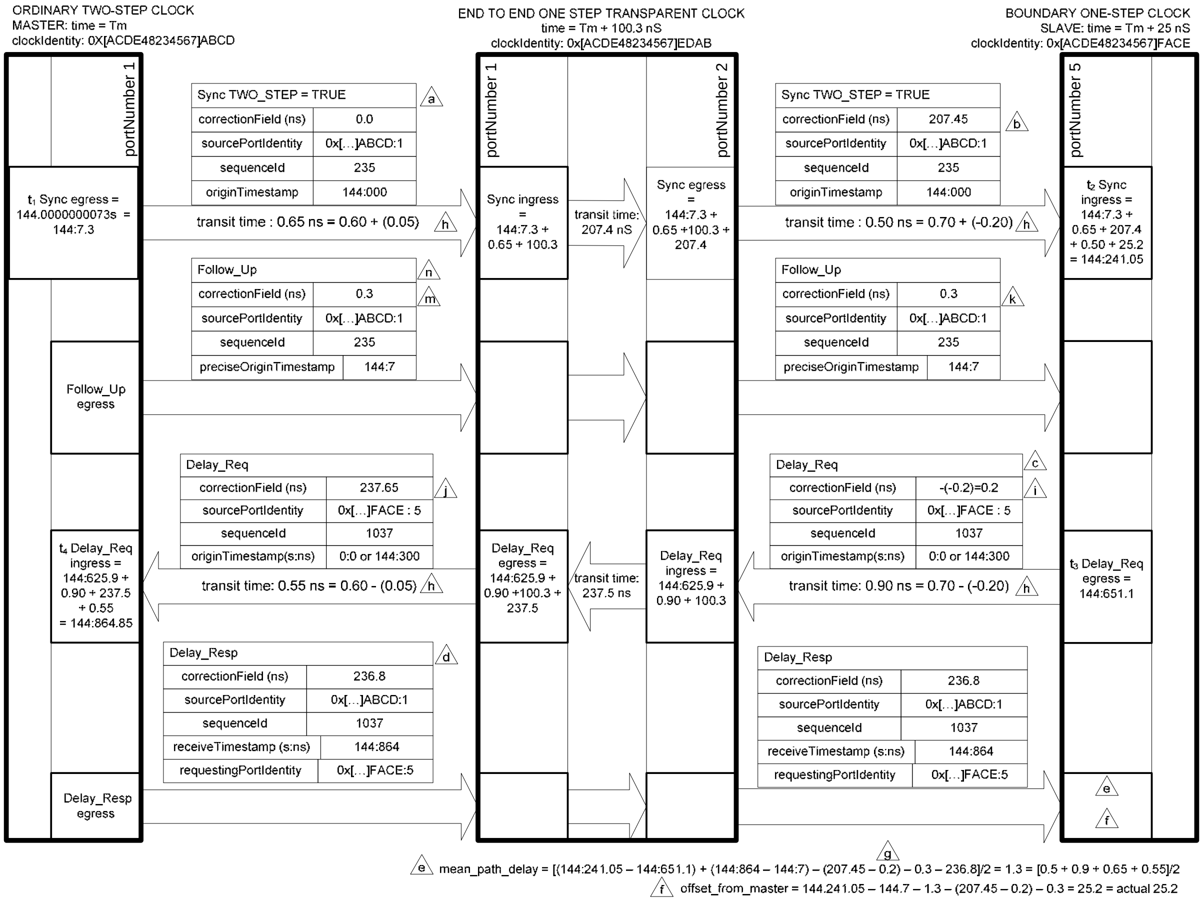

ORDINARY TWO-STEP CLOCK
MASTER: time = Tm
clockIdentity: 0X[ACDE48234567]ABCD
END TO END ONE STEP TRANSPARENT CLOCK
time = Tm + 100.3 nS
clockIdentity: 0x[ACDE48234567]EDAB
BOUNDARY ONE-STEP CLOCK
SLAVE: time = Tm + 25 nS
clockIdentity: 0x[ACDE48234567]FACE
portNumber 1
portNumber 1
portNumber 2
portNumber 5
Sync TWO_STEP = TRUE
a
correctionField (ns) 0.0
sourcePortIdentity 0x[…]ABCD:1
sequenceId 235
originTimestamp 144:000
t₁ Sync egress = 144.0000000073s = 144:7.3
transit time : 0.65 ns = 0.60 + (0.05)
h
Sync ingress = 144:7.3 + 0.65 + 100.3
transit time: 207.4 nS
Sync egress = 144:7.3 + 0.65 +100.3 + 207.4
Sync TWO_STEP = TRUE
correctionField (ns) 207.45
b
sourcePortIdentity 0x[…]ABCD:1
sequenceId 235
originTimestamp 144:000
transit time : 0.50 ns = 0.70 + (-0.20)
h
t₂ Sync ingress = 144:7.3 + 0.65 + 207.4 + 0.50 + 25.2 = 144:241.05
Follow_Up
n
correctionField (ns) 0.3
m
sourcePortIdentity 0x[…]ABCD:1
sequenceId 235
preciseOriginTimestamp 144:7
Follow_Up
correctionField (ns) 0.3
k
sourcePortIdentity 0x[…]ABCD:1
sequenceId 235
preciseOriginTimestamp 144:7
Follow_Up egress
Delay_Req
correctionField (ns) 237.65
j
sourcePortIdentity 0x[…]FACE : 5
sequenceId 1037
originTimestamp(s:ns) 0:0 or 144:300
Delay_Req
c
correctionField (ns) -(-0.2)=0.2
i
sourcePortIdentity 0x[…]FACE : 5
sequenceId 1037
originTimestamp(s:ns) 0:0 or 144:300
t₄ Delay_Req ingress = 144:625.9 + 0.90 + 237.5 + 0.55 = 144:864.85
transit time: 0.55 ns = 0.60 - (0.05)
h
Delay_Req egress = 144:625.9 + 0.90 +100.3 + 237.5
transit time: 237.5 ns
Delay_Req ingress = 144:625.9 + 0.90 + 100.3
transit time: 0.90 ns = 0.70 - (-0.20)
h
t₃ Delay_Req egress = 144:651.1
Delay_Resp
d
correctionField (ns) 236.8
sourcePortIdentity 0x[…]ABCD:1
sequenceId 1037
receiveTimestamp (s:ns) 144:864
requestingPortIdentity 0x[…]FACE:5
Delay_Resp
correctionField (ns) 236.8
sourcePortIdentity 0x[…]ABCD:1
sequenceId 1037
receiveTimestamp (s:ns) 144:864
requestingPortIdentity 0x[…]FACE:5
Delay_Resp egress
e
f
g
e mean_path_delay = [(144:241.05 – 144:651.1) + (144:864 – 144:7) – (207.45 – 0.2) – 0.3 – 236.8]/2 = 1.3 = [0.5 + 0.9 + 0.65 + 0.55]/2
f offset_from_master = 144.241.05 – 144.7 – 1.3 – (207.45 – 0.2) – 0.3 = 25.2 = actual 25.2

Figure 40: Master two-step and end-to-end transparent and slave one-step clocks- with asymmetry correction

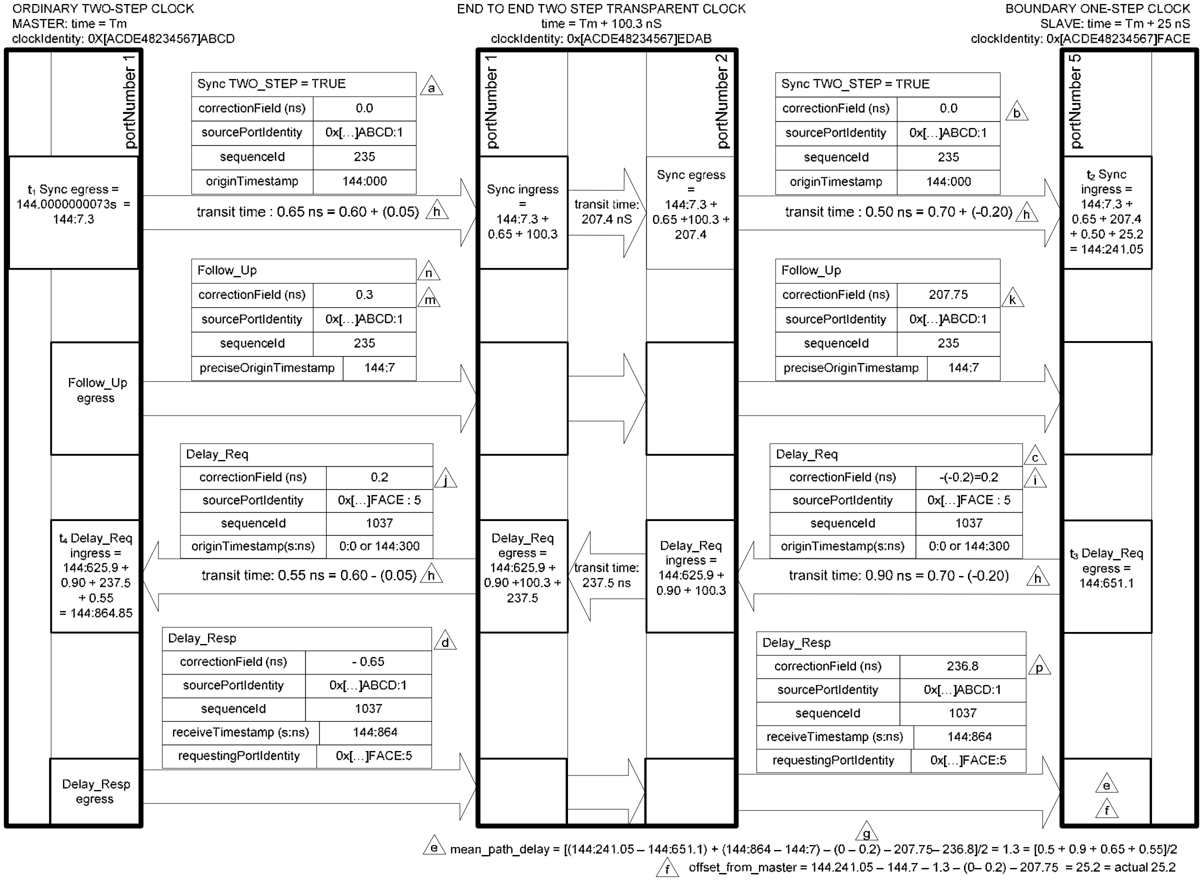
The interpretations of key values for Figure 40 are given in Table 114.

Table 114: Interpretation of Figure 40 key values

Key	Reference	Comments
a	9.5.9.4 & 11.3.2	The originTimestamp is an estimate, 144:0, of t_1 and the correctionField is 0.
b	11.5.2.1& 11.6.2	Residence time, 207.4, and the ingress path asymmetry (0.05) have been added to the correctionField
c	11.3.2	Timestamp set to 0 or an estimate, 144:300, of the egress timestamp. See "i" for correctionField
d	11.3.2	Note that requestingPortIdentity and sequenceId are those of the slave clock. The receiveTimestamp is t_4 excluding fractional nanoseconds. The correctionField is the correctionField from the Delay_Req message MINUS the fractional nanoseconds portion of t_4 (0.85)
e	11.3.2	The first term is the difference (t_2 – t_3). The second term is the difference in the receive and preciseOrigin timestamps. The last three terms are the correctionField of the Sync message modified as explained in "g", the correctionField of the Follow_Up and Delay_Resp messages respectively. Note that the computed meanPathDelay matches the actual assumed mean path delay from the figure.
f	11.2	The terms in order are t_2, preciseOriginTimestamp, meanPathDelay, Sync correctionField (modified as explained in "g"), and Follow_Up correctionField. Note that the computed offset is exactly as assumed due to the application of asymmetry corrections.
g	11.6.2	The ingress asymmetry (-0.20) on the path for the port receiving the Sync is added to the Sync correctionField before it is used in any computation. Thus this term (-0.20) appears added to the Sync correctionField (207.45) in both computations.
h	7.4.2	The asymmetry in the transit times is modeled as defined in 7.4.2. Thus for the path between the master and the transparent clock the mean transit time is 0.60 ns. The master to slave direction the actual transit time is 0.65 ns = 0.60 + (0.05) indicating that the correct asymmetry value is (0.05). In contrast the lower path the master-slave direction is shorter (0.7 compared to 0.9) yielding a negative asymmetry value (-0.2)
i	11.6.3	The egress path asymmetry (-0.2) has been subtracted from the correctionField
j	11.6.3 & 11.5.3.2	The transparent clock added the residence time (237.5) to and subtracted the egress path asymmetry (0.05) from the original correctionField value (0.2) or 0.2 + 237.5 – 0.05 = 237.65
k	11.5.2.1	No modification to the Follow_Up is made.
m	9.5.10	The sum of the correctionField and preciseOriginTimestamp is the egress time t_1
n	9.5.10	The sequenceId and sourcePortIdentity fields match those of the Sync message.

C.2.4 Master and end-to-end transparent two-step, and slave one-step clocks showing residence time and asymmetry computations

Figure 41 illustrates a two-step master clock interacting with a two-step end-to-end transparent clock and a one-step slave clock. A comparison of the content of the Sync and Follow_Up messages and the computations of meanPathDelay and offsetFromMaster in Figure 41, Figure 40, and Figure 39 shows the effect of using the Follow_Up message.


ORDINARY TWO-STEP CLOCK
MASTER: time = Tm
clockIdentity: 0X[ACDE48234567]ABCD
END TO END TWO STEP TRANSPARENT CLOCK
time = Tm + 100.3 nS
clockIdentity: 0x[ACDE48234567]EDAB
BOUNDARY ONE-STEP CLOCK
SLAVE: time = Tm + 25 nS
clockIdentity: 0x[ACDE48234567]FACE
portNumber 1
portNumber 1
portNumber 2
portNumber 5
a
b
t₁ Sync egress = 144.0000000073s = 144:7.3
transit time : 0.65 ns = 0.60 + (0.05)
h
Sync ingress = 144:7.3 + 0.65 + 100.3
transit time: 207.4 nS
Sync egress = 144:7.3 + 0.65 +100.3 + 207.4
transit time : 0.50 ns = 0.70 + (-0.20)
h
t₂ Sync ingress = 144:7.3 + 0.65 + 207.4 + 0.50 + 25.2 = 144:241.05
n
m
k
Follow_Up egress
c
i
j
t₄ Delay_Req ingress = 144:625.9 + 0.90 + 237.5 + 0.55 = 144:864.85
transit time: 0.55 ns = 0.60 - (0.05)
h
Delay_Req egress = 144:625.9 + 0.90 +100.3 + 237.5
transit time: 237.5 ns
Delay_Req ingress = 144:625.9 + 0.90 + 100.3
transit time: 0.90 ns = 0.70 - (-0.20)
h
t₃ Delay_Req egress = 144:651.1
d
p
Delay_Resp egress
e
f
g

Sync TWO_STEP = TRUE	
correctionField (ns)	0.0
sourcePortIdentity	0x[…]ABCD:1
sequenceId	235
originTimestamp	144:000

Sync TWO_STEP = TRUE	
correctionField (ns)	0.0
sourcePortIdentity	0x[…]ABCD:1
sequenceId	235
originTimestamp	144:000

Follow_Up	
correctionField (ns)	0.3
sourcePortIdentity	0x[…]ABCD:1
sequenceId	235
preciseOriginTimestamp	144:7

Follow_Up	
correctionField (ns)	207.75
sourcePortIdentity	0x[…]ABCD:1
sequenceId	235
preciseOriginTimestamp	144:7

Delay_Req	
correctionField (ns)	0.2
sourcePortIdentity	0x[…]FACE : 5
sequenceId	1037
originTimestamp(s:ns)	0:0 or 144:300

Delay_Req	
correctionField (ns)	-(-0.2)=0.2
sourcePortIdentity	0x[…]FACE : 5
sequenceId	1037
originTimestamp(s:ns)	0:0 or 144:300

Delay_Resp	
correctionField (ns)	- 0.65
sourcePortIdentity	0x[…]ABCD:1
sequenceId	1037
receiveTimestamp (s:ns)	144:864
requestingPortIdentity	0x[…]FACE:5

Delay_Resp	
correctionField (ns)	236.8
sourcePortIdentity	0x[…]ABCD:1
sequenceId	1037
receiveTimestamp (s:ns)	144:864
requestingPortIdentity	0x[…]FACE:5

e mean_path_delay = [(144:241.05 – 144:651.1) + (144:864 – 144:7) – (0 – 0.2) – 207.75– 236.8]/2 = 1.3 = [0.5 + 0.9 + 0.65 + 0.55]/2

f offset_from_master = 144.241.05 – 144.7 – 1.3 – (0– 0.2) – 207.75 = 25.2 = actual 25.2

Figure 41: Master and end-to-end transparent two-step, and one-step slave clocks- with asymmetry correction

The interpretations of key values for Figure 41 are given in Table 115.

Table 115: Interpretation of Figure 41 key values

Key	Reference	Comments
a	9.5.9.4 & 11.3.2	The originTimestamp is an estimate, 144:0, of t_1 and the correctionField is 0.
b	11.5.2.2 & 11.6.2	The residence time, 207.4, and the ingress path asymmetry (0.05) corrections are made in the Follow_Up message, see “k”.
c	11.3.2	Timestamp set to 0 or an estimate, 144:300, of the egress timestamp. See “i” for correctionField
d	11.3.2	Note that requestingPortIdentity and sequenceId are those of the slave clock. The receiveTimestamp is t_4 excluding fractional nanoseconds. The correctionField is the correctionField from the Delay_Req, 0.2, message MINUS the fractional nanoseconds portion of t_4 (0.85)
e	11.3.2	The first term is the difference (t_2 – t_3). The second term is the difference in the receive and preciseOrigin timestamps. The last three terms are the correctionField of the Sync message modified as explained in “g”, the correctionField of the Follow_Up and Delay_Resp messages respectively. Note that the computed meanPathDelay matches the actual assumed mean path delay from the figure.
f	11.2	The terms in order are t_2, preciseOriginTimestamp, meanPathDelay, Sync correctionField (modified as explained in “g”), and Follow_Up correctionField. Note that the computed offset is exactly as assumed due to the application of asymmetry corrections.

g	11.6.2	The ingress asymmetry (-0.20) on the path for the port receiving the Sync is added to the Sync correctionField before it is used in any computation. Thus this term (-0.20) appears added to the Sync correctionField (0.0) in both computations.
h	7.4.2	The asymmetry in the transit times is modeled as defined in 7.4.2. Thus for the path between the master and the transparent clock the mean transit time is 0.60 ns. The master to slave direction the actual transit time is 0.65 ns = 0.60 + (0.05) indicating that the correct asymmetry value is (0.05). In contrast the lower path the master-slave direction is shorter (0.7 compared to 0.9) yielding a negative asymmetry value (-0.2)
i	11.6.3	The egress path asymmetry (-0.2) has been subtracted from the correctionField
j	11.6.3 & 11.5.3.3	No corrections are made to the Delay_Req message. See "p" for the corrections for residence time and egress asymmetry.
k	11.5.2.2 & 11.6.2	The residence time, 207.4, and the ingress path asymmetry (0.05) corrections are added to the correctionField, (0.3) of the Follow_Up message.
m	9.5.10	The sum of the correctionField and preciseOriginTimestamp is the egress time t_1
n	9.5.10	The sequenceId and sourcePortIdentity fields match those of the Sync message.
p	11.6.3 & 11.5.3.3	The transparent clock added the residence time of the associated Delay_Req message, (237.5), to and subtracted the egress path asymmetry (0.05) from the original correctionField value of the Delay_Resp (-0.65) or -0.65 + 237.5 – 0.05 = 236.8

C.2.5 Master one-step, end-to-end transparent two-step, and one-step slave clocks showing residence time and asymmetry computations

Figure 42 illustrates a one-step master clock interacting with a two-step end-to-end transparent and a one-step slave clock. A comparison of the content of the Sync and Follow_Up messages and the computations of meanPathDelay and offsetFromMaster in Figure 42 , Figure 41, Figure 40, and Figure 39 shows the effect of using the Follow_Up message.

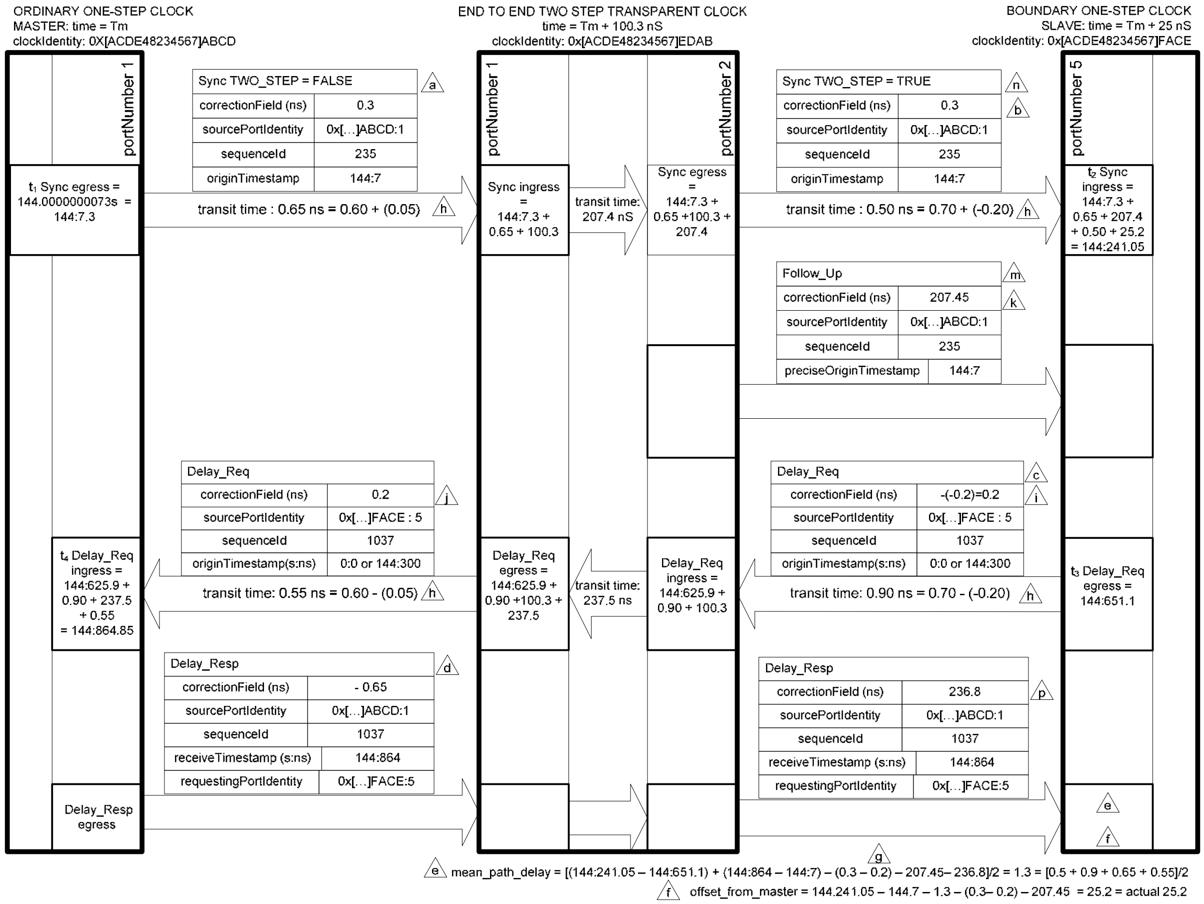

ORDINARY ONE-STEP CLOCK
MASTER: time = Tm
clockIdentity: 0X[ACDE48234567]ABCD
END TO END TWO STEP TRANSPARENT CLOCK
time = Tm + 100.3 nS
clockIdentity: 0x[ACDE48234567]EDAB
BOUNDARY ONE-STEP CLOCK
SLAVE: time = Tm + 25 nS
clockIdentity: 0x[ACDE48234567]FACE
portNumber 1
portNumber 1
portNumber 2
portNumber 5
Sync TWO_STEP = FALSE
a
correctionField (ns) 0.3
sourcePortIdentity 0x[…]ABCD:1
sequenceId 235
originTimestamp 144:7
t₁ Sync egress = 144.0000000073s = 144:7.3
transit time : 0.65 ns = 0.60 + (0.05)
h
Sync ingress = 144:7.3 + 0.65 + 100.3
transit time: 207.4 nS
Sync egress = 144:7.3 + 0.65 +100.3 + 207.4
Sync TWO_STEP = TRUE
n
b
correctionField (ns) 0.3
sourcePortIdentity 0x[…]ABCD:1
sequenceId 235
originTimestamp 144:7
transit time : 0.50 ns = 0.70 + (-0.20)
h
t₂ Sync ingress = 144:7.3 + 0.65 + 207.4 + 0.50 + 25.2 = 144:241.05
Follow_Up
m
k
correctionField (ns) 207.45
sourcePortIdentity 0x[…]ABCD:1
sequenceId 235
preciseOriginTimestamp 144:7
Delay_Req
j
correctionField (ns) 0.2
sourcePortIdentity 0x[…]FACE : 5
sequenceId 1037
originTimestamp(s:ns) 0:0 or 144:300
t₄ Delay_Req ingress = 144:625.9 + 0.90 + 237.5 + 0.55 = 144:864.85
transit time: 0.55 ns = 0.60 - (0.05)
h
Delay_Req egress = 144:625.9 + 0.90 +100.3 + 237.5
transit time: 237.5 ns
Delay_Req ingress = 144:625.9 + 0.90 + 100.3
Delay_Req
c
i
correctionField (ns) -(-0.2)=0.2
sourcePortIdentity 0x[…]FACE : 5
sequenceId 1037
originTimestamp(s:ns) 0:0 or 144:300
transit time: 0.90 ns = 0.70 - (-0.20)
h
t₃ Delay_Req egress = 144:651.1
Delay_Resp
d
correctionField (ns) - 0.65
sourcePortIdentity 0x[…]ABCD:1
sequenceId 1037
receiveTimestamp (s:ns) 144:864
requestingPortIdentity 0x[…]FACE:5
Delay_Resp egress
Delay_Resp
p
correctionField (ns) 236.8
sourcePortIdentity 0x[…]ABCD:1
sequenceId 1037
receiveTimestamp (s:ns) 144:864
requestingPortIdentity 0x[…]FACE:5
e
f
g
e mean_path_delay = [(144:241.05 – 144:651.1) + (144:864 – 144:7) – (0.3 – 0.2) – 207.45– 236.8]/2 = 1.3 = [0.5 + 0.9 + 0.65 + 0.55]/2
f offset_from_master = 144.241.05 – 144.7 – 1.3 – (0.3– 0.2) – 207.45 = 25.2 = actual 25.2

Figure 42: One-step master, two-step end-to-end transparent, and one-step slave clocks- with asymmetry correction

The interpretations of key values for Figure 42 are given in Table 116.

Table 116: Interpretation of Figure 42 key values

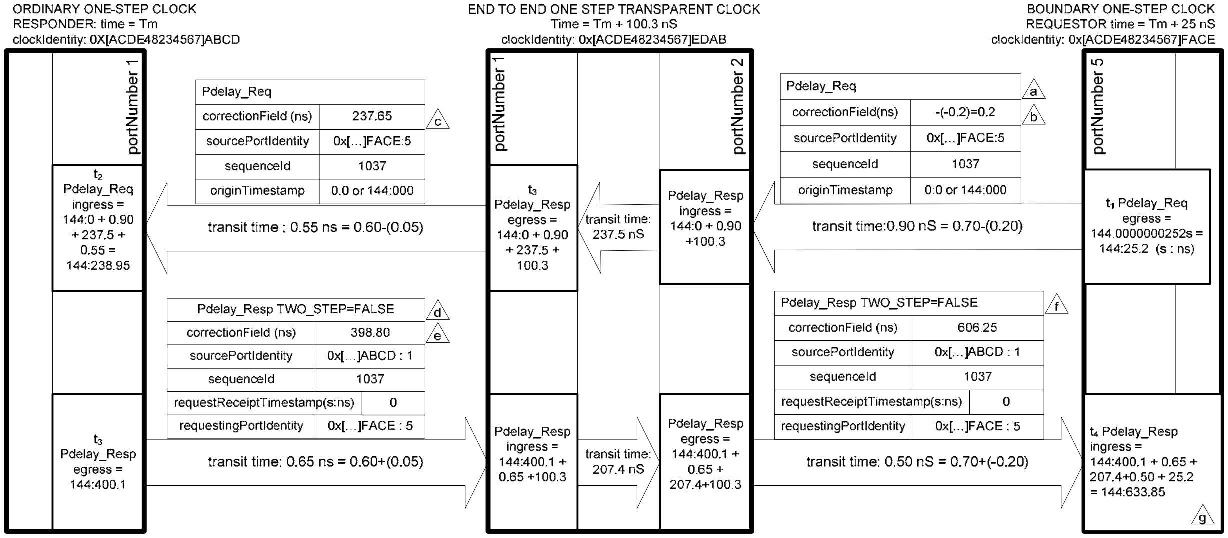
Key	Reference	Comments
a	9.5.9.3 & 11.3.2	Sum of timestamp and correctionField is t_1
b	11.5.2.2 & 11.6.2	The residence time, 207.4, and the ingress path asymmetry (0.05) corrections are made in the Follow_Up message, see “k”.
c	11.3.2	Timestamp set to 0 or an estimate, 144:300, of the egress timestamp. See “i” for correctionField
d	11.3.2	Note that requestingPortIdentity and sequenceId are those of the slave clock. The receiveTimestamp is t_4 excluding fractional nanoseconds. The correctionField is the correctionField from the Delay_Req, 0.2, message MINUS the fractional nanoseconds portion of t_4 (0.85)
e	11.3.2	The first term is the difference (t_2 – t_3). The second term is the difference in the receive and preciseOrigin timestamps. The last three terms are the correctionField of the Sync message modified as explained in “g”, the correctionField of the Follow_Up and Delay_Resp messages respectively. Note that the computed meanPathDelay matches the actual assumed mean path delay from the figure.
f	11.2	The terms in order are t_2, preciseOriginTimestamp, meanPathDelay, Sync correctionField (modified as explained in “g”), and Follow_Up correctionField. Note that the computed offset is exactly as assumed due to the application of asymmetry

		corrections.
g	11.6.2	The ingress asymmetry (-0.20) on the path for the port receiving the Sync is added to the Sync correctionField before it is used in any computation. Thus this term (-0.20) appears added to the Sync correctionField (0.3) in both computations.
h	7.4.2	The asymmetry in the transit times is modeled as defined in 7.4.2. Thus for the path between the master and the transparent clock the mean transit time is 0.60 ns. The master to slave direction the actual transit time is 0.65 ns = 0.60 + (0.05) indicating that the correct asymmetry value is (0.05). In contrast the lower path the master-slave direction is shorter (0.7 compared to 0.9) yielding a negative asymmetry value (-0.2)
i	11.6.3	The egress path asymmetry (-0.2) has been subtracted from the correctionField
j	11.6.3 & 11.5.3.3	No corrections are made to the Delay_Req message. See “p” for the corrections for residence time and egress asymmetry.
k	11.5.2.2 & 11.6.2	The residence time, 207.4, is entered in the correctionField and subsequently the ingress path asymmetry (0.05) correction is added to the correctionField of the Follow_Up message.
m	11.5.2.2	The originTimestamp of the Sync message is copied into the preciseOriginTimestamp. The domainNumber (not shown), sourcePortIdentity, and sequenceId fields of the Sync message are copied into the corresponding fields of the Follow_Up message.
n	11.5.2.2	The twoStepFlag is set to TRUE.
p	11.6.3 & 11.5.3.3	The transparent clock added the residence time of the associated Delay_Req message, (237.5), to and subtracted the egress path asymmetry (0.05) from the original correctionField value of the Delay_Resp (-0.65) or -0.65 + 237.5 – 0.05 = 236.8

C.3 Computations using the peer delay mechanism

C.3.1 One-step peer requestor, end-to-end transparent, and peer responder clocks showing residence time and asymmetry computations

Figure 43 illustrates a one-step peer requestor clock interacting with a one-step end-to-end transparent and peer responder clocks.


ORDINARY ONE-STEP CLOCK
RESPONDER: time = Tm
clockIdentity: 0X[ACDE48234567]ABCD
END TO END ONE STEP TRANSPARENT CLOCK
Time = Tm + 100.3 nS
clockIdentity: 0x[ACDE48234567]EDAB
BOUNDARY ONE-STEP CLOCK
REQUESTOR time = Tm + 25 nS
clockIdentity: 0x[ACDE48234567]FACE
portNumber 1
portNumber 1
portNumber 2
portNumber 5
a
b
c
d
e
f
g
t₁ Pdelay_Req egress = 144.0000000252s = 144:25.2 (s : ns)
transit time:0.90 nS = 0.70-(0.20)
Pdelay_Resp ingress = 144:0 + 0.90 +100.3
transit time: 237.5 nS
t₃ Pdelay_Resp egress = 144:0 + 0.90 + 237.5 + 100.3
transit time : 0.55 ns = 0.60-(0.05)
t₂ Pdelay_Req ingress = 144:0 + 0.90 + 237.5 + 0.55 = 144:238.95
t₃ Pdelay_Resp egress = 144:400.1
transit time: 0.65 ns = 0.60+(0.05)
Pdelay_Resp ingress = 144:400.1 + 0.65 +100.3
transit time: 207.4 nS
Pdelay_Resp egress = 144:400.1 + 0.65 + 207.4+100.3
transit time: 0.50 nS = 0.70+(-0.20)
t₄ Pdelay_Resp ingress = 144:400.1 + 0.65 + 207.4+0.50 + 25.2 = 144:633.85

Pdelay_Req	
correctionField (ns)	237.65
sourcePortIdentity	0x[…]FACE:5
sequenceId	1037
originTimestamp	0.0 or 144:000

Pdelay_Resp TWO_STEP=FALSE	
correctionField (ns)	398.80
sourcePortIdentity	0x[…]ABCD : 1
sequenceId	1037
requestReceiptTimestamp(s:ns)	0
requestingPortIdentity	0x[…]FACE : 5

Pdelay_Req	
correctionField(ns)	-(-0.2)=0.2
sourcePortIdentity	0x[…]FACE:5
sequenceId	1037
originTimestamp	0:0 or 144:000

Pdelay_Resp TWO_STEP=FALSE	
correctionField (ns)	606.25
sourcePortIdentity	0x[…]ABCD : 1
sequenceId	1037
requestReceiptTimestamp(s:ns)	0
requestingPortIdentity	0x[…]FACE : 5

g mean_path_delay = [(144.633.85 – 144:25.2) – 606.05]/2 = 1.3 = [0.5 + 0.9 + 0.65 + 0.55]/2

Figure 43: One-step peer responder, end-to-end transparent, and peer requestor clocks- with asymmetry correction

The interpretations of key values for Figure 43 are given in Table 117.

Table 117: Interpretation of Figure 43 key values

Key	Reference	Comments
a	11.4.3	The originTimestamp is 0 or and estimate of the egress timestamp t_1
b	11.4.3 & 11.6.4	The correctionField is 0 – (– 0.2) = 0.2 where (– 0.2) is the asymmetry on the egress path.
c	11.5.4.2 & 11.6.4	The correctionField is modified by adding the residence time in the end-to-end transparent clock and subtracting the asymmetry on the egress path. 0.2 + 237.5 – (0.05) = 237.65
d	11.4.3	requestReceiptTimestamp = 0. requestingPortIdentity and sequenceId fields are copied from the sourcePortIdentity and sequenceId fields of the Pdelay_Req message.
e	11.4.3	The correctionField is the sum of the correctionField of the Pdelay_Req message and the turnaround time (t_3 – t_2), i.e. 237.65 + (144:400.1 – 144:238.95) = 398.80
f	11.5.4.2 & 11.6.5	The correctionField is modified by adding the residence time 207.4 and the ingress asymmetry 0.05 to the original correctionField. 398.80 + 207.4 +0.05 = 606.25
g	11.4.3 & 11.6.5	Prior to computing the meanPathDelay the correctionField of the Pdelay_Resp message is modified by adding the ingress asymmetry 0.2. i.e. 606.25 + (- 0.2) = 606.05. meanPathDelay = [(t_4 – t_1) – correctionField of Pdelay_Resp]/2 = [(144:633.85 – 144:25.2) – 606.05]/2 = 1.3

C.3.2 One-step peer requestor, two-step end-to-end transparent, and one-step peer responder clocks showing residence time and asymmetry computations

Figure 44 illustrates a one-step peer requestor clock interacting with a two-step end-to-end transparent and one-step peer responder clocks.

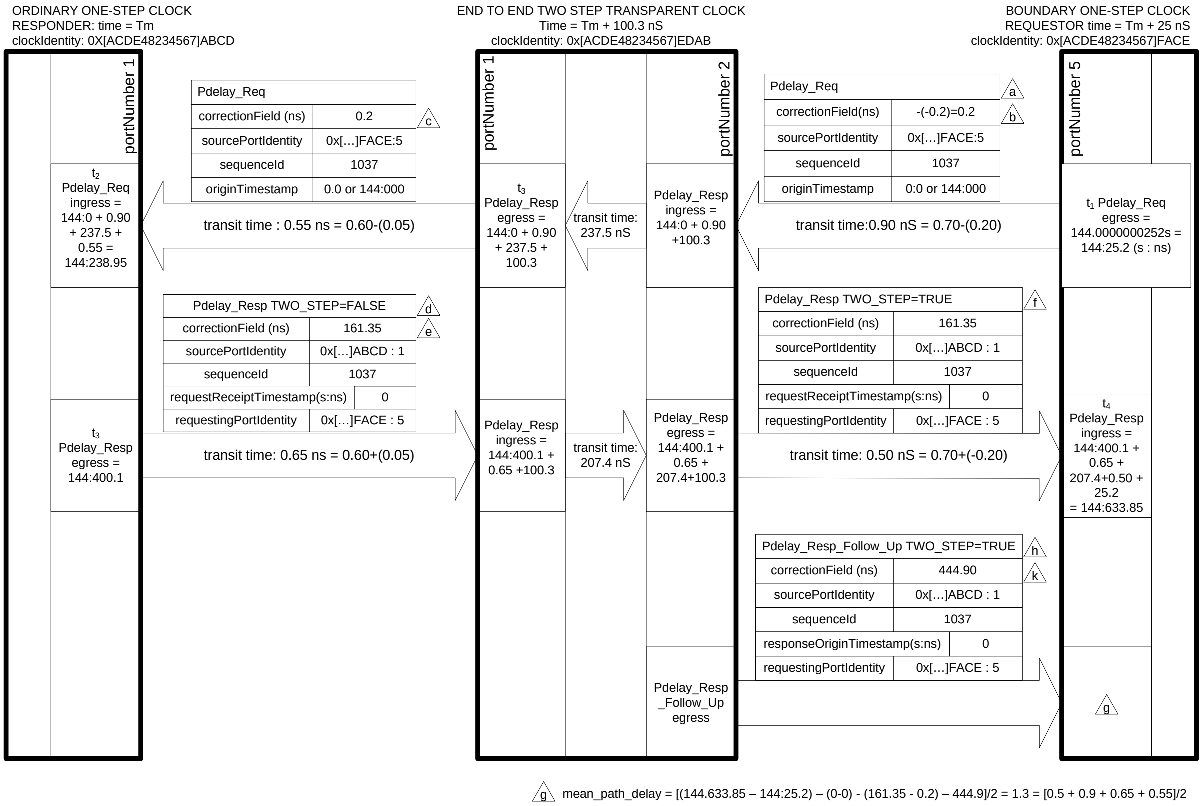

ORDINARY ONE-STEP CLOCK
RESPONDER: time = Tm
clockIdentity: 0X[ACDE48234567]ABCD
END TO END TWO STEP TRANSPARENT CLOCK
Time = Tm + 100.3 nS
clockIdentity: 0x[ACDE48234567]EDAB
BOUNDARY ONE-STEP CLOCK
REQUESTOR time = Tm + 25 nS
clockIdentity: 0x[ACDE48234567]FACE
portNumber 1
portNumber 1
portNumber 2
portNumber 5
Pdelay_Req
correctionField (ns) 0.2
sourcePortIdentity 0x[…]FACE:5
sequenceId 1037
originTimestamp 0.0 or 144:000
c
Pdelay_Req
correctionField(ns) -(-0.2)=0.2
sourcePortIdentity 0x[…]FACE:5
sequenceId 1037
originTimestamp 0:0 or 144:000
a
b
t2 Pdelay_Req ingress = 144:0 + 0.90 + 237.5 + 0.55 = 144:238.95
transit time : 0.55 ns = 0.60-(0.05)
t3 Pdelay_Resp egress = 144:0 + 0.90 + 237.5 + 100.3
transit time: 237.5 nS
Pdelay_Resp ingress = 144:0 + 0.90 +100.3
transit time:0.90 nS = 0.70-(0.20)
t1 Pdelay_Req egress = 144.0000000252s = 144:25.2 (s : ns)
Pdelay_Resp TWO_STEP=FALSE
correctionField (ns) 161.35
sourcePortIdentity 0x[…]ABCD : 1
sequenceId 1037
requestReceiptTimestamp(s:ns) 0
requestingPortIdentity 0x[…]FACE : 5
d
e
Pdelay_Resp TWO_STEP=TRUE
correctionField (ns) 161.35
sourcePortIdentity 0x[…]ABCD : 1
sequenceId 1037
requestReceiptTimestamp(s:ns) 0
requestingPortIdentity 0x[…]FACE : 5
f
t3 Pdelay_Resp egress = 144:400.1
transit time: 0.65 ns = 0.60+(0.05)
Pdelay_Resp ingress = 144:400.1 + 0.65 +100.3
transit time: 207.4 nS
Pdelay_Resp egress = 144:400.1 + 0.65 + 207.4+100.3
transit time: 0.50 nS = 0.70+(-0.20)
t4 Pdelay_Resp ingress = 144:400.1 + 0.65 + 207.4+0.50 + 25.2 = 144:633.85
Pdelay_Resp_Follow_Up TWO_STEP=TRUE
correctionField (ns) 444.90
sourcePortIdentity 0x[…]ABCD : 1
sequenceId 1037
responseOriginTimestamp(s:ns) 0
requestingPortIdentity 0x[…]FACE : 5
h
k
Pdelay_Resp _Follow_Up egress
g
g mean_path_delay = [(144.633.85 – 144:25.2) – (0-0) - (161.35 - 0.2) – 444.9]/2 = 1.3 = [0.5 + 0.9 + 0.65 + 0.55]/2

Figure 44: One-step peer responder, two-step end-to-end transparent, and one-step peer requestor clocks- with asymmetry correction

The interpretations of key values for Figure 44 are given in Table 118.

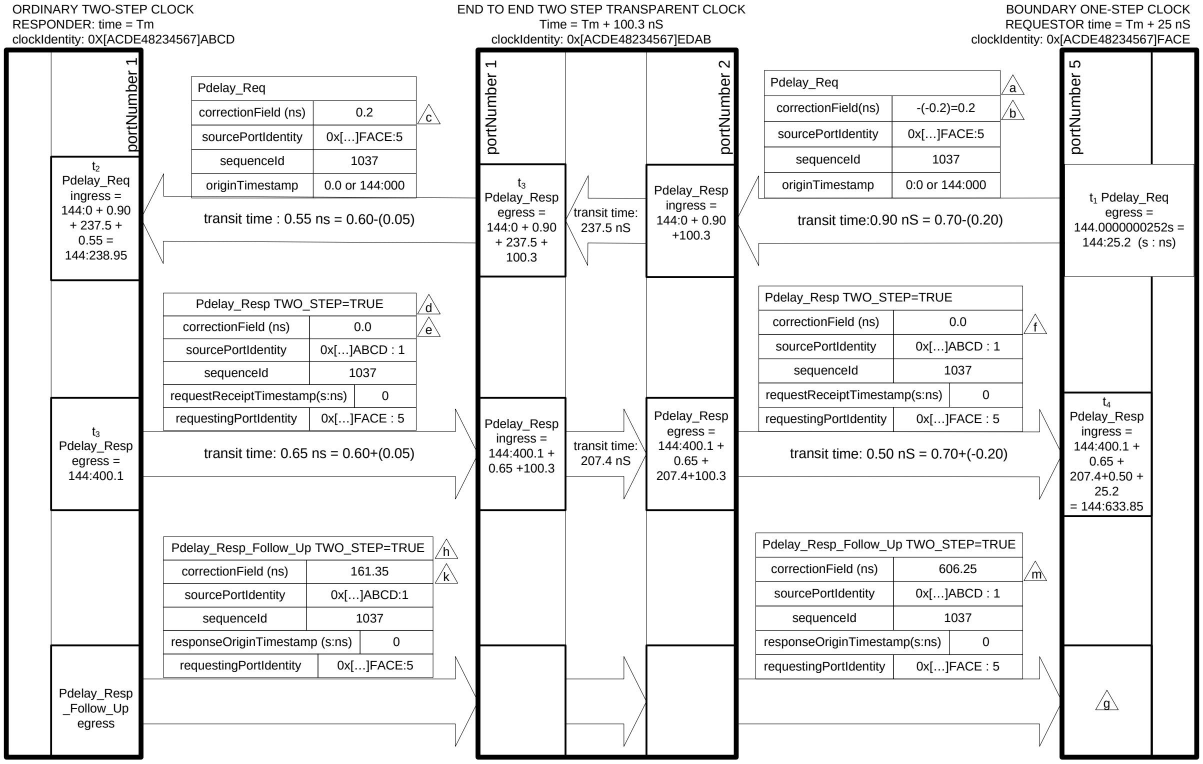
Table 118: Interpretation of Figure 44 key values

Key	Reference	Comments
a	11.4.3	The originTimestamp is 0 or and estimate of the egress timestamp t_1
b	11.4.3 & 11.6.4	The correctionField is 0 – (– 0.2) = 0.2 where (– 0.2) is the asymmetry on the egress path.
c	11.5.4.2 & 11.6.4	The correctionField is not modified. See “j” for corrections for residence time of Pdelay_Req and egress path asymmetry.
d	11.4.3	requestReceiptTimestamp = 0. requestingPortIdentity and sequenceId fields are copied from the sourcePortIdentity and sequenceId fields of the Pdelay_Req message.
e	11.4.3	The correctionField is the sum of the correctionField of the Pdelay_Req message and the turnaround time ($t_3 – t_2$), i.e. 0.2 + (144:400.1 – 144:238.95) = 161.35
f	11.5.4.3 & 11.6.5	The PTP fields of the Pdelay_Resp are not modified except for the twoStepFlag, which is set to TRUE. See “j” for corrections for the residence time of the Pdelay_Resp message and ingress path asymmetry.
g	11.4.3 & 11.6.5	Prior to computing the meanPathDelay the correctionField of the Pdelay_Resp message is modified by adding the ingress asymmetry 0.2. i.e. 161.35 + (- 0.2) = 161.05. meanPathDelay = [(t_4 – t_1) – (responseOriginTimestamp – requestReceiptTimestamp) – correctionField of Pdelay_Resp – correctionField of Pdelay_Resp_Follow_Up]/2 = [(144:633.85 – 144:25.2) – (0 – 0) – (161.35 – 0.2)) – 444.90]/2 = 1.3
h	11.5.4.3	The sourcePortIdentity, sequenceId, and requestingPortIdentity fields of the Pdelay_Resp message are copied into the same fields of the Pdelay_Resp_Follow_Up message. The TWO-STEP flag is set to TRUE. The

		responseOriginTimestamp is set to 0.
j	11.5.4.3, 11.6.4 & 11.6.5	The correctionField is the sum of the residence times of the Pdelay_Req and Pdelay_Resp messages minus the egress asymmetry for the Pdelay_Req plus the ingress asymmetry for the Pdelay_Resp messages, i.e. 237.5 + 207.4 – (0.05) + (0.05) = 444.90

C.3.3 One-step peer requestor, two-step end-to-end transparent, and two-step peer responder clocks showing residence time and asymmetry computations: option 1

Figure 45 illustrates a one-step peer requestor clock interacting with a two-step end-to-end transparent and two-step peer responder clocks. The two-step responder uses the first option of 11.4.3 in generating the Pdelay_Resp and Pdelay_Resp_Follow_Up messages.


ORDINARY TWO-STEP CLOCK
RESPONDER: time = Tm
clockIdentity: 0X[ACDE48234567]ABCD
END TO END TWO STEP TRANSPARENT CLOCK
Time = Tm + 100.3 nS
clockIdentity: 0x[ACDE48234567]EDAB
BOUNDARY ONE-STEP CLOCK
REQUESTOR time = Tm + 25 nS
clockIdentity: 0x[ACDE48234567]FACE
portNumber 1
portNumber 1
portNumber 2
portNumber 5
Pdelay_Req
correctionField (ns) 0.2
sourcePortIdentity 0x[…]FACE:5
sequenceId 1037
originTimestamp 0.0 or 144:000
c
Pdelay_Req
correctionField(ns) -(-0.2)=0.2
sourcePortIdentity 0x[…]FACE:5
sequenceId 1037
originTimestamp 0:0 or 144:000
a
b
t2 Pdelay_Req ingress = 144:0 + 0.90 + 237.5 + 0.55 = 144:238.95
transit time : 0.55 ns = 0.60-(0.05)
t3 Pdelay_Resp egress = 144:0 + 0.90 + 237.5 + 100.3
transit time: 237.5 nS
Pdelay_Resp ingress = 144:0 + 0.90 +100.3
transit time:0.90 nS = 0.70-(0.20)
t1 Pdelay_Req egress = 144.0000000252s = 144:25.2 (s : ns)
Pdelay_Resp TWO_STEP=TRUE
correctionField (ns) 0.0
sourcePortIdentity 0x[…]ABCD : 1
sequenceId 1037
requestReceiptTimestamp(s:ns) 0
requestingPortIdentity 0x[…]FACE : 5
d
e
Pdelay_Resp TWO_STEP=TRUE
correctionField (ns) 0.0
sourcePortIdentity 0x[…]ABCD : 1
sequenceId 1037
requestReceiptTimestamp(s:ns) 0
requestingPortIdentity 0x[…]FACE : 5
f
t3 Pdelay_Resp egress = 144:400.1
transit time: 0.65 ns = 0.60+(0.05)
Pdelay_Resp ingress = 144:400.1 + 0.65 +100.3
transit time: 207.4 nS
Pdelay_Resp egress = 144:400.1 + 0.65 + 207.4+100.3
transit time: 0.50 nS = 0.70+(-0.20)
t4 Pdelay_Resp ingress = 144:400.1 + 0.65 + 207.4+0.50 + 25.2 = 144:633.85
Pdelay_Resp_Follow_Up TWO_STEP=TRUE
correctionField (ns) 161.35
sourcePortIdentity 0x[…]ABCD:1
sequenceId 1037
responseOriginTimestamp (s:ns) 0
requestingPortIdentity 0x[…]FACE:5
h
k
Pdelay_Resp_Follow_Up TWO_STEP=TRUE
correctionField (ns) 606.25
sourcePortIdentity 0x[…]ABCD : 1
sequenceId 1037
responseOriginTimestamp(s:ns) 0
requestingPortIdentity 0x[…]FACE : 5
m
Pdelay_Resp _Follow_Up egress
g

g mean_path_delay = [(144.633.85 – 144:25.2) – (0-0) – 0 – (606.25 + (-0.2))]/2 = 1.3 = [0.5 + 0.9 + 0.65 + 0.55]/2

Figure 45: Two-step peer responder, two-step end-to-end transparent, and one-step peer requestor clocks: option 1- with asymmetry correction

The interpretations of key values for Figure 45 are given in Table 119.

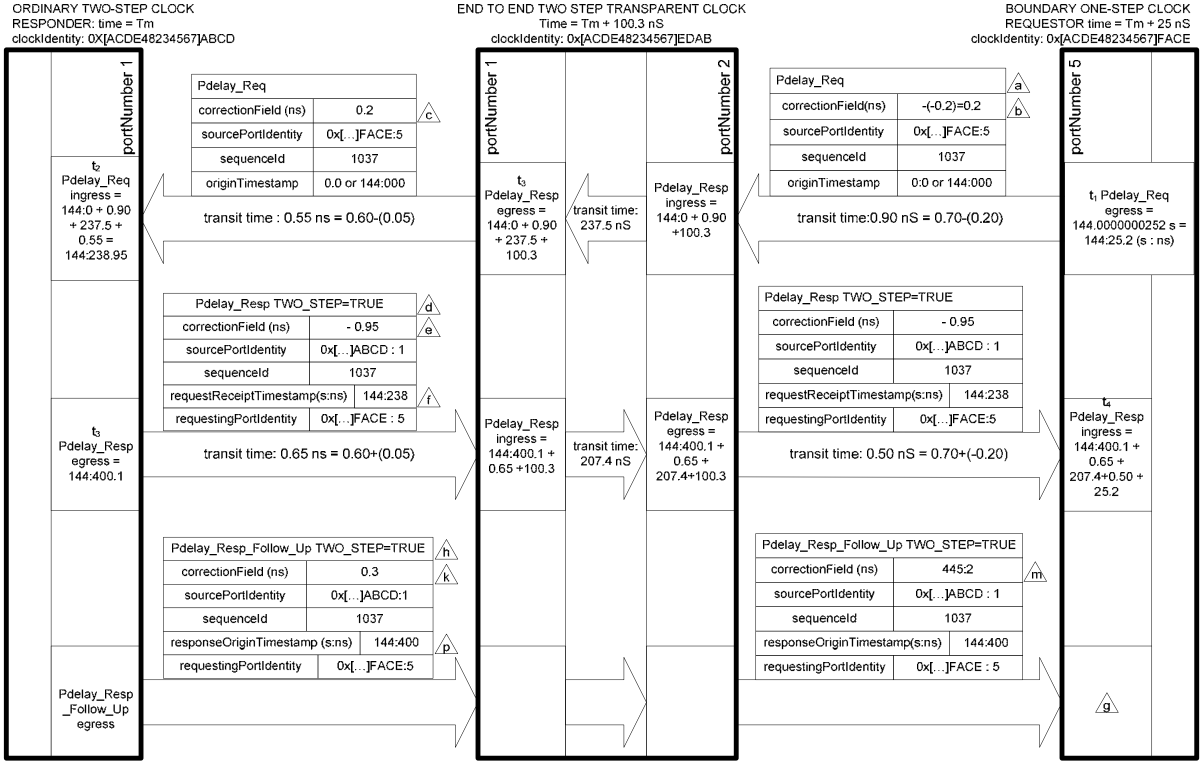
Table 119: Interpretation of Figure 45 key values

Key	Reference	Comments
a	11.4.3	The originTimestamp is 0 or and estimate of the egress timestamp t_1
b	11.4.3 & 11.6.4	The correctionField is 0 – (– 0.2) = 0.2 where (– 0.2) is the asymmetry on the egress path.
c	11.5.4.2 & 11.6.4	The correctionField is not modified. See “m” for corrections for residence time of Pdelay_Req and egress path asymmetry.
d	11.4.3	requestReceiptTimestamp = 0. requestingPortIdentity and sequenceId fields are copied from the sourcePortIdentity and sequenceId fields of the Pdelay_Req

		message.
e	11.4.3	The correctionField is set to 0.
f	11.5.4.3 & 11.6.5	The PTP fields of the Pdelay_Resp are not modified. See “m” for corrections for the residence time of the Pdelay_Resp message and ingress path asymmetry.
g	11.4.3 & 11.6.5	Prior to computing the meanPathDelay the correctionField of the Pdelay_Resp message is modified by adding the ingress asymmetry 0.2. i.e. 606.25 + (- 0.2) = 606.05. meanPathDelay = [(t_4 – t_1) – (responseOriginTimestamp – requestReceiptTimestamp) – correctionField of Pdelay_Resp – correctionField of Pdelay_Resp_Follow_Up]/2 = [(144:633.85 – 144:25.2) – (0 – 0) – 0 – 606.05]/2 = 1.3
h	11.5.4.3	The sourcePortIdentity, sequenceId, and requestingPortIdentity fields of the Pdelay_Resp message are copied into the same fields of the Pdelay_Resp_Follow_Up message. The TWO-STEP flag is set to TRUE. The responseOriginTimestamp is set to 0.
k	11.5.4.3, 11.6.4 & 11.6.5	The correctionField is the sum of the correctionField of the Pdelay_Req message and the turnaround time, i.e. (t_3 – t_2). 0.2 + (144:400.1 – 144:238.95) = 161.35
m	11.5.4.3, 11.6.4 & 11.6.5	The correctionField is the sum of the original correctionField and the residence times of the Pdelay_Req and Pdelay_Resp messages minus the egress asymmetry for the Pdelay_Req plus the ingress asymmetry for the Pdelay_Resp messages, i.e. 161.35 +237.5 + 207.4 – (0.05) + (0.05) = 606.25

C.3.4 One-step peer requestor, two-step end-to-end transparent, and two-step peer responder clocks showing residence time and asymmetry computations: option 2

Figure 46 illustrates a one-step peer requestor clock interacting with a two-step end-to-end transparent and two-step peer responder clocks. The two-step responder uses the second option of 11.4.3 in generating the Pdelay_Resp and Pdelay_Resp_Follow_Up messages.


ORDINARY TWO-STEP CLOCK
RESPONDER: time = Tm
clockIdentity: 0X[ACDE48234567]ABCD
END TO END TWO STEP TRANSPARENT CLOCK
Time = Tm + 100.3 nS
clockIdentity: 0x[ACDE48234567]EDAB
BOUNDARY ONE-STEP CLOCK
REQUESTOR time = Tm + 25 nS
clockIdentity: 0x[ACDE48234567]FACE
portNumber 1
portNumber 1
portNumber 2
portNumber 5
Pdelay_Req
correctionField (ns) | 0.2
sourcePortIdentity | 0x[…]FACE:5
sequenceId | 1037
originTimestamp | 0.0 or 144:000
c
Pdelay_Req
correctionField(ns) | -(-0.2)=0.2
sourcePortIdentity | 0x[…]FACE:5
sequenceId | 1037
originTimestamp | 0:0 or 144:000
a
b
t2 Pdelay_Req ingress = 144:0 + 0.90 + 237.5 + 0.55 = 144:238.95
transit time : 0.55 ns = 0.60-(0.05)
t3 Pdelay_Resp egress = 144:0 + 0.90 + 237.5 + 100.3
transit time: 237.5 nS
Pdelay_Resp ingress = 144:0 + 0.90 +100.3
transit time:0.90 nS = 0.70-(0.20)
t1 Pdelay_Req egress = 144.0000000252 s = 144:25.2 (s : ns)
Pdelay_Resp TWO_STEP=TRUE
correctionField (ns) | - 0.95
sourcePortIdentity | 0x[…]ABCD : 1
sequenceId | 1037
requestReceiptTimestamp(s:ns) | 144:238
requestingPortIdentity | 0x[…]FACE : 5
d
e
f
Pdelay_Resp TWO_STEP=TRUE
correctionField (ns) | - 0.95
sourcePortIdentity | 0x[…]ABCD : 1
sequenceId | 1037
requestReceiptTimestamp(s:ns) | 144:238
requestingPortIdentity | 0x[…]FACE:5
t3 Pdelay_Resp egress = 144:400.1
transit time: 0.65 ns = 0.60+(0.05)
Pdelay_Resp ingress = 144:400.1 + 0.65 +100.3
transit time: 207.4 nS
Pdelay_Resp egress = 144:400.1 + 0.65 + 207.4+100.3
transit time: 0.50 nS = 0.70+(-0.20)
t4 Pdelay_Resp ingress = 144:400.1 + 0.65 + 207.4+0.50 + 25.2
Pdelay_Resp_Follow_Up TWO_STEP=TRUE
correctionField (ns) | 0.3
sourcePortIdentity | 0x[…]ABCD:1
sequenceId | 1037
responseOriginTimestamp (s:ns) | 144:400
requestingPortIdentity | 0x[…]FACE:5
h
k
p
Pdelay_Resp_Follow_Up TWO_STEP=TRUE
correctionField (ns) | 445:2
sourcePortIdentity | 0x[…]ABCD : 1
sequenceId | 1037
responseOriginTimestamp(s:ns) | 144:400
requestingPortIdentity | 0x[…]FACE : 5
m
Pdelay_Resp _Follow_Up egress
g

g mean_path_delay = [(144.633.85 – 144:25.2) – (144:400 –144:238) – (– 0.95 – 0.20) – 445.2]/2 = 1.3 = [0.5 + 0.9 + 0.65 + 0.55]/2

Figure 46: Two-step peer responder, two-step end-to-end transparent, and one-step peer requestor clocks: option 2- with asymmetry correction

The interpretations of key values for Figure 46 are given in Table 120.

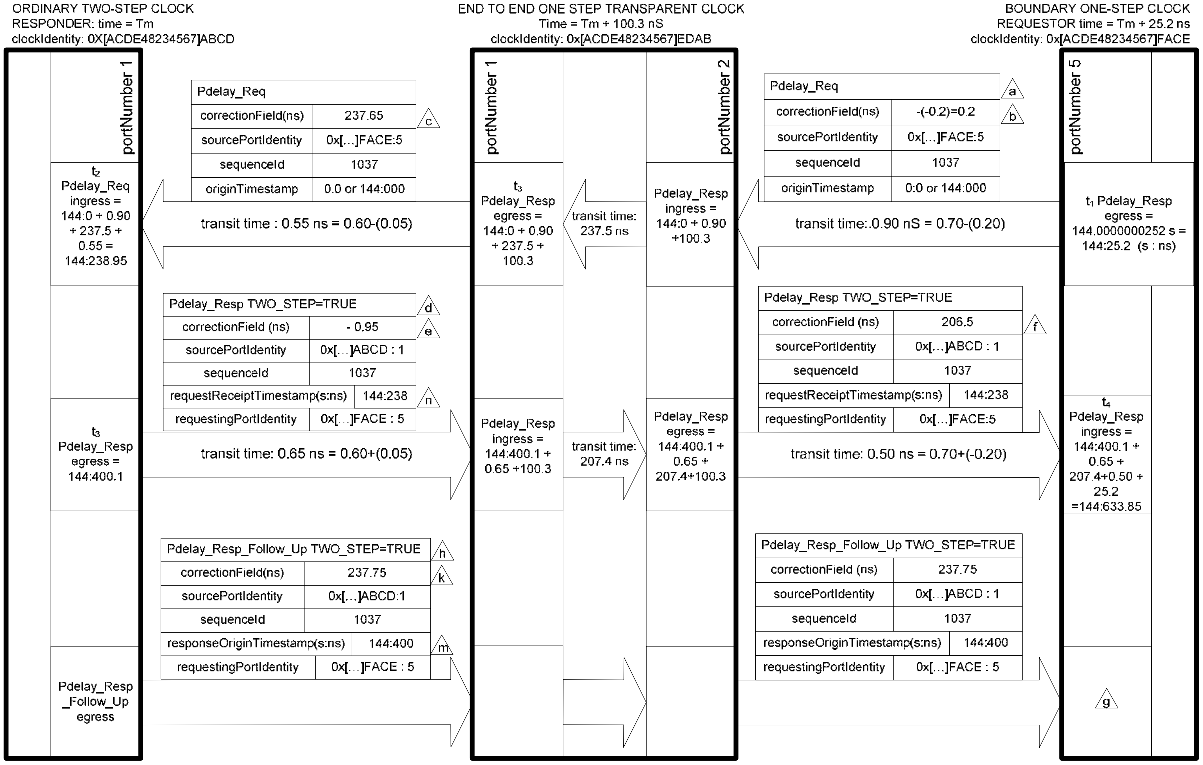
Table 120: Interpretation of Figure 46 key values

Key	Reference	Comments
a	11.4.3	The originTimestamp is 0 or and estimate of the egress timestamp t_1
b	11.4.3 & 11.6.4	The correctionField is 0 – (– 0.2) = 0.2 where (– 0.2) is the asymmetry on the egress path.
c	11.4.3 & 11.5.4.3 & 11.6.4	The correctionField of the Pdelay_Req message is not modified.
d	11.4.3	requestingPortIdentity and sequenceId fields are copied from the sourcePortIdentity and sequenceId fields of the Pdelay_Req message. The TWO-STEP flag is set to TRUE.
e	11.4.3	The correctionField is 0 less the fractional nanoseconds portion of t_2, i.e. 0 – 0.95 = – 0.95
f	11.4.3 & 11.5.4.3 & 11.6.5	The requestReceiptTimestamp is set to the seconds and nanoseconds portion of t_2, 144:238.
g	11.4.3 & 11.6.5	Prior to computing the meanPathDelay the correctionField of the Pdelay_Resp message is modified by adding the ingress asymmetry – 0.2. i.e. – (0.95) + (– 0.2) = –1.15. meanPathDelay = [(t_4 – t_1) – (responseOriginTimestamp – requestReceiptTimestamp) – correctionField of Pdelay_Resp – correctionField of Pdelay_Resp_Follow_Up]/2 = [(144:633.85 – 144:25.2) – (144:400 – 144:238) – (– 0.95 + (– 0.20)) – (445.2)]/2 = 1.3
h	11.4.3	The sourcePortIdentity, sequenceId, and requestingPortIdentity fields of the

		Pdelay_Resp message are copied into the same fields of the Pdelay_Resp_Follow_Up message. The TWO-STEP flag is set to TRUE.
k	11.4.3 & 11.6.5	The correctionField is set to the correctionField of the Pdelay_Req message plus the fractional nanoseconds portion of $t_{3,}$ i.e. 0.2 + 0.1 = 0.3
m	11.5.4.3, 11.6.4 & 11.6.5	The correctionField is the sum of the original correctionField and the residence times of the Pdelay_Req and Pdelay_Resp messages minus the egress asymmetry for the Pdelay_Req plus the ingress asymmetry for the Pdelay_Resp messages, i.e. 0.3 +237.5 + 207.4 – (0.05) + (0.05) = 445.2
p	11.4.3	The responseOriginTimestamp is set to the seconds and nanoseconds portion of t_3, 144:400.

C.3.5 One-step peer requestor, one-step end-to-end transparent, and two-step peer responder clocks showing residence time and asymmetry computations: option 2

Figure 47 illustrates a one-step peer requestor clock interacting with a one-step end-to-end transparent and two-step peer responder clocks. The two-step responder uses the second option of 11.4.3 in generating the Pdelay_Resp and Pdelay_Resp_Follow_Up messages.


ORDINARY TWO-STEP CLOCK
RESPONDER: time = Tm
clockIdentity: 0X[ACDE48234567]ABCD
END TO END ONE STEP TRANSPARENT CLOCK
Time = Tm + 100.3 nS
clockIdentity: 0x[ACDE48234567]EDAB
BOUNDARY ONE-STEP CLOCK
REQUESTOR time = Tm + 25.2 ns
clockIdentity: 0x[ACDE48234567]FACE
portNumber 1
portNumber 1
portNumber 2
portNumber 5
Pdelay_Req
correctionField(ns) 237.65
sourcePortIdentity 0x[…]FACE:5
sequenceId 1037
originTimestamp 0.0 or 144:000
c
Pdelay_Req
correctionField(ns) -(-0.2)=0.2
sourcePortIdentity 0x[…]FACE:5
sequenceId 1037
originTimestamp 0:0 or 144:000
a
b
t₂ Pdelay_Req ingress = 144:0 + 0.90 + 237.5 + 0.55 = 144:238.95
transit time : 0.55 ns = 0.60-(0.05)
t₃ Pdelay_Resp egress = 144:0 + 0.90 + 237.5 + 100.3
transit time: 237.5 ns
Pdelay_Resp ingress = 144:0 + 0.90 +100.3
transit time:.0.90 nS = 0.70-(0.20)
t₁ Pdelay_Resp egress = 144.0000000252 s = 144:25.2 (s : ns)
Pdelay_Resp TWO_STEP=TRUE
correctionField (ns) - 0.95
sourcePortIdentity 0x[…]ABCD : 1
sequenceId 1037
requestReceiptTimestamp(s:ns) 144:238
requestingPortIdentity 0x[…]FACE : 5
d
e
n
Pdelay_Resp TWO_STEP=TRUE
correctionField (ns) 206.5
sourcePortIdentity 0x[…]ABCD : 1
sequenceId 1037
requestReceiptTimestamp(s:ns) 144:238
requestingPortIdentity 0x[…]FACE:5
f
t₃ Pdelay_Resp egress = 144:400.1
transit time: 0.65 ns = 0.60+(0.05)
Pdelay_Resp ingress = 144:400.1 + 0.65 +100.3
transit time: 207.4 ns
Pdelay_Resp egress = 144:400.1 + 0.65 + 207.4+100.3
transit time: 0.50 ns = 0.70+(-0.20)
t₄ Pdelay_Resp ingress = 144:400.1 + 0.65 + 207.4+0.50 + 25.2 =144:633.85
Pdelay_Resp_Follow_Up TWO_STEP=TRUE
correctionField(ns) 237.75
sourcePortIdentity 0x[…]ABCD:1
sequenceId 1037
responseOriginTimestamp(s:ns) 144:400
requestingPortIdentity 0x[…]FACE : 5
h
k
m
Pdelay_Resp_Follow_Up TWO_STEP=TRUE
correctionField (ns) 237.75
sourcePortIdentity 0x[…]ABCD : 1
sequenceId 1037
responseOriginTimestamp(s:ns) 144:400
requestingPortIdentity 0x[…]FACE : 5
Pdelay_Resp _Follow_Up egress
g

g mean_path_delay = [(144.633.85 – 144:25.2) – (144:400 – 144:238) – (206.5 + (– 0.2)) – (237.75)]/2 = 1.3 = [0.5 + 0.9 + 0.65 + 0.55]/2

Figure 47: Two-step peer responder, one-step end-to-end transparent, and one-step peer requestor clocks: option 2- with asymmetry correction

The interpretations of key values for Figure 47 are given in Table 121.

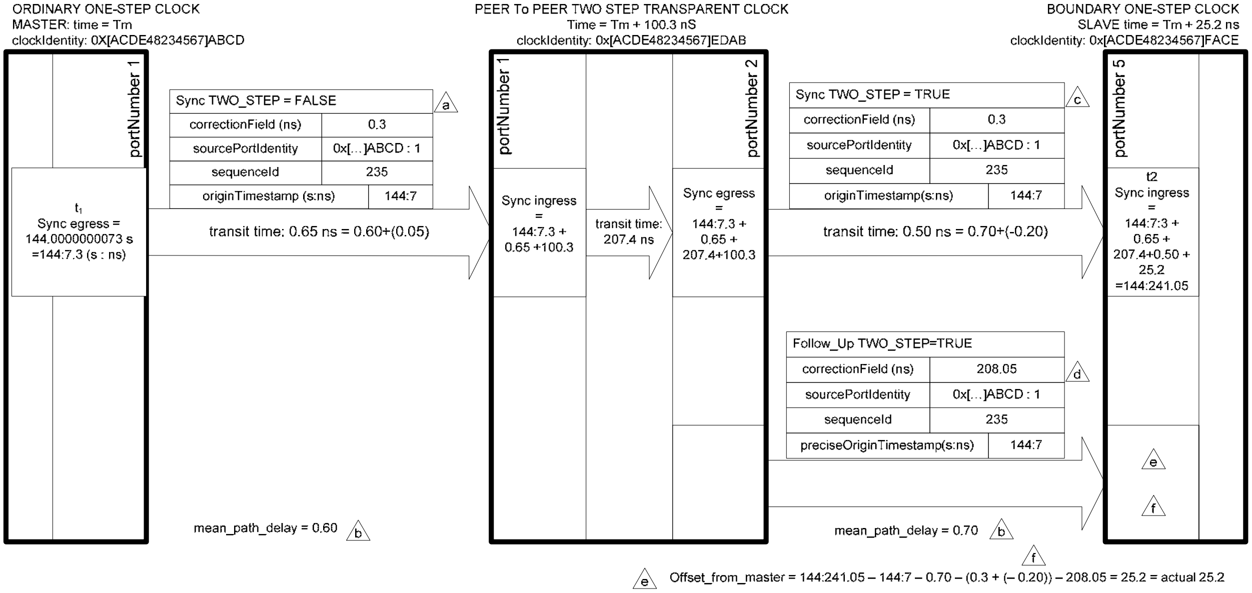
Table 121: Interpretation of Figure 47 key values

Key	Reference	Comments

a	11.4.3	The originTimestamp is 0 or and estimate of the egress timestamp t_1
b	11.4.3 & 11.6.4	The correctionField is 0 – (– 0.2) = 0.2 where (– 0.2) is the asymmetry on the egress path.
c	11.5.4.2 & 11.6.4	The correctionField is the sum of the original correctionField and residence time of the Pdelay_Req message less egress path asymmetry. 0.2 + 237.5 – 0.05 = 237.65
d	11.4.3	requestingPortIdentity and sequenceId fields are copied from the sourcePortIdentity and sequenceId fields of the Pdelay_Req message. The TWO-STEP flag is set to TRUE
e	11.4.3	The correctionField is set to 0 less the fractional nanoseconds portion of t_2. = – 0.95
f	11.5.4.2 & 11.6.5	The correctionField is the sum of the original correctionField and the residence time of the Pdelay_Resp message + the ingress asymmetry for the Pdelay_Resp messages. – 0.95 +207.4 + (0.05) = 206.5
g	11.4.3 & 11.6.5	Prior to computing the meanPathDelay the correctionField of the Pdelay_Resp message is modified by adding the ingress asymmetry – 0.2. i.e. 206.5 + (– 0.2) = 206.3. meanPathDelay = [(t_4 – t_1) – (responseOriginTimestamp – requestReceiptTimestamp) – correctionField of Pdelay_Resp – correctionField of Pdelay_Resp_Follow_Up]/2 = [(144:633.85 – 144:25.2) – (144:400 – 144:238) – (206.5 + (– 0.2)) – (237.75)]/2 = 1.3
h	11.4.3	The sourcePortIdentity, sequenceId, and requestingPortIdentity fields of the Pdelay_Resp message are copied into the same fields of the Pdelay_Resp_Follow_Up message. The TWO-STEP flag is set to TRUE.
k	11.4.3 & 11.6.5	The correctionField is set to the correctionField of the Pdelay_Req message plus the fractional nanoseconds portion of t_3, i.e.237.65 + 0.1 = 237.75
m	11.4.3	The responseOriginTimestamp is set to the seconds and nanoseconds portion of t_3, 144:400.
n	11.4.3 & 11.6.5	The requestReceiptTimestamp is set to the seconds and nanoseconds portion of t_2, 144:238.

C.3.6 One-step peer master, two-step peer-to-peer transparent, and one-step peer slave clocks showing time transfer from master to slave

Figure 48 illustrates a one-step peer master clock interacting with a two-step peer-to-peer transparent and one-step peer slave clocks to transfer time from the master to the slave.


ORDINARY ONE-STEP CLOCK
MASTER: time = Tm
clockIdentity: 0X[ACDE48234567]ABCD
PEER To PEER TWO STEP TRANSPARENT CLOCK
Time = Tm + 100.3 nS
clockIdentity: 0x[ACDE48234567]EDAB
BOUNDARY ONE-STEP CLOCK
SLAVE time = Tm + 25.2 ns
clockIdentity: 0x[ACDE48234567]FACE
portNumber 1
a
t1
Sync egress = 144.0000000073 s =144:7.3 (s : ns)
transit time: 0.65 ns = 0.60+(0.05)
portNumber 1
Sync ingress = 144:7.3 + 0.65 +100.3
transit time: 207.4 ns
portNumber 2
Sync egress = 144:7.3 + 0.65 + 207.4+100.3
c
transit time: 0.50 ns = 0.70+(-0.20)
portNumber 5
t2
Sync ingress = 144:7:3 + 0.65 + 207.4+0.50 + 25.2 =144:241.05
d
e
f
mean_path_delay = 0.60
b
mean_path_delay = 0.70
b
f
e
Offset_from_master = 144:241.05 – 144:7 – 0.70 – (0.3 + (– 0.20)) – 208.05 = 25.2 = actual 25.2

Sync TWO_STEP = FALSE	
correctionField (ns)	0.3
sourcePortIdentity	0x[...]ABCD : 1
sequenceId	235
originTimestamp (s:ns)	144:7

Sync TWO_STEP = TRUE	
correctionField (ns)	0.3
sourcePortIdentity	0x[...]ABCD : 1
sequenceId	235
originTimestamp(s:ns)	144:7

Follow_Up TWO_STEP=TRUE	
correctionField (ns)	208.05
sourcePortIdentity	0x[...]ABCD : 1
sequenceId	235
preciseOriginTimestamp(s:ns)	144:7

Figure 48: One-step peer master, two-step peer-to-peer transparent, and one-step peer slave clocks: time computation

The interpretations of key values for Figure 48 are given in Table 122.

Table 122: Interpretation of Figure 48 key values

Key	Reference	Comments
a	9.5.9.3	The originTimestamp is the seconds and nanoseconds portion of the egress timestamp t_1. The sum of the originTimestamp and the correctionField is the exact value of t_1.
b	11.4.3	The meanPathDelay values for both links are determined using the peer delay mechanism to be 0.60 and 0.70 respectively.
c	11.5.2.2	The TWO-STEP flag is set to TRUE prior to transmitting the Sync message.
d	11.5.2.2 & 11.4.5.1 & 11.6.2	sourcePortIdentity and sequenceId fields are copied from the sourcePortIdentity and sequenceId fields of the Sync message. The preciseOriginTimestamp field is set to the value of the originTimestamp field. The correctionField is first set to the residence time 207.4 of the Sync message. This value is then further corrected by adding the meanPathDelay on the Sync ingress port path, 207.4 + 0.6. The result is then further corrected by adding the Sync ingress port path asymmetry, 207.4 + 0.6 + 0.05 = 208.05.
e	11.2	The offsetFromMaster = t_2 – preciseOriginTimestamp – meanPathDelay on ingress link – correctionField of Sync message – correctionField of Follow_Up message. OffsetFromMaster = 144:241.05 – 144:7 – 0.70 – (0.3 + (–0.20)) – 208.05 = 25.2
f	11.6.2	The correctionField of the received Sync message is modified by adding the value of the ingress path asymmetry prior to any use in computation "e". 0.3 + (– 0.20) = 0.1

Annex D
(normative)
Transport of PTP over User Datagram Protocol over Internet Protocol Version 4

D.1 General

Annex D specifies those portions of the PTP standard that are specific to implementations that transport messages over the User Datagram Protocol (UDP) as defined in IETF RFC 768 [M14], and Internet Protocol version 4 (IPv4), as defined in IETF RFC 791 [M15]. The specifications in this annex shall apply to all PTP implementations using UDP/IPv4 as a communication service.

The first octet of the PTP message shall immediately follow the final octet of the UDP header.

The transmitting or intermediate node may set the UDP checksum to 0.

When using this transport with unicast transmission, modifications to PTP event packets by transparent clocks may corrupt applications that incorrectly use the PTP destination port.

NOTE—The UDP destination ports below are reserved values assigned to PTP and no interference should occur. However it is known that there are applications in use that disregard these assignments. It is these applications that are vulnerable to the action of transparent clocks.

D.2 UDP port numbers

The UDP destination port of an event message shall be 319[1].

The UDP destination port of a multicast general message shall be 320.

The UDP destination port of a unicast general message that is addressed to a clock shall be 320.

The UDP destination port of a unicast general message that is addressed to a manager shall be the UDP source port value of the PTP message to which this is a response.

D.3 IPv4 multicast addresses

PTP messages shal use the multicast message specified in table 122

Table 123 IPv4 multicast addresses

IANA assigned Name[2]	Message types	Address
PTP-primary	All except peer delay mechanism messages	224.0.1.129
PTP-pdelay	Peer delay mechanism	224.0.0.107

[1] The Internet Assigned Numbers Authority (IANA) asigned the dedicated ports numbers shown, see http://www.iana.org/assignments/port-numbers.

[2] The Internet Assigned Numbers Authority (IANA) assigned the dedicated multicast addresses along with the IANA Names. These names appear in the IANA listings identifying multicast addresses and names.

	messages	

For messages sent to the PTP-pdelay address, the Time to Live (TTL) field shall be set to 1.

D.4 transportSpecific field values

The transportSpecific field, see 13.3.2.1, shall be interpreted as specified in Table 123.

Table 124 transportSpecific field values

Bit	Name	Meaning
0	Version 1 Hardware Compatibility	Some Version 1 implementations of hardware assist timestamping check the length of the incoming packet before qualifying the timestamp and require the UDP payload of the PTP event messages to be at least 124 octets in length. Nodes using such hardware shall set bit 0 equal to '1' in all Announce, and PTP event messages transmitted from the node. The receiver of any PTP Announce or event message with bit 0 equal to '1', shall extend the UDP payload of all PTP event messages transmitted to the receiving node, so that the UDP payload length equals 124 octets. The padding octets shall have all bits zero. This padding shall be added to all transmitted PTP event messages to the receiving node, for a time duration equal to the value of portDS.announceReceiptTimeout seconds since the last PTP Announce or event message received from that node with bit 0 equal to '1'. The padding shall be added irrespective of whether the PTP event messages are transmitted using a multicast or a unicast model. If the transmitter is not making a request for padding, the bit shall be transmitted as zero. Except as required for backward compatibility with some version 1 hardware, nodes shall disregard the padding octets. See NOTE.
1-3	Reserved	The bit shall be transmitted as zero and ignored by the receiver.
NOTE-This specification can result in nodes that do not require the padding receiving padded event messages		

D.5 Optional values

For PTP event messages, the value of the differentiated service (DS) field in the Type of Service (ToS) field should be set to the highest traffic class selector codepoint available.

NOTE—When the layer 2 transport mechanism allows for multiple priorities, it is recommended that the highest priority be used for event messages.

D.6 IPv4 Options

IPv4 options shall not be used.

D.7 Protocol addresses

For any quantity of data type PortAddress, see 5.3.6, when the networkProtocol member value is UDP/IPv4, see 7.4.1:

— The addressLength member value shall be 4, and

— The address member value shall be the IPv4 address of the port represented as four groups of two hexadecimal digits. For example the IPv4 address 207.142.131.235 expressed in the usual text notation, appear as the octet array CF8E83EB$_{16}$

Annex E
(normative)
Transport of PTP over User Datagram Protocol over Internet Protocol Version 6

E.1 General

Annex E specifies those portions of the PTP standard that are specific to implementations that transport messages over the User Datagram Protocol (UDP) as defined in IETF RFC 768 [M14], and Internet Protocol version 6 (IPv6), as defined in IETF RFC 2460 [M11]. The specifications in this annex shall apply to all PTP implementations using UDP/IPv6 as a communication service.

The first octet of the PTP message shall immediately follow the final octet of the UDP header.

A transmitting node shall extend the UDP payload of all PTP messages by two octets beyond the end of the PTP message. The contents of the UDP checksum field or the final two octets of the UDP payload may be modified by the initiator or an intermediate node to ensure that the UDP checksum remains uncompromised after any modification of PTP fields. This modification to update the UDP checksum may be implemented using the mechanism defined in IETF RFC 1624 [M8]. Other than for purposes of calculating the UDP checksum, the contents of the UDP field beyond the end of the PTP fields shall be ignored by the receiver.

E.2 UDP port numbers

The UDP destination port value of an event message shall be 319[3].

The UDP destination port value of a multicast general message shall be 320.

The UDP destination port value of a unicast general message that is addressed to a clock shall be 320.

The UDP destination port value of a unicast general message that is addressed to a manager shall be the UDP source port value of the PTP message to which this is a response.

E.3 IPv6 multicast addresses

Table 125 IPv6 Multicast Addresses

IANA assigned Name	Message Types	Address (hex)
PTP-primary	All except peer delay mechanism messages	FF0X:0:0:0:0:0:0:181 see NOTE
PTP-pdelay	Peer delay mechanism messages	FF02:0:0:0:0:0:0:6B
NOTE – The hexidecial values for 'X' in the PTP-primary address are defined in RFC 4291 [M13]. These are: 0 reserved 1 Interface-Local scope 2 Link-Local scope 3 reserved		

[3] The Internet Assigned Numbers Authority (IANA) asigned the dedicated ports numbers shown, see http://www.iana.org/assignments/port-numbers.

4	Admin-Local scope
5	Site-Local scope
6	(unassigned)
7	(unassigned)
8	Organization-Local scope
9	(unassigned)
A	(unassigned)
B	(unassigned)
C	(unassigned)
D	(unassigned)
E	Global scope
F	reserved

For messages sent to the PTP-pdelay address, the Hop Limit (HL) field shall be set to 1.

E.4 transportSpecific field values

All bits of the transportSpecific field, see 13.3.2.1, shall be transmitted as zero and ignored by the receiver..

E.5 Optional values

For PTP event messages, the value of the Differentiated Service (DS) field in the Traffic Class (TC) field should be set to the highest traffic class selector codepoint available.

NOTE 1— When the layer 2 transport mechanism allows for multiple priorities, it is recommended that the highest priority be used for event messages.

NOTE 2— The use of IPv6 Extension Headers is outside the scope of this standard.

E.6 Protocol addresses

For any quantity of data type PortAddress, see 5.3.6, when the networkProtocol member value is UDP/IPv6, see 7.4.1:

— The addressLength member value shall be 16, and

— The address member value shall be the IPv6 address of the port represented as 16 groups of two hexadecimal digits. For example the IPv6 address 2001:0DB8:85A3:08D3:1332:8A2E:0270:7225 expressed in the usual text notation per IETF RFC 4291 [M13], appear as the octet array 20010DB885A308D313328A2E02707225.

Annex F
(normative)
Transport of PTP over IEEE 802.3 /Ethernet

F.1 General

Annex F specifies those portions of the PTP standard that are specific to implementations that transport messages directly over Ethernet frames as specified in IEEE Std 802.3:2005. .

The first Octet of the PTP message shall occupy the first octet of the client data field.

F.2 Ethertype

The specifications in this annex shall apply to all PTP implementations directly using Ethernet format packets with the 88F7$_{16}$ EtherType as a communication service.

F.3 Multicast MAC Addresses

By default PTP messages shall use MAC addresses as specified in Table 126

Table 126 Multicast MAC addresses

Message types	Address (hex)
All except peer delay mechanism messages	01-1B-19-00-00-00
Peer delay mechanism messages	01-80-C2-00-00-0E

The OUI value of 00-1B-19 represents the value assigned to this standard by the IEEE/RAC. The MAC address value of 01-1B-19-00-00-00 represents a multicast address derived from the pool of multicast addresses within that space.

The MAC address of 01-80-C2-00-00-0E represents a multicast address derived from the pool of multicast addresses administered by IEEE 802.1. It is permissible, however to use address 01-1B-19-00-00-00 or address 01-80-C2-00-00-0E for all PTP messages, if such use is defined in a PTP profile.

To ensure peer delay measurements on ports blocked by (Rapid/Multiple) Spanning Tree Protocols a reserved address, 01-80-C2-00-00-0E, shall be used as a Destination MAC Address for PTP peer delay mechanism messages.

NOTE 1—Per 8.6.3 of IEEE Std 802.1Q-2005, frames containing reserved addresses in their destination address field are not relayed by the bridge.

NOTE 2—At its July 17 - 20, 2006 meeting the IEEE 802.1 working group approved a motion that included the following text: 're-designate the reserved address currently identified for use by 802.1AB as an address that can be used by protocols that require the scope of the address to be limited to an individual LAN' and 'The reserved multicast address that IEEE 1588 should use is 01-80-C2-00-00-0E'. This address is not reserved exclusively for PTP, but rather is a shared address.

Per port peer delay measurements shall use the egress port's MAC Address as the source MAC Address in PTP peer delay mechanism messages.

F.4 transportSpecific field values

The transportSpecific field, see 13.3.2.1, shall be interpreted as a subtype of the Ethertype as defined in Table 127.

If the device recognizes the subtype then the message is passed to the PTP layer. If the device does not recognize the subtype then the message is treated as any other message with an unrecognized Ethertype.

Table 127: Ethernet transport specific field

Enumeration	**Value**(hex)	**Specification**
DEFAULT	0	All PTP layer 2 Ethernet transmissions not covered by another enumeration value.
ETHERNET_AVB	1	This value is reserved for use in connection with the standard being developed by the 802.1 AVB Task Group as P802.1AS .
Reserved	2 – F	Reserved for assignment in future versions of this standard.

F.5 Optional values

When the Ethernet transport mechanism allows for multiple priorities, it is recommended that the highest priority be used for event messages.

Note — On Ethernet, the IEEE 802.1Q discusses the implementation of priorities.

F.6 Protocol addresses

For any quantity of data type PortAddress, see 5.3.6, when the networkProtocol member value is IEEE 802.3, see 7.4.1:

— The addressLength member value shall be 6, and

— The address member value shall be the six octet source address field of the Ethernet header.

Annex G
(normative)
Transport of PTP over DeviceNET

G.1 Protocol

Annex G specifies those portions of the PTP standard that are specific to DeviceNet implementations. The specifications in this annex shall apply to all PTP implementations using DeviceNet as a communication network. For additional information on DeviceNet, consult the DeviceNet specification provided by ODVA (Open DeviceNet Vendors Association, http://www.odva.org).

NOTE— DeviceNet is also covered by IEC 62026-3.

G.2 Event message timestamp point

The event message timestamp point, see 7.3.4.1, shall correspond to the trailing edge of the sixth bit of the end of frame field of the first fragmented packet of a PTP event message as shown in Figure 49.

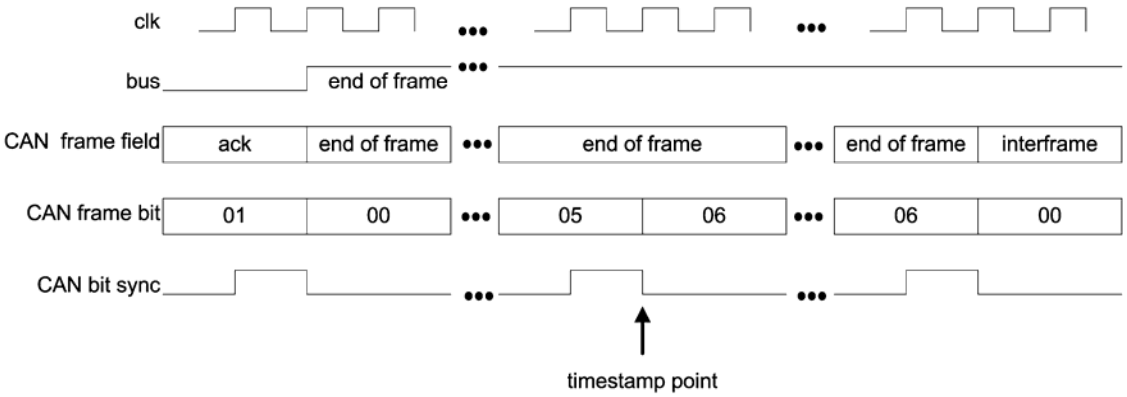


Figure 49: Event message timestamp point

G.3 clockIdentity

The clockIdentity octets [0] thru [7], see 7.5.2.2.3, shall be the combination of the node's vendor ID and serial number uniquely associated with the node.

Example:

— The vendor ID for Company X is AC7B (hex). If Company X wished to create a DeviceNet device, a legal value for the clockIdentity would be: 0201AC7BF2235C01 (hex) where the 4 octet array F2235C01 (hex) would be guaranteed by Company X to be unique among all Company X assigned DeviceNet numbers.

Table 128 depicts the layout of the clockIdentity for octets 2 through 7. Octets [0] and [1] are specified in 7.5.2.2.3.

Table 128: DeviceNet clockIdentity Octets 0 through 7

technology	vendorId	serialNumber	field

								octet order hex
00000010	00000010	10101100	00111011	11110010	00100011	01011100	00000001	octet
| | | most significant byte most significant bit							| | least significant byte | least significant bit	order hex bits

G.4 PTP message formats

PTP messages are transmitted with the most significant byte of a data type transmitted first followed sequentially by bytes in order of decreasing significance. The first octet of the PTP message shall immediately follow the final octet of the DeviceNet header.

This data is sent using multiple DeviceNet packets (frames) following the standard DeviceNet Explicit Message fragmentation logic. All PTP messages are fragmented on DeviceNet. The DeviceNet header is present in all packets.

DeviceNet headers for all PTP message packets are specified in Table 129. This header is present in each DeviceNet frame of the PTP message

Table 129: DeviceNet headers for all PTP message packets

Octet 0	Octet 1	Octet 2	Type (informative)	Field name
h_0h_1	j_0j_1	k_0k_1	octet | octet | octet	Fragment = 1, XID = 0, Source MACID | Fragment Type, Fragment Count | R/R = 1, Service Code = UCMM Service Code

G.5 DeviceNet addressing for PTP

All PTP messages shall be transmitted by a UnConnect Message Manager (UCMM) capable device as an Unconnected Response Message (Message Group 3, Message ID5) and by a Group 2 Only server as an Unconnected Response Message (Message Group 2, Message ID 3). Thus, each node on the subnet has its own unique multicast address (Controller Area Network (CAN) identifier). The same multicast address is used for all Domains.

All PTP messages shall have the Request/Response bit in the DeviceNet header set to TRUE.

The PTP multicast addresses are shared with other DeviceNet functions, some of which are point to point messages. To distinguish a PTP message, the transmitting node shall place its own node address in the Destination Node field of the DeviceNet message header. The message is then further identified as a PTP message through the use of the UCMM service code.

The UCMM service code field shall be 88 (58_{16}) for the event class of messages and 89 (59_{16}) for the General class of messages.

For any quantity of data type PortAddress, see 5.3.6, when the networkProtocol member value is DeviceNet, see 7.4.1:

— The addressLength member value shall be 2, and

— The address member value shall be the devicenet mac ID

G.6 transportSpecific field values

All bits of the transportSpecific field, see 13.3.2.1, shall be transmitted as zero and ignored by the receiver.

Annex H
(normative)
Transport of PTP over ControlNET

H.1 Protocol

Annex H specifies those portions of the PTP standard that are specific to ControlNet implementations. The specifications in this annex shall apply to all PTP implementations using ControlNet as a communication network. For additional information on ControlNet, consult the ControlNet specification provided by ControlNet International (http://www.controlnet.org).

NOTE— ControlNet is also covered by IEC 61158, type 2 elements.

H.2 clockIdentity

The clockIdentity octets [2] thru [7], see 7.5.2.2.3, shall be the combination of the node's vendor ID and serial number uniquely associated with the node.

Example:

— The vendor ID for Company X is AC7B (hex). If Company X wished to create a ControlNet device, a legal value for the clockIdentity would be: 0202AC7BF2235C01 (hex) where the 4 octet array F2235C01 (hex) would be guaranteed by Company X to be unique among all Company X assigned ControlNet numbers.

Table 130 depicts the layout of the clockIdentity for octets 2 through 7. Octets [0] and [1] are specified in 7.5.2.2.3.

Table 130: ControlNet clockIdentity octets 2 through 7

<table>
<tr><th colspan="2">technology</th><th colspan="2">vendorId</th><th colspan="4">serialNumber</th><th>field</th></tr>
<tr><td>0
02</td><td>1
02</td><td>2
AC</td><td>3
7B</td><td>4
F2</td><td>5
23</td><td>6
5C</td><td>7
01</td><td rowspan="2">octet order
hex
bits</td></tr>
<tr><td>00000010</td><td>00000010</td><td>10101100</td><td>01111011</td><td>11110010</td><td>00100011</td><td>01011100</td><td>00000001</td></tr>
<tr><td colspan="9">| | | |
| most significant byte least significant byte |
most significant bit least significant bit</td></tr>
</table>

H.3 PTP message formats

PTP messages are transmitted with the most significant byte of a data type transmitted first followed sequentially by bytes in order of decreasing significance. The first octet of the PTP message shall immediately follow the final octet of the ControlNet LPacket header.

H.4 ControlNet addressing for PTP

The Destination address field for a PTP LPacket shall be 255(FF_{16}) (Broadcast).

The Fixed Tag field for a PTP LPacket shall be 141 ($8D_{16}$) for event messages and 142($8E_{16}$) for general of messages.

For any quantity of data type PortAddress, see 5.3.6, when the networkProtocol member value is ControlNet, see 7.4.1:

— The addressLength member value shall be 2, and

— The address member value shall be the controlNet node number of the device.

H.5 transportSpecific field values

All bits of the transportSpecific field, see 13.3.2.1, shall be transmitted as zero and ignored by the receiver.

Annex I
(normative)
Transport of PTP over IEC 61158 Type 10

I.1 Background

PROFINET (IEC 61158 Type 10) specifies a fieldbus communication system. More specific information on how this fieldbus communication system is used to interoperate in a system is given in the communication profiles IEC 61784-1 and IEC 61784-2.

IEC 61784-1 and IEC 61784-2 specify Communication Profile Families (CPF) and, within a CPF, one or more Communication Profiles (CP). A CP refers to IEC 61158 Types. IEC 61784-1 specifies various fieldbusses. IEC 61784-2 specifies various real-time Ethernet fieldbusses. PROFIBUS and PROFINET are specified in CPF 3. CP 3/4, CP 3/5, and CP 3/6 specify PROFINET in IEC 61784-2.

The IEC 61158 Type 10 protocol is specified in IEC 61158-6-10. IEC 61158 Type 10 services are specified in IEC 61158-5-10.

This Annex specifies the protocol over Layer 2 for the CP 3/4, CP 3/5, and CP 3/6 of IEC 61784-2, also known as PROFINET. These CPs refer to IEC 61158-5-10, IEC 61158-6-10, and other standards.

Figure 50 illustrates a PTP region and an IEC 61158 Type 10 region. A boundary clock is used to translate between the protocol in the two regions.

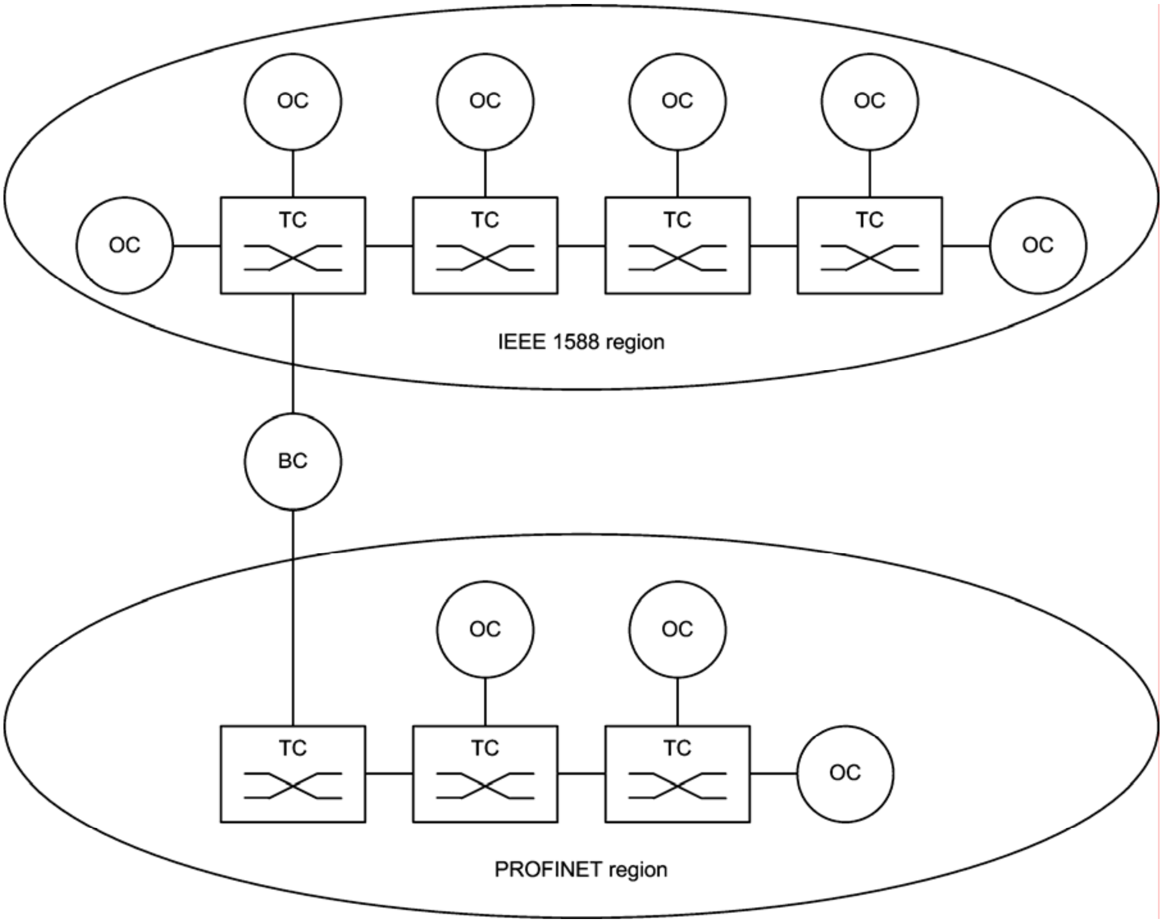


Figure 50: PROFINET region combined with domains

The protocol of this Annex is functionally equivalent to transparent and ordinary clock functionality of PTP over Layer 2 in the main subclauses and annexes of this standard. However, the protocol of this Annex has a different encoding of the PTP messages to meet the encoding specifications for CP 3/4, CP 3/5, and CP 3/6 of IEC 61784-2 within IEC 61158. This Annex is not applicable to CP 3/1, CP 3/2, and CP 3/3 of IEC 61784-1.

NOTE— Existing ASICs support the PTP over Layer 2 of CP 3/4, CP 3/5, and CP 3/6 of IEC 61784-2.

The encoding of this Annex shall be used for implementations required to meet the encoding specifications for CP 3/4, CP 3/5, and CP 3/6 of IEC 61784-2 within IEC 61158.

I.2 Message specification

The mappings of the different message names are provided in Table 131.

Table 131: Mapping of messages

Names for PROFINET	Names for PTP
SyncPDU	Sync
FollowUpPDU	Follow_Up
AnnouncePDU	Announce
Not used	Delay_Req

Not used	Delay_Resp
DelayReqPDU	Pdelay_Req
DelayResPDU	Pdelay_Resp
DelayFuResPDU	Pdelay_Resp_Follow_Up
Not used	Signaling
Not used	Management

The coding of the PROFINET messages and the used acronyms, abbreviations and conventions shall be used according IEC 61158-5-10 and IEC 61158-6-10.
For any quantity of data type PortAddress, see 5.3.6, when the networkProtocol member value is PROFINET, see 7.4.1:

— The addressLength member value shall be ???, and

— The address member value shall be <<add the following instructions on this issue- FRANZ or LUDWIG- please draft text>>

I.3 DLPDU of the IEC 61158 TYPE10

I.3.1 Abstract syntax of the DLPDU

Table 132 gives an outline of the abstract syntax of the DLPDU according to IEEE 802.3.

The encoding and decoding of the fields in Table 132 shall be according to IEEE 802.3 for the DLPDU.

Table 132: IEEE 802.3 DLPDU syntax

DLPDU name	DLPDU structure
DLPDU	Preamble [a], StartFrameDelimiter, DestinationAddress, SourceAddress, DLSDU [b], DLPDU_Padding* [c], FrameCheckSequence
DLSDU	LT, FIDAPDU
FIDAPDU	FrameID, SyncPDU ^ AnnouncePDU ^ FollowUpPDU ^ DelayReqPDU ^ DelayResPDU ^ DelayFuResPDU
NOTE— According to IEEE 802.3 the DLPDUs have a minimum length of 64 octets (excluded Preamble, Start Frame Delimiter).	
[a] The field contains at least 7 octets [b] The minimum DLSDU size is 2 octets. [c] The number of padding octets shall be in the range of 0..46 depending on the DLSDU size. The value shall be set to zero.	

I.3.2 Coding of the DLPDU field DestinationAddress

The DLPDU field shall be coded as data type Octet[6]. The value of the field DestinationAddress shall be an IEEE 802 MAC address.

For PTP over PROFINET-PDUs, the value shall be set according to Table 133.

Table 133: Multicast-MAC-Address

Group MAC Address	Meaning
01-0E-CF00-01-02	In conjunction with SyncPDU and FrameID (=0080$_{16}$) used for clock synchronization during reserved Period

Group MAC Address	**Meaning**
01-0E-CF-00-04-00	In conjunction with SyncPDU, AnnouncePDU and FrameID (=0000_{16}, =$FF00_{16}$) used for clock synchronization
01-0E-CF-00-04-01	In conjunction with SyncPDU, AnnouncePDU and FrameID (=0001_{16}, =$FF01_{16}$) used for time synchronization
01-0E-CF-00-04-xx	In conjunction with SyncPDU, AnnouncePDU and FrameID (=$00xx_{16}$, =$FFxx_{16}$) used for synchronization
01-0E-CF-00-04-1F	In conjunction with SyncPDU, AnnouncePDU and FrameID (=$001F_{16}$, =$FF1F_{16}$) used for synchronization
01-0E-CF-00-04-20	In conjunction with FollowUpPDU and FrameID (=$FF20_{16}$) used for clock synchronization
01-0E-CF-00-04-21	In conjunction with FollowUpPDU and FrameID (=$FF21_{16}$) used for time synchronization
01-0E-CF-00-04-xx	In conjunction with FollowUpPDU and FrameID (=$FFxx_{16}$) used for synchronization
01-0E-CF-00-04-3F	In conjunction with FollowUpPDU and FrameID (=$FF3F_{16}$) used for synchronization
01-80-C2-00-00-0E see NOTE 3	In conjunction with DelayReqPDU and FrameID (=$FF40_{16}$), DelayResPDU with follow up and FrameID (=$FF41_{16}$), DelayFuResPDU and FrameID (=$FF42_{16}$) and DelayResPDU without follow up and FrameID (=$FF43_{16}$) used for peer-to-peer delay measurement

NOTE 1—Octet 1 contains the Individual/Group Address Bit (LSB).

NOTE 2—The addresses in Table 132 that begin with 01-0E-CF are based on the OUI owned by PROFIBUS Nutzerorganisation e.V.

NOTE 3—The MAC address of 01-80-C2-00-00-0E represents a multicast address derived from the pool of multicast addresses administered by IEEE 802.1. See F.3 for detail on this address.

I.3.3 Coding of the field LT

The LT field shall be coded with the values according to IEEE 802.3 (Unsigned16). This specification uses the values according to Table 134.

Table 134: LT (Length/Type)

Value$_{16}$	**Meaning**
8892	PROFINET

I.3.4 Coding of the field FrameID

The FrameID field shall be coded as data type Unsigned16 with the values according to Table 135. This field identifies the structure and the type of the APDU.

Table 135: FrameID

Value$_{16}$	**Meaning**	**Use**
0000	SyncPDU	SyncPDU without follow up used for clock synchronization (isochronous application)
0001	SyncPDU	SyncPDU without follow up used for time synchronization
0003 – 001F	Reserved	
0020	SyncPDU	SyncPDU with follow up used for clock synchronization (isochronous application)
0021	SyncPDU	SyncPDU with follow up used for time synchronization

Value$_{16}$	**Meaning**	**Use**
0022 – 00FF	Reserved	
0080 – 0081	SyncPDU for reserved period	SyncPDU without follow up used for clock synchronization (isochronous application)
0082 – 00FF	Reserved	
FF00	AnnouncePDU (clock)	AnnouncePDU is used for clock synchronization (isochronous application)
FF01	AnnouncePDU (time)	AnnouncePDU is used for time synchronization
FF02 – FF1F	Reserved	
FF20	FollowUpPDU (clock)	FollowUpPDU is used for clock synchronization
FF21	FollowUpPDU (time)	FollowUpPDU is used for time synchronization
FF22 – FF3F	Reserved	
FF40	DelayReqPDU	DelayReqPDU is used for path delay measurement
FF41	DelayResPDU	DelayResPDU is used for path delay measurement with follow up
FF42	DelayFuResPDU	DelayFuResPDU is used for path delay measurement
FF43	DelayResPDU	DelayResPDU is used for path delay measurement without follow up
FF44 – FFFF	Reserved	

I.4 Encoding specifications

A bridge can convert the two formats at the edge. The mapping of the different formats and the different parameter and attribute names are provided in Table 136.

Table 136: Mapping of the parameter and attribute names

Names for PROFINET	Message type	Names for PTP version 2
No counterpart	—	transportSpecific
FrameID	SyncPDU FollowUpPDU AnnouncePDU DelayReqPDU DelayResPDU DelayFuResPDU	messageType
No counterpart	—	versionPTP
No counterpart	—	messageLength
SubdomainUUID	SyncPDU FollowUpPDU AnnouncePDU DelayReqPDU DelayResPDU DelayFuResPDU	domainNumber
According to Table 137	SyncPDU	flags
SequenceId	SyncPDU FollowUpPDU AnnouncePDU DelayReqPDU DelayResPDU DelayFuResPDU	sequenceId
No counterpart	—	controlField
MasterSourceAddress	SyncPDU FollowUpPDU AnnouncePDU	clockIdentity
Is specified in PROFINET	—	logMessageInterval
Seconds	SyncPDU	seconds (Bit 0..31)
NanoSeconds	SyncPDU	nanoseconds

EpochNumber	SyncPDU	seconds (Bit 32 ..47)
CurrentUTCOffset	SyncPDU	currentUTCOffset
No counterpart	—	timeSource
According to Table 139	SyncPDU AnnouncePDU	clockClass
According to Table 138	SyncPDU AnnouncePDU	priority1
According to Table 138	SyncPDU AnnouncePDU	priority2
ClockVariance	SyncPDU AnnouncePDU	offsetScaledLogVariance
No counterpart	—	stepsRemoved
No counterpart	—	grandmasterIdentity
No counterpart	—	parentPortIdentity
RequestSourceAddress	DelayReqPDU DelayResPDU DelayFuResPDU	clockIdentity
RequestPortID	DelayReqPDU DelayResPDU DelayFuResPDU	portNumber

Table 137: Translation of flags field from PTP version 2 to PROFINET

Names for PROFINET	Names for PTP version 2
Last minute has 61 seconds	LL_61
Last minute has 59 seconds	LL_59
Signaled by the AnnouncePDU	alternateMasterFlag
Coded in FrameID	twoStepFlag
TRUE for time synchronization and ClockStratum = 1 or 2	timeTraceable
TRUE for ClockStraum = 1 or 2	FREQUENCYTRACEABLE
FALSE for clock synchronization (ARP) TRUE for time synchronization if identifier is not INIT or DFLT. Otherwise FALSE	ptpTimescale
FALSE for clock synchronization (ARP) TRUE for time synchronization if identifier is not INIT or DFLT. Otherwise FALSE	CURRENTUTCOFFSETValid
FALSE	UNICASTFLAG
Set to FALSE	All other flag fields

Table 138: Translation of priority1 and priority2 from PTP version 2 to PROFINET

PROFINET		PTP version 2
ClockRole	**Meaning**	**grandmasterPriority1**
0	Reserved	—
1	Primary	128
2	Secondary	>128
3 – 255	Reserved	—

There is no counterpart to grandmasterPriority2 in PROFINET.

Table 139: Translation of clockClass from PTP version 2 to PROFINET

PROFINET	PTP version 2
Clock stratum	**clockClass**
0	0

1	6, 7, 13,14
2	10
3	15 – 248
4	251
255	255

The coding of the IEC 61158 Type 10 parameter, attributes and the used acronyms, abbreviations and conventions shall be used according IEC 61158-5-10 and IEC 61158-6-10.

Annex J
(normative)
Default PTP profiles

J.1 General

Each default PTP profile specifies a selection of options and attributes. Each selection specifies a system that works without requiring user configuration.

J.2 General requirements

Nodes shall implement all requirements in the respective PTP profile that specify default values or choices such that these default values or choices apply without requiring user configuration, i.e. as delivered from the manufacturer.

J.3 Delay Request-Response Default PTP profile

J.3.1 Identification

The identification values for this PTP profile, see 19.3.3 are:

PTP Profile
Default PTP profile for use with the delay request-response mechanism.
Version 1.0
Profile identifier: 00-1B-19-00-01-00

This profile is specified by the Precise Networked Clock Synchronization Working Group of the IM/ST Committee.

A copy may be obtained by ordering the standard IEEE 1588-2008 from the IEEE Standards Organization http://standards.ieee.org/

J.3.2 PTP attribute values

All nodes shall support the ranges and shall have the default initialization values for attributes as follows:

- domain: The default initialization value shall be 0.
- logAnnounceInterval: The default initialization value shall be 1. The configurable range shall be 0 to 4
- logSyncInterval: The default initialization value shall be 0. The configurable range shall be -1 to +1
- logMinPdelayReqInterval: The default initialization value shall be 0. The configurable range shall be 0 to 5.
- announceReceiptTimeout: The default initialization value shall be 3. The configurable range shall be in the range 2 to 10.
- priority1: The default initialization value shall be 128.
- priority2: The default initialization value shall be 128.
- slaveOnly: If this parameter is configurable the default value shall be FALSE.
- primaryDomain: The default initialization value shall be 0.

— τ, see 7.6.3.2: The default initialization value shall be 1.0 seconds.

For each defined range, manufacturers are free to allow wider ranges.

J.3.3 PTP Options

All options of Clauses 16 and 17 are permitted. By default these options shall be inactive unless specifically activated by a management procedure.

Node management shall implement the management message mechanism of this standard.

The best master clock algorithm shall be the algorithm specified by 9.3.2.

The delay request-response mechanism shall be the default path delay measurement mechanism. The peer delay mechanism may also be implemented.

NOTE— Only a single mechanism is allowed per link. Boundary clocks should be used between links that use different path delay mechanisms.

J.3.4 Clock physical requirements

J.3.4.1 Frequency accuracy

Every grandmaster clock shall maintain a frequency deviating no more than 0.01% from the SI second.

J.3.4.2 Frequency adjustment range

Any clock in the SLAVE state shall be able to correct its frequency to match any master clock meeting the requirements of J.3.4.1

NOTE—The frequency adjustment range should be at least ±0.025%.

J.4 Peer-to-Peer Default PTP profile

J.4.1 Identification

The identification values for this PTP profile, see 19.3.3 are:

PTP Profile
Default PTP profile for use with the delay request-response mechanism.
Version 1.0
Profile identifier: 00-1B-19-00-02-00

This profile is specified by the Precise Networked Clock Synchronization Working Group of the IM/ST Committee.

A copy may be obtained by ordering the standard IEEE 1588-2008 from the IEEE Standards Organization http://standards.ieee.org/

J.4.2 PTP attribute values

All nodes shall support the ranges and shall have the default initialization values for attributes as follows:

— domain: The default initialization value shall be 0.

— logAnnounceInterval: The default initialization value shall be 1. The configurable range shall be 0 to 4

- logSyncInterval: The default initialization value shall be 0. The configurable range shall be -1 to +1
- logMinPdelayReqInterval: The default initialization value shall be 0. The configurable range shall be 0 to 5.
- announceReceiptTimeout: The default initialization value shall be 3. The configurable range shall be in the range 2 to 10.
- priority1: The default initialization value shall be 128.
- priority2: The default initialization value shall be 128.
- slaveOnly: If this parameter is configurable the default value shall be FALSE.
- primaryDomain: The default initialization value shall be 0.
- τ, see 7.6.3.2: The default initialization value shall be 1.0 seconds.

For each defined range, manufacturers are free to allow wider ranges.

J.4.3 PTP Options

All options of Clauses 16 and 17 are permitted. By default these options shall be inactive unless specifically activated by a management procedure.

Node management shall implement the management message mechanism of this standard.

The best master clock algorithm shall be the algorithm specified by 9.3.2.

The peer delay mechanism shall be the default path delay measurement mechanism. The delay request-response mechanism may also be implemented.

NOTE—Only a single mechanism is allowed per link. Boundary clocks should be used between links that use different path delay mechanisms.

J.4.4 Clock physical requirements

J.4.4.1 Frequency accuracy

Every grandmaster clock shall maintain a frequency deviating no more than 0.01% from the SI second.

J.4.4.2 Frequency adjustment range

Any clock in the SLAVE state shall be able to correct its frequency to match any master clock meeting the requirements of J.4.4.1

NOTE—The frequency adjustment range should be at least ±0.025%.

Annex K
(informative)
Security Protocol (experimental)

K.1 General

This Annex defines an experimental security extension to PTP, see 14.2. Since this Annex is not normative, the requirements are not expressed by the term “shall”. Instead the words “is (are) required to” are used. Implementers of this extension are advised to interpret the words “is (are) required to” in this Annex as “shall” in order to correctly implement the extension in the event that this annex becomes normative in future editions of the standard.

The PTP security extension and protocol provide group source authentication, message integrity, and replay attack protection for PTP messages.

The PTP security protocol is composed of two basic mechanisms:

a) An integrity protection mechanism, which uses message authentication code to verify that a received message was transmitted by an authenticated source, was not modified in transit, and it is fresh (i.e. not a message replay). Replay protection is implemented using counters.

b) A challenge-response mechanism, which is used to affirm the authenticity of new sources and to maintain the freshness of the trust relations.

K.2 Protocol overview

The PTP security protocol uses symmetric message authentication code functions. It provides group source authentication, message integrity, and replay protection. The security protocol does not provide non-repudiation. The protocol support HMAC-SHA1-96, HMAC-SHA256-128, and allows addition of other message authentication codes in the future. The implementation of these algorithms is required to be in accordance with the references [M9], [M10], [M20], and [M21].

The participants in the protocol share secret symmetric keys. The keys can be shared by the whole domain, or by subsets of the domain. The key distribution can be done either by manual configuration, or by an automatic key management protocol; the PTP security extension support both. Key distribution is out of the scope of this specification.

The participants in the PTP security protocol communicate through Security Associations (SAs). An SA contains a source (source port, protocol address), a destination (destination port, protocol address), a key, a random lifetimeID, and a replay counter. The SA is uni-directional, and it protects traffic going from the source to the destination. Each node maintains a table of incoming SAs, which it uses for verification of incoming traffic, and a table of outgoing SAs, which it uses for protection of outgoing traffic. An SA can be shared by a single sender, and multiple receivers. The sender holds a single copy of the SA, and each one of the receivers has its own copy of it. The receivers’ copies might contain at the same time different values of the SAs replay protection counter, but all of them are smaller than the replay counter stored at the same time in the sender’s copy of the SA.

The SA is created by the sender, and it is communicated to the receivers. The sender can choose to create a single SA for each source and all destinations; i.e. create an SA with the source being one of its interface unicast addresses, and the destination being “all”; or it can choose to create an SA per a source and destination, i.e. create multiple SAs with the same interface source address and different multicast and unicast destinations. This is an implementation-specific decision. If a single SA for all destinations is used, then the replay protection counter wraps much faster, increasing the rate of the SA's update. If multiple SAs are used for multiple destinations, then the outgoing SA table is larger. The receivers of messages on the SA don't care which SA implementation method is used by the sender.

The integrity check value (ICV) is the result of applying the message authentication code function specified by the algorithm ID with the appropriate key identified by key ID to the packet, including the security AUTHENTICATION TLV. Only nodes who know the shared secret key can modify the message and its ICV respectively, hence any message tamper attempt by an attacker who doesn't have the secret key will be detected when the receiver fails to verify the ICV.

The replay protection mechanism relies on a random lifetime ID, and replay protection counter, which are part of the SA. The replay counter is incremented by two whenever a packet sent through this SA. The receiver verifies that lifetime ID in the packet, matches the lifetime ID of the sender's SA in its incoming SA table, and that the counter in the packet is larger than the value stored in the SA.

K.3 General requirements

The PTP security protocol uses a flag bit in the PTP message header to indicate that the message carries the security AUTHENTICATION TLV. The PTP header Flags field, see 13.3.2.6, is extended by one bit as defined in Table 140.

Table 140: SECURE flag (PTP header Flags field)

Octet	Bit	Message type	Name	Description
0	7	All	SECURE	TRUE if the message is suffixed by the security AUTHENTICATION TLV and FALSE otherwise.

Each secured transmitted PTP message is required to set the SECURE flag to TRUE and include the security AUTHENTICATION TLV extension defined in subclause K.15. This extension is required to be the final TLV extension appended to the PTP message.

NOTE— To facilitate hardware implementation of the security protocol, the ICV field should be the last field in the messages.

In the following, the action specified in certain cases is to "silently discard" a received message. This means that the message is discarded without further processing and that no internal or external resources are allocated as a result of processing this message. However the event may be recorded in a statistics counter or in any other similar action that does not allocate new system resources. This definition limits the ability of a denial of service attack to exhaust system or network resources.

Some PTP systems require support of a mixture of secure and non-secure clocks. One example is a PTP system composed of two grandmasters clocks (one used as backup) supporting the security protocols, a group of ordinary clocks that supports the security protocol, and a group of slave-only clocks that do not support the security protocol. The administrator of this PTP system may want to allow the non-secure slave-only clocks to synchronize to the secure grandmaster. As defined in this annex all communication between clocks must be secure. Therefore, although the slave clocks can parse Announce and Sync messages sent by the grandmaster, the grandmaster clock discards the unsecure Delay_Req messages received from the slave-only clocks. It is expected that extensions of this annex either within a profile or in the next version of this standard will introduce other 'security policies' to enable mixtures of secure and non-secure clocks. Examples of such extensions include allowing a clock to process and reply to non-secure Delay_Req and Pdelay_Req messages or to process and reply to non-secure management Get requests, up to a limited rate of messages.

K.4 The challenge-response exchange

The challenge-response exchange is a three way mutual authentication protocol, which two nodes use to affirm their authenticity and freshness. A node trusts integrity and replay information it receives from another node only after it successfully executes a challenge-response with that node.

Each clock maintains a list of incoming security associations. Incoming SAs can be either static, i.e. configured in advance, or created on the fly once the clock receives a PTP message from a port that does not match the sourcePortID and protocol address of any member of the current set of incoming SAs.

The incoming SA's trust state is set to UNTRUSTED once the clock initializes and once new SAs are created. The trust state is set to TRUSTED only after it successfully executes the challenge-response test per K.10. Each incoming SA maintains a timer that measures the period that has passed since the last time it received an authenticated message. When this time expires the incoming SA trust state is set to UNTRUSTED. If the SA is not static, the SA is discarded once the timeout expires.

A clock initiates a challenge-response exchange when it receives a message from an untrusted source. The incoming message must have the SECURE flag set to ON, must include the security AUTHENTICATION TLV, and must pass integrity check value (ICV) test per K.6. If the message received does not match an incoming SA, a new dynamic incoming SA is created as long as resources permit. The incoming SA maintains challenge state and a challenge timer. The challenge state is set to CHALLENGING when a challenge request is sent and a reply is pending. No new challenge request is sent through the SA as long as the challenge state is CHALLENGING. If a challenge reply is not received, the challenge timer expires and the challenge state is set to IDLE.

The challenge-response exchange uses the AUTHENTICATION_CHALLENGE TLV. The AUTHENTICATION_CHALLENGE TLV includes a field that defines the challenge type; Request, Response-Request, and Response. The AUTHENTICATION_CHALLENGE TLV must be appended as the first TLV of the security signaling message. Signaling messages used for the challenge exchange are required to be used only for security protocol operations. A challenge-response exchange is also initiated to update security association parameters as defined in the next subclause.

K.5 The security association update exchange

The replay protection requires that the replay counter does not roll over. Outgoing SAs maintain both current and next randomly generated non-zero lifetime IDs. When the replay counter of the outgoing SA rolls over, the SA switches from the current lifetime ID to the next lifetime ID and generates a new next lifetime ID.

Incoming SAs maintain current and next lifetime IDs as well. The incoming SA switches to the next lifetime ID once it receives first authenticated messages with that lifetime ID in the Security AUTHENTICATION TLV. The incoming SA copies the replay-counter value from the AUTHENTICATION TLV and initiates a challenge-response exchange to determine the new next lifetime ID.

The next lifetime ID is delivered in challenge-response and challenge-response-request messages using the SECURITY_ASSOCIATION_UPDATE TLV. Multiple TLVs can be used if the responder holds a different outgoing SA for each outgoing address (unicast, multicast and p-multicast) communicating with the requestor.

The security association update exchange includes provisions to exchange next key IDs to update security association key once a given key is about to expire. The key IDs should be replaced often enough, to protect from replay attack due to lifetime ID re-use. The shared secret keys must be updated accordingly by shared key distribution mechanism to maintain enough valid keys.

Incoming SAs maintain current and next key IDs. The incoming SA switches to the next key ID once it receives first authenticated message with that key ID in the security AUTHENTICATION TLV. The incoming SA initiates a challenge-response exchange to determine the new next key ID. The next key ID is delivered in challenge-response and challenge-response-request messages using the SECURITY_ASSOCIATION_UPDATE TLV.

K.6 The integrity check value (ICV) test

The ICV is the result of applying the message authentication code function specified by the algorithm ID with the appropriate key identified by key ID. See also K.15.7 for algorithm-specific processing rules.

The ICV field of the security AUTHENTICATION TLVs is computed as follows for each PTP message sent through an outgoing security association:

a) The secret key specified by the key ID of the outgoing SA is used as the key value required by the hash algorithm.

b) The hash algorithm specified by the algorithm ID associated with the key ID is used to compute the ICV value. The algorithm ID is retrieved from the key list data set indexed by the key ID.

c) Using the selected hash algorithm and secret key the ICV value is computed over all PTP message fields beginning with the first octet of the common header and ending with and including the last octet of the security AUTHENTICATION TLV. Prior to this computation the value of all bits of the ICV field must be set to 0.

The ICV of incoming messages is calculated and compared with the ICV carried in the security AUTHENTICATION TLV. The check is performed as follows:

a) If the key specified by keyID in the AUTHENTICATION TLV is not valid or is unknown the ICV check fails.

b) If the algorithmID in the AUTHENTICATION TLV is not equal to the algorithm ID associated with the keyID in the key list data set the ICV check fails.

c) Using the message authentication code function selected by the algorithm and secret key the ICV value is computed over all PTP message fields beginning with the first octet of the common header and ending with and including the last octet of the security extension TLV. Prior to this computation the value of all bits of the ICV field of the security AUTHENTICATION TLV must be set to 0. The ICV test fails if the computed ICV does not match the ICV carried in the AUTHENTICATION TLV of the incoming message.

Otherwise the ICV test passes.

K.7 The security association lookup

Received PTP messages are matched against the incoming security association to determine if they are received from a trusted source. A received message matches a security association if the sourcePortId of the PTP header and the source protocol address match source port and source address of the incoming security association, and the destination port (if specified) and destination address of the message match the incoming security association destination port and protocol address. The security association lookup indicates whether a matching security association exists or not, and returns the trust state of the security association.

K.8 KeyID check

This test verifies that the keyID of an incoming message matches the incoming security association value. The keyID of a PTP message received from a trusted source (which also passed the ICV test) is compared against the corresponding value of the SA.

The incoming keyID is compared with the SA key_id. If the two are identical the test passes. If the keyID matches the next_key_id maintained by the SA's next_key_id is copied to the SA's key_id, the next_key_id is set to zero and a security update exchange is initiated. Messages that do not match either key_id or next_key_id are silently discarded.

As long as the incoming SA next_key_id is zero, and the challenge state is not CHALLENGING, the incoming SA association initiates a security update exchange each time a new message is successfully received by the SA.

K.9 The replay protection mechanism

The replay protection mechanism relies on the fact that the probability of the sourcePortID, lifetime ID and replay counter triplet, appearing twice is extremely low (practically zero). The probability depends on the shared keys lifetime, and it is smaller, the higher the key exchange frequency is.

The lifeTimeID and replay counter are set by the outgoing security association. The replay counter is incremented by 2 each message sent via the outgoing SA. When the replay counter rolls over the lifeTimeID is replaced by another random lifeTimeID. The replay counter and lifeTimeID are sent in the security AUTHENTICATION TLV.

The replay protection test is performed on packets received from a trusted incoming SA. The incoming lifeTimeID is compared with the SA lifetime_id. If the two are identical the incoming replay counter is compared with the SA replay counter. If the incoming replay counter is smaller or equal to the SA replay counter the message is silently discarded. If the lifeTimeID matches the next_lifetime_id maintained by the SA the replayCounter field of the incoming AUTHENTICATION TLV is copied to the SA's replay counter, the SA's next_lifetime_id is copied to the SA's lifetime_id, the next_lifetime_id is set to zero, and a security update exchange is initiated. Messages that do not match either lifetime_id or next_lifetime_id are silently discarded.

As long as the incoming SA next_lifetime_id is zero, and the challenge state is not CHALLENGING, the incoming SA association initiates a security update exchange each time a new message is successfully received by the SA.

K.10 The challenge-response check

The challenge-response exchange uses random nonce to verify authenticity and freshness of the source. The AUTHENTICATION_CHALLENGE TLV includes a request nonce and a reply nonce. The sender of the challenge request and challenge-response-request sets the request nonce to a random number. The receiver of challenge-response or challenge-response-request matches the request nonce it sent in the request to the reply. If the response nonce in the incoming challenge message does not match the nonce sent in the request nonce field of the challenge message the challenge-response check fails and the challenge message is silently discarded.

Challenge messages must pass the ICV test and for trusted sources must also pass the replay protection test. If either of these tests fails the challenge message is silently discarded.

K.11 Shared key distribution

The distribution of the shared keys to populate and update the security key list data set of each PTP node in a system is out of scope of this standard.

K.12 Generation of secret keys

The generation of secret keys is out of scope of this standard. An optional mechanism for generation of secret key for use with HMAC-SHA1-96 and HMAC-SHA256-128 and populating the shared secret key data set is described below. The general approach is described first.

Let H(x) be the n-bit hash of the message x. Let k be an n-bit key to be generated. Let p be a password, ss be a short-term salt, and sl be long-term salt. The variables ss, sl, p and x are all arbitrary character strings. While p must be kept secret, sl and ss are public values. Typically sl might be the name of the network or the organization that runs it (e.g., "Physics Lab"), and sl would be the dates of the crypto-period or a key name (e.g., "Jan-June 2007"). Let || represent concatenation. Then:

```
k= H(sl||ss||p)

for i=1, 2,..1000
        $k_i=H(k_{(i-1)})$
```

The final value of k, i.e., k_{1000}, can be truncated as desired to form the final key. The long-term salt, sl, could be initialized once in all stations, and never changed thereafter; it prevents an attack dictionary generated for another network with a different name from being reused on this network. The short-term salt should be changed whenever the key is changed; this prevents an attack dictionary computed for a previous key from being reused to attack a new key.

In particular, for PTP security protocol the following steps can be taken to generate secret keys:

For each keyID:

a) Select a password of varying length for each key.

b) Select a short-term salt.

c) Select a long-term salt.

d) Concatenate the long-term salt, short-term salt and password to a single message.

e) Hash the message using SHA-1.

f) Rehash the resulting message hash for 1000 times.

g) The resulting 20 octet output of SHA-1 is the 20 Octet shared secret key as defined in subclause K.13.2.3.

K.13 Security data set

K.13.1 General

The Security data set defined in this subclause is provided for illustration only and does not mandate specific implementation.

The security data set is composed of a list of incoming and outgoing security associations, a list of keys and a list of defaultDS data set parameters. Unless otherwise specified, the default value for all members is zero.

K.13.2 Key list

A key list must be maintained in a PTP node implementing the security protocol. Each key list entry is composed of kl_key_id, kl_algorithm_id, kl_security_key, kl_start_time, kl_expiration_time and kl_valid bit.

K.13.2.1 kl_key_id (UInteger16)

The kl_key_id is a unique identifier of the secret key. The value zero is required to not be used to indicate a valid key.

K.13.2.2 kl_algorithm_id (UInteger8)

The kl_algorithm_id indicates the algorithm to be used with the secret key.

K.13.2.3 kl_security_key (Octet[N])

The kl_secret_key field holds the security key. The value N depends on the algorithm used. For SHA-1 and SHA-256 N=20.

K.13.2.4 kl_start_time (Timestamp)

The kl_start_time indicates when the key will be active. This key is activated by setting the valid bit to TRUE. The key should not be used prior to kl_start_time.

K.13.2.5 kl_expiration_time (Timestamp)

The kl_expiration_time indicates when the key will expire. Once the key expires, the validity bit is set to FALSE and the key isno longer used. A zero kl_expiration_time value indicates that the key is permanent and therefore does not expire. One permanent key should be distributed to all clocks. The permanent key should be used when all other communication with this clock fails and is limited to establishing a security association and then change to a non-permanent key.

K.13.2.6 kl_valid (Boolean)

If kl_valid is set to FALSE the key is not valid and is required to not be used by the security protocol.

K.13.3 Security associations

The security associations are maintained in two lists, the incoming and outgoing security association lists. Each security association is composed of sa_src_port, sa_src_address, sa_dest_port, sa_dest_address, replay_counter, lifetime_id, key_id, next_lifetime_id, next_key_id, trust_state, trust_timer, trust_timeout, challenge_state, request_nonce, response_nonce, challenge_timer, challenge_timeout, response_required, challenge_required and sa_type as defined below.

K.13.3.1 sa_src_port (PortIdentity)

The sa_src_port is matched to the sourcePortIdentity field in the PTP header. For outgoing SAs it indicates one of the ports of the clock and for incoming SAs it equals the portIdentity of the clock that sent the message. For outgoing SAs the sa_src_port's portNumber member may be set to all-ones value to indicate the SA is used for all ports of the clock. For incoming SAs the portNumber member or clockIdentity member are required to not be set to all-ones values.

K.13.3.2 sa_src_address (Octet[N])

The sa_src_address is matched against the source protocol address of the PTP message. For outgoing Sas the sa_src_address may be set to all-ones values, indicating that the SA matches all addresses. For incoming SAs the sa_src_address is required to not equal all-ones values. The source protocol address of the received PTP message is matched against the sa_src_address parameter of incoming SAs. The source protocol address of PTP messages sent is matched against the sa_src_address parameter of outgoing SAs. For IPv4 encapsulations N=4, for IPv6 N=16, and for Ethernet N=6..

K.13.3.3 sa_dest_port (PortIdentity)

For outgoing SAs the sa_dest_port equals the portIdentity the message is sent to, and for incoming SAs it equals the identity of one of the clock's ports. For outgoing SAs sa_dest_port may be set to all-ones to indicate 'all clocks and all ports'. The sa_dest_port portNumber may be set to all-ones to indicate the SA is used for 'all ports of a particular clock'. For example, sync and announce messages sent to multicast address are sent to all clocks and all ports.

K.13.3.4 sa_dest_address (Octet[N])

The sa_dest_address equals the destination protocol address of the PTP message. For outgoing SAs it is the addresses of the clock the message is sent to and for incoming SAs it equals one of the clock's Unicast addresses or one of PTP multicast addresses. For outgoing SAs the dest_address can be set to all-ones values indicating that the SA matches all addresses. For IPv4 encapsulations N=4, for IPv6 N=16 and for Ethernet N=6.

K.13.3.5 replay_counter (UInteger32)

For outgoing SA the replay_counter is incremented by 2 each time PTP message is sent through the SA. For incoming SA the replay_counter stores the replayCounter of the last successfully authenticated incoming AUTHENTICATION TLV and is used in the replay protection mechanism.

K.13.3.6 lifetime_id (UInteger16)

For outgoing SA the lifetime_id is a random number used to mark all packets sent through the SA. For incoming SAs the lifetime_id is compared to the lifeTimeID in the incoming AUTHENTICATION TLV. The value zero indicates that lifetime_id has not yet been set.

K.13.3.7 key_id (UInteger16)

The key_id indicates which key is used for computation of the ICV. For incoming SAs it is determined during the challenge-response exchange.

K.13.3.8 next_lifetime_id (UInteger16)

When the replay_counter of an outgoing SA rolls over the value of next_lifetime_id is copied to lifetime_id and a new random non zero value is generated. The next_lifetime_id of an incoming SA is determined during the challenge-response exchange. The value zero indicates that next_lifetime_id has not yet been set.

K.13.3.9 next_key_id (UInteger16)

The next_key_id indicates the key that is going to be used after this key expires. For incoming SAs it is determined during the challenge-response exchange. The value zero indicates that next_key_id has not been set yet.

K.13.3.10 trust_state (Enumeration)

The trust state of incoming SA is set to TRUSTED following successful challenge-response exchange and is set to UNTRUSTED due to timeout or initialization event. An outgoing SA's trust_state is not used.

K.13.3.11 trust_timer (UInteger16)

The trust_timer of incoming SAs is set to the trust_timeout value each time a successfully authenticated PTP message is received by the SA. It is decremented by one every period by the security_event task. When the trust_timer reaches zero the SA trust state is set to UNTRUSTED. An outgoing SA's trust_timer is fixed at zero.

K.13.3.12 trust_timeout (UInteger16)

If within trust_timeout periods of security_event no successfully authenticated message is received through the SA the incoming SA is timed-out. An outgoing SA's trust_timeout is fixed at zero.

K.13.3.13 challenge_state (Enumeration)

Incoming SA's challenge_state indicates whether the SA is waiting for a reply for a challenge request. It can be set to CHALLENGING or IDLE values. An outgoing SA's challenge_state is fixed.

K.13.3.14 challenge_timer (UInteger16)

The challenge_timer of incoming SAs is set to challenge_timeout once a challenge-request or challenge-response-request is sent by the SA. The challenge_timer is decremented each period by the security_event task. When the challenge_timer reaches zero the SA challenge state is set to IDLE. An outgoing SA's challenge_timer is fixed at zero.

K.13.3.15 challenge_timeout (UInteger16)

If within challenge_timeout periods of the security_event a challenge-response or challenge-response-request is not received, the challenge state is changed to IDLE. An outgoing SA's trust_timeout is fixed at zero.

K.13.3.16 request_nonce (UInteger32)

The request_nonce values is the requestNonce field sent in challenge requests and compared to the nonce in responses. It is a randomly generated non-zero number used in the challenge-response exchange.

K.13.3.17 response_nonce (UInteger32)

The response_nonce is the requestNonce received in challenge requests and maintained in the SA to facilitate generation of the challenge-response.

K.13.3.18 challenge_required (Boolean)

The challenge_required value is set to TRUE when a challenge request needs to be sent to update incoming security association.

K.13.3.19 response_required (Boolean)

The response_required value is set to TRUE when a challenge-response needs to be sent.

K.13.3.20 sa_type (Enumeration)

The sa_type can be set to STATIC or DYNAMIC values. Static SAs are pre-configured associations, are maintained during clock initialization and kept in non volatile memory. Dynamic SAs are created for communication with clocks for which static SA were not set in advance.

The following attributes are maintained in non-volatile memory and in initialization events:

- Outgoing SA: sa_src_port, sa_src_address, sa_dest_port, sa_dest_address and key_id.
- Incoming SA: sa_src_port, sa_src_address, sa_dest_port, sa_dest_address, trust_timeout, challenge_timeout.

K.13.4 DefaultDefaultDS data set security variables

The defaultDS data set includes the following two security parameters, security_enabled and number_security_associations.

K.13.4.1 security_enabled (Boolean)

If the security_enabled value set to TRUE all PTP communication is required to use the security protocol extension.

K.13.4.2 number_security_associations (UInteger16)

The number_security_associations is the maximal number of security associations supported by the clock including all incoming and outgoing SAs as well as both static and dynamic SAs.

K.14 Protocol operation

K.14.1 General

This subclause illustrates the operation of the security protocol. The processing of PTP messages received and transmitted, and challenge processing are illustrated. A security event periodic process for handling timeouts and sending challenge messages is also described. The last sub clause details transparent clock processing rules.

K.14.2 Receive message processing

Figure 51 illustrates the processing of incoming PTP messages when security_enabled is set to TRUE. This subclause does not mandate specific implementation as long as the results of the tests are consistent. The processing steps are:

a) Silently discard incoming messages received without the SECURE bit set to TRUE.

b) Silently discard incoming messages received without the appended security AUTHENTICATION TLV.

c) Silently discard incoming messages that do not pass the integrity check verification test per subclause K.6.

d) Lookup the matching incoming SA per subclause K.7.

e) If no matching SA is found and there are available SAs generate a new SA. If there are no available SAs silently discard the message.

f) If the trust state is UNTRUSTED:

   1) If the incoming message is a challenge message, continue processing as defined in subclause K.14.3.

   2) Else set the challenge_required bit in the SA and drop message.

g) If the trust state is TRUSTED:

   1) If the keyID test per subclause K.8 fails, silently discard message.

   2) If the replay protection test per subclause K.9 fails, silently discard message.

   3) If the incoming message is a challenge message, continue processing as defined in subclause K.14.3.

   4) If the incoming SA next_lifetime_id or next_key_id are zero, set the challenge_required bit in the SA.

   5) Trigger the trust timer.

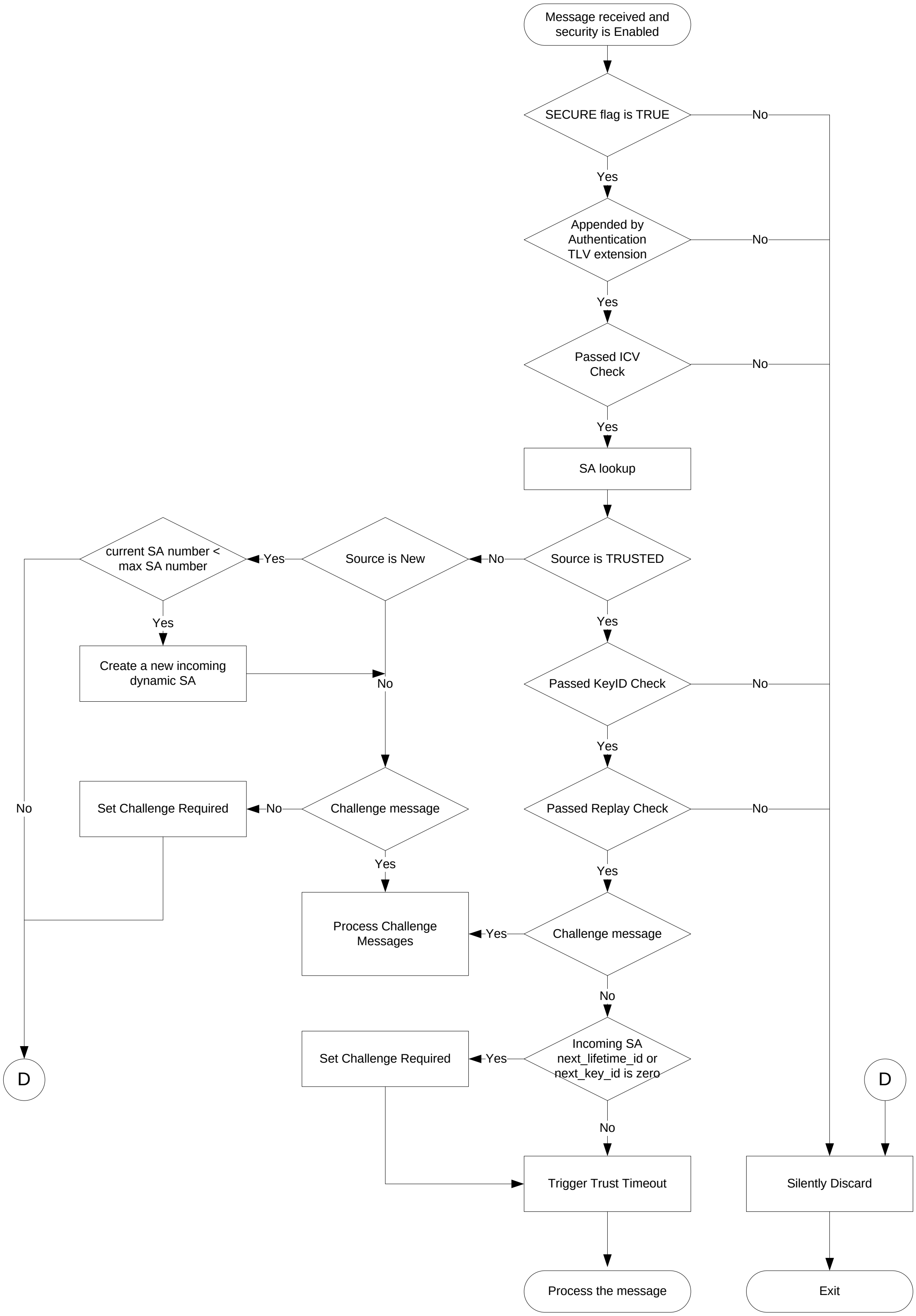


Figure 51: PTP secure message processing

K.14.3 Challenge processing

Figure 52 illustrates the processing of incoming challenge messages. The challenge processing is a continuation of the general incoming message processing as described in subclause K.14.2.

Incoming challenge message processing includes the following steps:

a) If the message is a challenge Request, set the response required bit in the SA, store the requestNonce in the SA’s response_nonce field, and exit.

b) If the message is a challenge-response or challenge-response Request and the SA challenge status is not CHALLENGING (i.e. the SA does not expect a response), or if the challenge-response check per subclause K.10 fails, silently discard message.

c) Else (successful challenge-response):

   1) Set the trust state to TRUSTED.

   2) Set the challenge state to IDLE.

   3) Copy the lifeTimeID from the AUTHENTICATION TLV to the SA’s lifetime_id.

   4) Copy the replayCounter from the AUTHENTICATION TLV to the SA’s replay_counter.

   5) Copy the nextlifeTimeID from the SECURITY_ASSOCIATION_UPDATE TLV to the relevant SAs’ next_lifetime_id selected by addressType.

   6) Copy the nextKeyID from the SECURITY_ASSOCIATION_UPDATE TLV to the relevant SAs’ next_key_id selected by the addressType.

   7) Trigger trust timeout.

d) If the challenge message is response-request, set the response required bit in the SA and store the requestNonce in the SA’s response_nonce field.

e) Discard the message and exit.

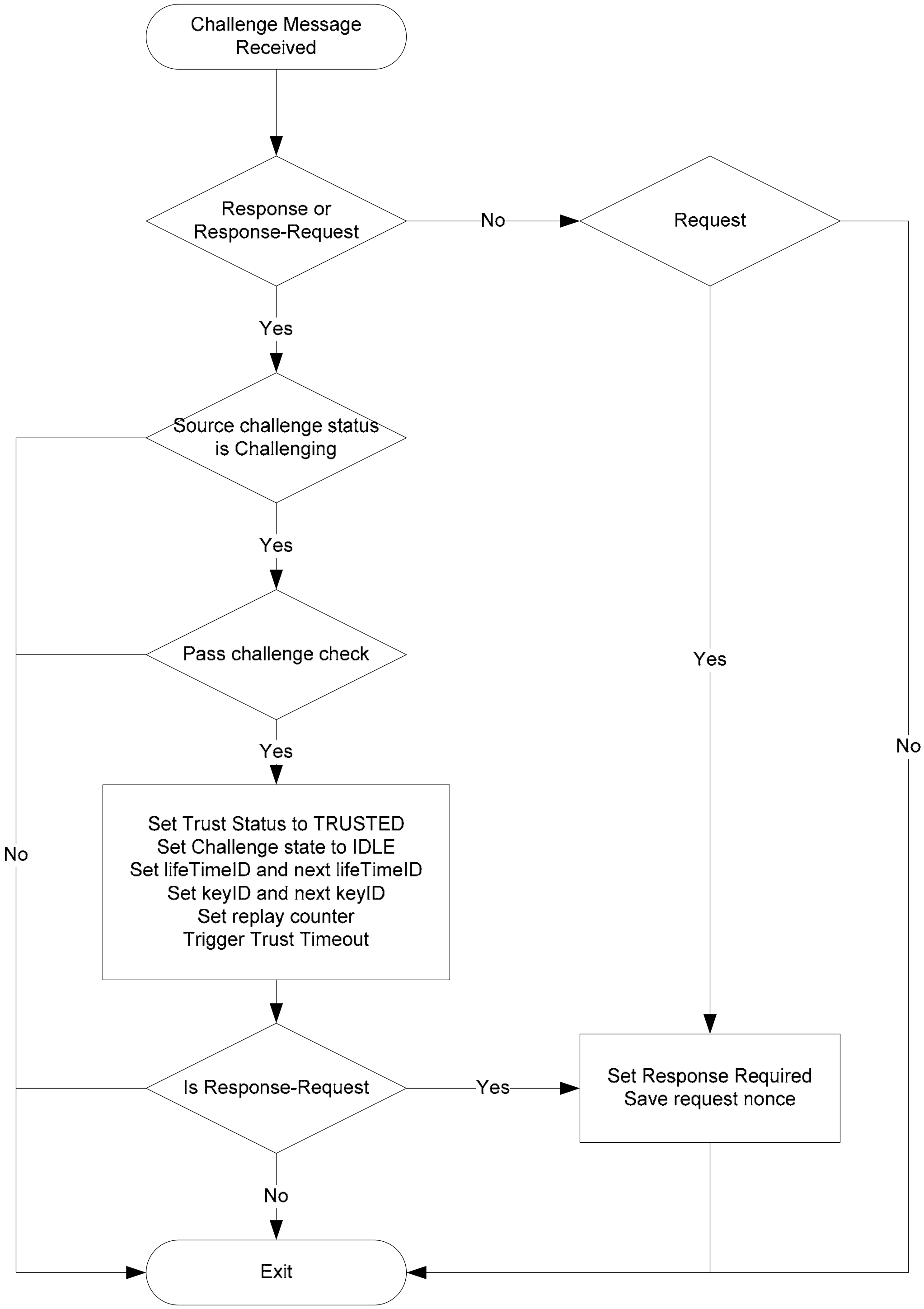

Challenge Message Received
Response or Response-Request
No
Request
Yes
Source challenge status is Challenging
Yes
Pass challenge check
Yes
Yes
No
Set Trust Status to TRUSTED
Set Challenge state to IDLE
Set lifeTimeID and next lifeTimeID
Set keyID and next keyID
Set replay counter
Trigger Trust Timeout
No
Is Response-Request
Yes
Set Response Required
Save request nonce
No
Exit

Figure 52: Challenge processing

K.14.4 Secure transmit processing

Figure 53 illustrates the additional processing of outgoing PTP messages. All messages are appended with the AUTHENTICATION TLV. The ICV calculation is required to follow subclause K.6. The steps include:

a) Set the SECURE flag to TRUE

b) Append the AUTHENTICATION TLV

   1) Copy the replay_counter from the outgoing SA to the TLV's replayCounter .

   2) Copy the life_time_id from the outgoing SA to the TLV's lifeTimeID.

   3) Copy the key_id from the outgoing SA to the TLV's keyID.

   4) Copy the kl_algorithm_id that corresponds to the keyID from the key list data set to the TLV's algorithmID.

c) Calculate the ICV and place it in the ICV field of the AUTHENTICATION TLV.

d) Advance the SA replay counter by 2.

e) If the SA replay counters rolls over then:

   1) Copy the SA's next_lifetime_id to the lifetime_id field.

   2) Generate a new random non zero next_lifeTime_id. Ensure that the next_lifeTime_id value is different from lifeTime_id value, otherwise generate a new non zero random value until a non-equal value is generated.

   3) Set the replay_counter to zero.

f) Send the message

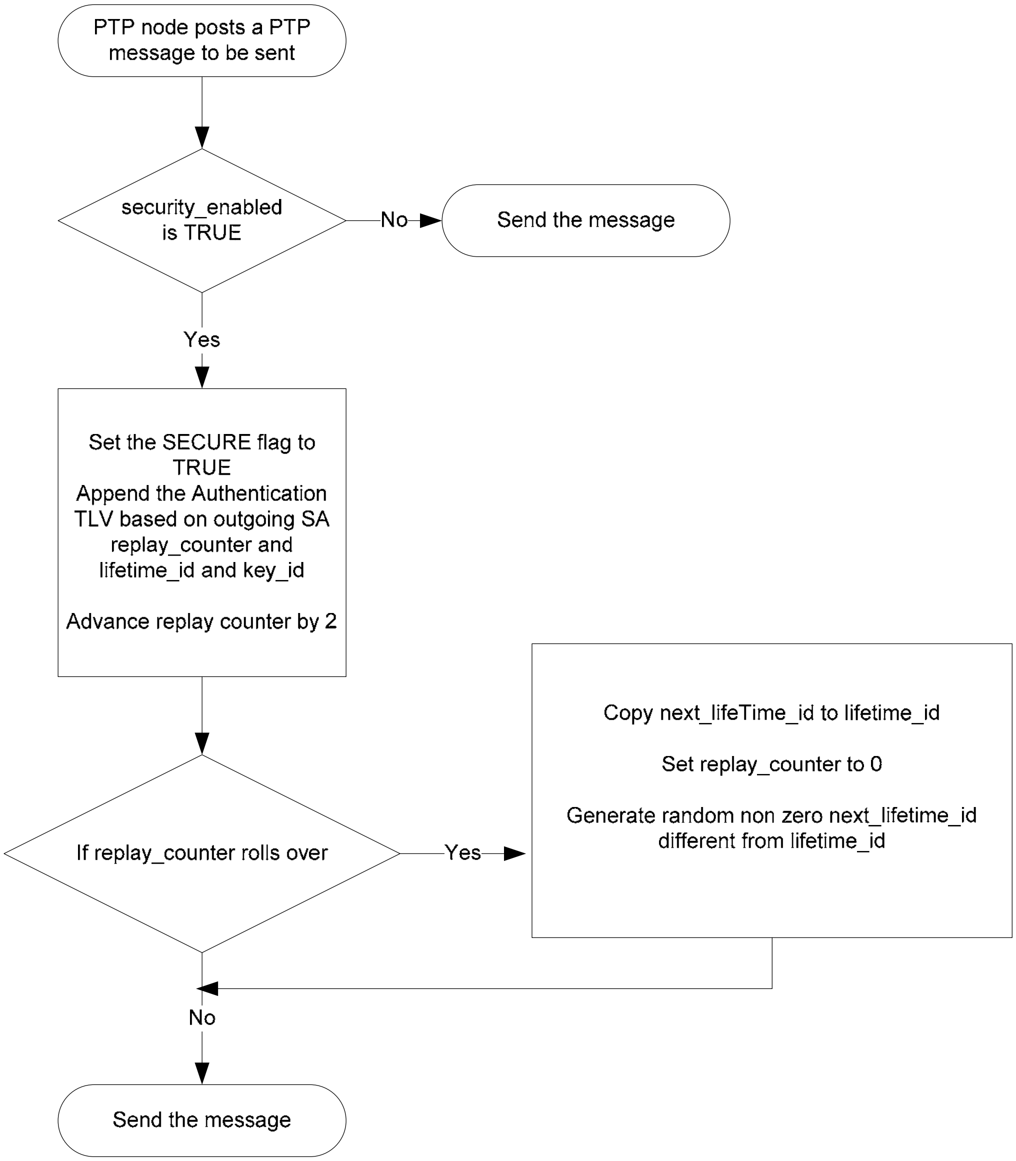

PTP node posts a PTP message to be sent
security_enabled is TRUE
No
Send the message
Yes
Set the SECURE flag to TRUE
Append the Authentication TLV based on outgoing SA replay_counter and lifetime_id and key_id
Advance replay counter by 2
If replay_counter rolls over
Yes
Copy next_lifeTime_id to lifetime_id
Set replay_counter to 0
Generate random non zero next_lifetime_id different from lifetime_id
No
Send the message

Figure 53: Secure transmit processing

K.14.5 Secure event processing

The secure event is a periodic event that handles timeouts and sends challenge messages. The secure event should be triggered frequently enough to handle timeouts. The secure event process should be triggered at least every time decision event process is triggered. The secure event process is required to be carried out atomically, see 3.1.2.

Figure 53 illustrates the secure event processing steps for each incoming SA. The steps of the secure event processing for incoming SA are:

a) If the challenge_state is CHALLENGING and the challenge_timeout has expired, change challenge_state to IDLE.

b) If the challenge_state is IDLE and both challenge_required and response_required bits are set, then:

   1) Build the SECURITY_ASSOCIATION_UPDATE TLV from all matching outgoing SAs.

   2) Generate the random request_nonce

   3) Build the AUTHENTICATION_CHALLENGE TLV. Copy the SA's request_nonce to the TLV's requestNonce field and the SA's response_nonce to the TLV's responseNonce.

   4) Send the challenge-response-request

   5) Set the challenge_state to CHALLENGING.

c) If the challenge_state is IDLE, the challenge_required bit is set, and the response_required bit is not set

   1) Generate the random request_nonce

   2) Build the AUTHENTICATION_CHALLENGE TLV. Copy the SA's request_nonce to the TLV's requestNonce field and the SA's response_nonce to the TLV's responseNonce.

   3) Send the challenge request.

   4) Set the challenge state to CHALLENGING.

d) If the challenge state is CHALLENGING, and the response_required bit is set, OR the challenge_state is IDLE and the response_required bit is set and the challenge_required bit is not set,

   1) Build the SECURITY_ASSOCIATION_UPDATE TLV from all matching outgoing SAs.

   2) Build the AUTHENTICATION_CHALLENGE TLV. Set the TLV's requestNonce field to zero and copy the SA's response_nonce to the TLV's responseNonce.

   3) Send the challenge-response

e) If the trust_timeout expired, set trust_state to UNTRUSTED.

f) Set the response_required and challenge_required bits to FALSE

g) Exit

The secure event process should also handle key expiration, activation and renewal. The secure event should not timeout keys unless it has access to accurate time to allow it to determine whether a key is about to expire or has already expired. The secure event process should activate keys once their start time is due. Whenever the secure event process does not have access to sufficiently accurate time all keys are activated. The security event process ensures that outgoing SAs exchange new keyIDs prior to key expiration through the security association update exchange defined in subclause 0. When the key is about to expire, or the key needs to be replaced for some other reason, the security event process copies the outgoing SA's next_key_id to the key_id field and sets the next_key_id to the key that would be used once the current one expires.

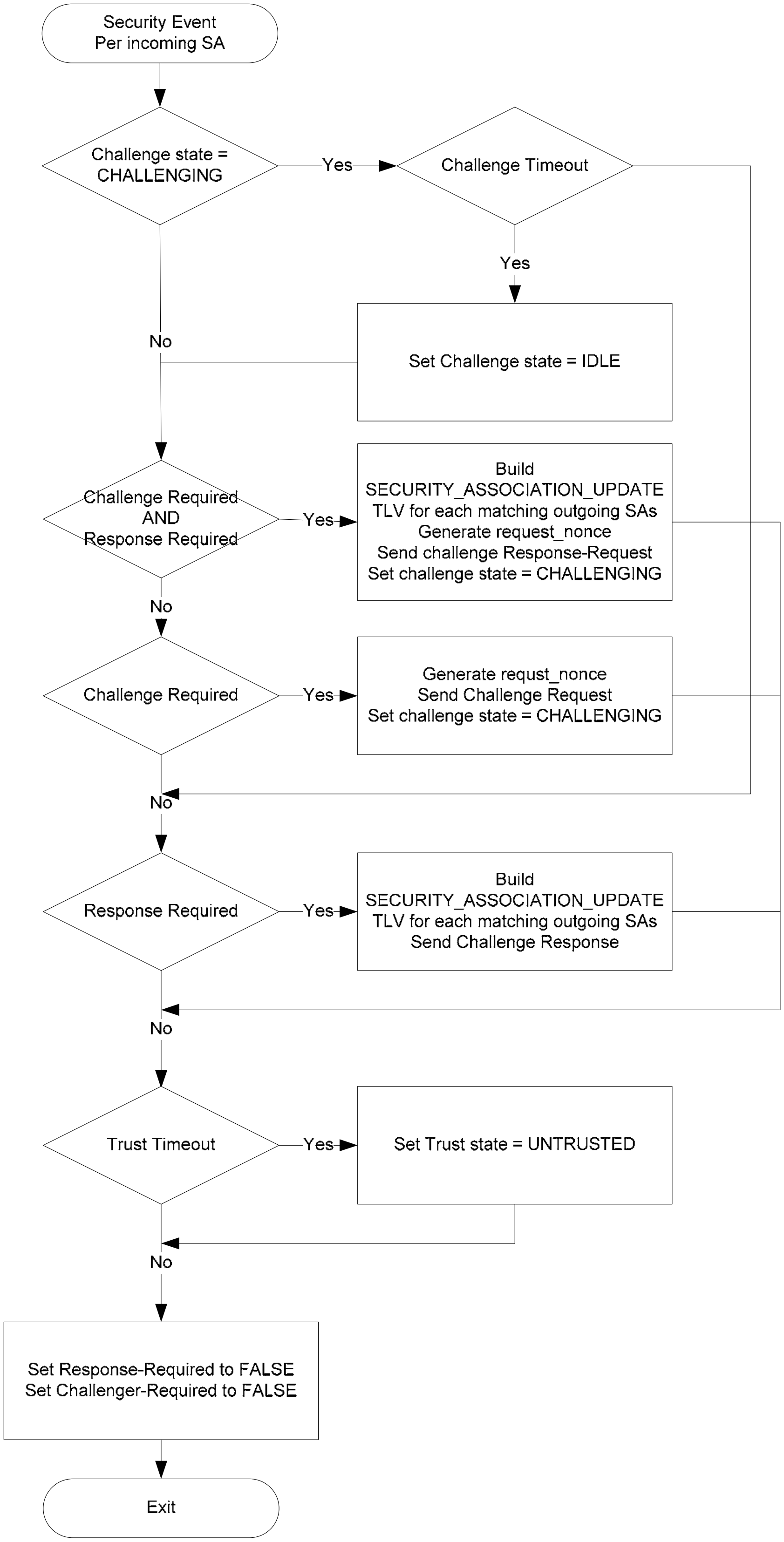

Security Event
Per incoming SA
Challenge state =
CHALLENGING
Yes
Challenge Timeout
Yes
Set Challenge state = IDLE
No
Challenge Required
AND
Response Required
Yes
Build
SECURITY_ASSOCIATION_UPDATE
TLV for each matching outgoing SAs
Generate request_nonce
Send challenge Response-Request
Set challenge state = CHALLENGING
No
Challenge Required
Yes
Generate requst_nonce
Send Challenge Request
Set challenge state = CHALLENGING
No
Response Required
Yes
Build
SECURITY_ASSOCIATION_UPDATE
TLV for each matching outgoing SAs
Send Challenge Response
No
Trust Timeout
Yes
Set Trust state = UNTRUSTED
No
Set Response-Required to FALSE
Set Challenger-Required to FALSE
Exit

Figure 54: Secure event processing

K.14.6 Secure transparent clock processing rules

This subclause details additional transparent clock processing rules required to support secure PTP communications. PTP transparent clocks are either security-unaware, security-aware or security capable clocks, as defined below.

A security unaware transparent clock ignores the value of the SECURE flag. A secure PTP message modified by a security-unaware transparent clock will fail the PTP receiver's integrity check value (ICV) test and would be silently discarded. A security-unaware transparent clock must not be deployed within a secure communication path.

A security aware transparent clock does not modify PTP messages with SECURE flag set to TRUE. PTP secure messages are forwarded according to the addressing rules of the network. Security-aware transparent clock can be deployed in environments that use both secure and non-secure PTP communications (e.g. over different domains).

A security capable transparent clock can either participate in the PTP security protocol if the defaultDS data set security_enabled is set to TRUE or behave as a security aware clock if security_enabled is set to FALSE. A security capable transparent clock is required to follow all secure protocol processing rules for all PTP messages addressed to it and for all messages it originates. A security capable transparent clock is required to follow the transparent clock processing rules defined in clause 10 and 11. In addition, if security_enabled is set to TRUE, the following additional processing rules are required to be followed:

a) If the SECURE flag is set to FALSE the regular processing rules of transparent clocks are required to be followed. Further processing rules below are applicable to PTP messages with SECURE flag set to TRUE.

b) The transparent clock is required to perform the ICV test as defined in subclause K.6 for all PTP event messages. If the ICV test fails the transparent clock is required to silently discard the PTP message.

c) A two-step transparent clock is required to perform ICV test for Delay_Resp and Follow_Up messages. If the ICV test fails the transparent clock is required to silently discard the PTP message.

d) A two-step clock generating a Follow_Up message per subclause 11.5.2.2 b) is required to copy the Sync message security AUTHENTICATION TLV to the Follow_Up message. The two-step clock is required to increment the replayCounter of the AUTHENTICATION TLV by one.

e) The transparent clock is required to recalculate the ICV value of the PTP message after it completes modifying the PTP message fields (correction field, twoStepFlag, etc.) according to the procedure defined in subclause K.6. The transparent clock is required to update the ICV field of the security AUTHENTICATION TLV with the calculated correct ICV.

In addition, if the defaultDS data set security_enabled attributes is set to TRUE, the transparent clock is required to not syntonize to a master clock unless it accomplishes a full trust relation with that clock.

Transparent clocks are not required to support security association data set for the purpose of secure residence time and path delay corrections. Security capable transparent clocks are required to support and maintain security associations only for the purpose of management, syntonization and peer-to-peer messaging.

K.15 Authentication TLV

K.15.1 General

The AUTHENTICATION TLV is required to be appended to all PTP messages with SECURE flag set. The field values of the TLV are required to be determined as defined in this subclause. Reserved fields are required to be set to zero on transmit and ignored on receipt.

Table 141: AUTHENTICATION TLV

Bits								Octets
7	6	5	4	3	2	1	0	
tlvType = AUTHENTICATION								2
lengthField								2
lifeTimeID								2
replayCounter								4
keyID								2
algorithmID								1
reserved								1
pad								M
ICV (Integrity Check Value)								N

K.15.2 tlvType

The tlvType value is AUTHENTICATION

K.15.3 lengthField

The length of the TLV depends on the ICV and Pad lengths. For all extensions defined in this standard the length is 26 decimal.

K.15.4 lifeTimeID (UInteger16)

The lifeTimeID is a fixed number determined by the SA. The lifeTimeID is checked as part of the replay protection mechanism. The lifeTimeID is required to not be set to the value 0.

K.15.5 replayCounter (UInteger32)

The replayCounter is incremented by two for each packet sent through the SA. The replayCounter is checked as part of the replay protection mechanism.

K.15.6 keyID (UInteger16)

The key identifier (keyID) field is used to select which of possibly many shared secret keys is used.

K.15.7 algorithmID (UInteger8)

The possible values for algorithmID are required to be taken from the enumeration of Table 142.

Table 142: algorithmID values

algorithmID	Value

NULL	0
HMAC-SHA1-96	1
HMAC-SHA256-96	2
Reserved	3-128
Implementation-specific	129-255

All PTP nodes supporting security extensions are required to support HMAC-SHA1-96 algorithm.

HMAC processing is defined in [M20] and in [M21]. It defines output truncation and padding procedures. Both SHA1 and SHA256 use block size of 512 bits. The message is padded before hash computation begins with zero valued bytes to ensure that the padded message is a multiple of 512 bits. The output of SHA1 is truncated from 160 bits to 96 bits and the output of SHA256 is truncated from 256 bits to 128 bits to generate the ICV value. Truncation selects the left most bits of the generated hash.

The NULL algorithm does not provide integrity protection and is defined for testing purposes only. The ICV field is of length zero for the NULL algorithm.

K.15.8 Pad (Octet[M])

The value of the pad field is required to be set to zero and ignored on receipt. The pad field length M is per algorithmID is listed in Table 143

NOTE— The pad field length was selected such that the AUTHENTICATION TLV length is fixed for all algorithmIDs defined in this version of the standard. Fixed length AUTHENTICATION TLV facilitates hardware implementation. However, future algorithmIDs may define ICV and pad lengths that are not summed to this fixed length.

K.15.9 ICV (Octet[N])

The method of computing the ICV value is specified in subclause K.6. The ICV length N is per algorithmID is listed in Table 143.

Table 143: ICV and pad length

algorithmID	ICV length (Bytes)	Pad length (Bytes)
NULL	0	16
HMAC-SHA1-96	12	4
HMAC-SHA256-128	16	0

K.16 Authentication challenge TLV

K.16.1 General

The AUTHENTICATION_CHALLENGE TLV is used for the for the authentication challenge-response exchange.

The extension is illustrated in Table 144. The AUTHENTICATION_CHALLENGE TLV is required to be sent in signaling messages. The AUTHENTICATION_CHALLENGE TLV is required to be appended as the first TLV in the message.

Table 144: AUTHENTICATION_CHALLENGE TLV

Bits								Octets
7	6	5	4	3	2	1	0	

tlvType = AUTHENTICATION_CHALLENGE	2
lengthField	2
challengeType	1
reserved	1
requestNonce	4
responseNonce	4

K.16.2 tlvType

The tlvType field is AUTHENTICATION_CHALLENGE

K.16.3 lengthField

The lengthField value is 14.

K.16.4 ChallengeType (UInteger8)

The possible values of challengeType are required to be taken from the enumeration of Table 145.

Table 145: challengeType values

challengeType	Value
Request	0
Response-Request	1
Response	2
Reserved	3-255

K.16.5 requestNonce (UInteger32)

The requestNonce is a random number generated by the sender of challenge requests or challenge-response-requests. The requestNonce should be set to 0 in challenge-response messages.

K.16.6 responseNonce (UInteger32)

The challenge responder copies the requestNonce from the challenge Request and Response-Request messages to the responseNonce in challenge-response-Request and Responses. The responseNonce is set to zero in challenge Request messages.

K.17 Security association update TLV

K.17.1 General

The SECURITY_ASSOCIATION_UPDATE TLV is used for delivery of the security association lifeTimeID value to be used once replay counter of the current security association rollover and the keyID value to be used once the key is replaced. The security association update TLV is required to be sent in challenge-response and challenge response-request signaling messages. The TLV can be used to deliver the updated security association relevant to all addresses (unicast, multicast and p-delay multicast) or deliver security association update information for a particular address if different outgoing security associations are maintained by the challenge responder. Several TLVs can be sent in the same message each providing the updated SA for a particular address. The extension is illustrated in Table 146. Reserved fields are required to be set to zero on transmit and ignored on receipt.

Table 146: SECURITY_ASSOCIATION_UPDATE TLV

Bits								Octets
7	6	5	4	3	2	1	0	
tlvType = SECURITY_ASSOCIATION_UPDATE								2
lengthField								2
addressType								1
reserved								1
nextKeyID								2
nextLifeTimeID								2

K.17.2 tlvType

The tlvType field is SECURITY_ASSOCIATION_UPDATE

K.17.3 lengthField

The lengthField value is 10.

K.17.4 addressType (UInteger8)

The possible values of addressType are required to be taken from the enumeration of Table 147.

Table 147: addressType values

addressType	Value
All	0
Multicast	1
P-Multicast	2
Unicast	3
Reserved	4-255

K.17.5 nextKeyID (UInteger16)

The nextKeyID indicates the security key the outgoing security association will use once the current key expires or replaced. The nextKeyID is used to update the incoming security association.

K.17.6 nextLifeTimeID (UInteger16)

The nextLifeTimeID indicates the lifeTimeID the outgoing security association will use once replay counter of the current outgoing security association rollovers. The nextKeyID is used to update the incoming security association.

Annex L
(informative)
Transport of cumulative frequency scale factor offset (experimental)

L.1 General

This Annex defines an experimental cumulative frequency scale factor offset extension to PTP, see 14.2. Since this Annex is not normative, the requirements are not expressed by the term “shall”. Instead the words “is (are) required to” are used. Implementers of this extension are advised to interpret the words “is (are) required to” in this Annex as “shall” in order to correctly implement the extension in the event that this Annex becomes normative in future editions of the standard.

In some compensation schemes, rather than directly adjusting the phase of a boundary clock, the frequency is adjusted in such a way as to reduce both the phase and frequency error. The boundary clock computes a frequency scale factor using successive time stamps it receives from its master; specifically, it uses the time the master sent each timestamp (the originTimestamp or preciseOriginTimestamp) and the time it received each timestamp. In using this information, the scale factor is computed relative to the master clock. However, if the boundary clock also knows the cumulative frequency scale factor relative to its grandmaster, the accumulated phase error at the boundary clock can be reduced. This subclause specifies an optional TLV that can be used to transport cumulative frequency scale factor information from a boundary clock to its slaves. Specifically, the TLV accumulates the cumulative difference between the frequency scale factor and 1. The difference between the frequency scale factor and 1 is referred to as the frequency scale factor offset; the accumulation of the frequency scale factor offset over multiple boundary clock hops is referred to as the cumulative frequency scale factor offset.

L.2 Description of a frequency compensation scheme that uses cumulative frequency scale factor

[M3] describes in detail a frequency compensation scheme that uses a frequency scale factor. A frequency compensation value is computed at each successive boundary clock node, at each sync interval, as

$$\begin{aligned} &FreqCompensationValue_{k,0} = 1 \\ &FreqCompensationValue_{k,n} = F_{k,n} \cdot FreqCompensationValue_{k,n-1} \end{aligned} \qquad \text{(L-1)}$$

where $F_{k,n}$ is the frequency scale factor computed at node k and sync interval n. The adjusted, i.e., compensated, frequency is synthesized by multiplying the frequency of the free-running local oscillator by the current $FreqCompensationValue$. The computation of $F_{k,n}$ is specified in L.3.

[M25] shows that the phase error accumulation over a chain of boundary clocks using this compensation scheme can be greatly reduced by (1) synchronizing the message exchanges between successive pairs of boundary clocks so that a slave exchanges messages with and synchronizes to its master immediately after the master synchronizes to its master, and (2) synthesizing the compensated frequency using the cumulative frequency scale factor rather than the frequency scale factor. The cumulative frequency scale factor, $F_{cum,k,n}$, is given by (note that the accumulation is over the chain of nodes, and not over time)

$$F_{cum,k,n} = F_{cum,k-1,n} \cdot F_{k,n} . \qquad \text{(L-2)}$$

The frequency scale factor for the grandmaster, $F_{0,n}$, is equal to 1 (because the grandmaster is not synchronized to another clock via the protocol). Then

$$F_{cum,k,n} = \prod_{i=1}^{k} F_{i,n} \; . \qquad \text{(L-3)}$$

The frequency scale factor offset is defined as the difference between the frequency scale factor and 1 as shown in Equation L-4.

$$\begin{aligned} \delta_{k,n} &= F_{k,n} - 1 \\ F_{k,n} &= 1 + \delta_{k,n} \end{aligned} \; . \qquad \text{(L-4)}$$

Inserting Equation L-4 into Equation L-3 and noting that $\delta_{k,n} << 1$ (i.e., the frequency scale factor is very close to 1; this may be deduced from Equation L-6 or Equation L-8 in L.3), the cumulative frequency scale factor may be written

$$F_{cum,k,n} \cong 1 + \sum_{i=1}^{k} \delta_{k,n} \; . \qquad \text{(L-5)}$$

Equation L-5 shows that the cumulative frequency scale factor may be obtained by a cumulative sum of the frequency scale factor offsets (as opposed to a cumulative product of the frequency scale factors). The cumulative sum is computationally less costly, and is transported and accumulated in the CUM_FREQ_SCALE_FACTOR_OFFSET TLV specified in L.4.

L.3 General specification of cumulative frequency scale factor offset

If a boundary clock and its master implement this TLV, the boundary clock is required to compute its frequency scale factor offset on receipt of the TLV appended to either a Sync or Follow_Up message.

If the TLV is appended to the Sync message, the frequency scale factor offset is computed on receipt of each Sync message as (the node index *k* is omitted for simplicity, as all the variables of this subclause are referenced to the same node)

$$F_n = \frac{(T_{1,n} - T_{1,n-1}) + (T_{1,n} + d_n - T_{2,n})}{T_{2,n} - T_{2,n-1}} \qquad \text{(L-6)}$$

$$\delta_n = F_n - 1 \qquad \text{(L-7)}$$

where

δ_n = frequency scale factor offset on receipt of the n^{th} Sync message
F_n = frequency scale factor on receipt of the n^{th} Sync message
$T_{1,n}$ = originTimestamp contained in the n^{th} Sync message
$T_{2,n}$ = time stamp for receipt of the n^{th} Sync message
d_n = sum of :

- the current (i.e., at the time of receipt of the n^{th} Sync message) measured propagation time on the path on which the n^{the} Sync message is received and
- the correction field of the n^{th} Sync message

If the TLV is appended to the Follow_Up message, the frequency scale factor offset is computed on receipt of each Follow_Up message as

$$F_n = \frac{(T_{1,n} - T_{1,n-1}) + (T_{1,n} + d_n - T_{2,n})}{T_{2,n} - T_{2,n-1}} \qquad \text{(L-8)}$$

$$\delta_n = F_n - 1 \tag{L-9}$$

where

δ_n = frequency scale factor offset on receipt of the n^{th} Follow_Up message
F_n = frequency scale factor on receipt of the n^{th} Follow_Up message
$T_{1,n}$ = preciseOriginTimestamp contained in the n^{th} Follow_Up message
$T_{2,n}$ = time stamp for receipt of the n^{th} Sync message
d_n = sum of

- the current (i.e., at the time of receipt of the n^{th} Sync message) measured propagation time on the path on which the n^{the} Sync message and corresponding Follow_Up are received,
- the correction field of the n^{th} Sync message, and
- the correction field of the Follow_Up message corresponding to the n^{th} Sync message.

The boundary clock computes cumulative frequency scale factor offset relative to the grandmaster by adding δ_n computed as above to the value of cumulative_frequency_scale_factor_offset in the CUM_FREQ_SCALE_FACTOR_OFFSET TLV, see L.4, received most recently from the master.

If a boundary clock implements this TLV and its master does not implement this TLV, the boundary clock is required to set the cumulative frequency scale factor offset it computes equal to δ_n computed as above.

NOTE— If a boundary clock does not implement this TLV, it can still use a compensation scheme that uses the non-cumulative frequency scale factor, F_n. In this case, the boundary clock ignores any CUM_FREQ_SCALE_FACTOR_OFFSET TLV it receives from its master, and does not send CUM_FREQ_SCALE_FACTOR_OFFSET TLVs to its slaves.

If a boundary clock implements the CUM_FREQ_SCALE_FACTOR_OFFSET TLV, the TLV should be sent in every Follow_Up message. The CUM_FREQ_SCALE_FACTOR_OFFSET TLV may be sent in Sync messages only if all clocks within the communication path support the extension and pad the Delay_Req messages to the same length to avoid asymmetry effects.

L.4 CUM_FREQ_SCALE_FACTOR_OFFSET TLV specification

The CUM_FREQ_SCALE_FACTOR_OFFSET TLV format is required to be as specified in Table 72.

Table 72: CUM_FREQ_SCALE_FACTOR_OFFSET TLV format

			Bits					Octets	Offset
7	6	5	4	3	2	1	0		
tlvType								2	0
lengthField								2	2
cumulative_frequency_scale_factor_offset								4	4

L.4.1 tlvType

The value of tlvType is required to be CUM_FREQ_SCALE_FACTOR_OFFSET.

L.4.2 cumulative_frequency_scale_factor_offset

The value of cumulative_frequency_scale_factor_offset is the most recently computed cumulative frequency scale factor offset relative to the grandmaster of the boundary clock that sends the Sync message that contains the TLV, multiplied by 2^{41}, and expressed as a 32 bit signed integer in two's complement form. After the multiplication by 2^{41}, any remaining fractional part of the cumulative frequency offset is truncated, i.e., a positive cumulative frequency scale factor offset is replaced by its floor and a negative cumulative frequency scale factor offset is replaced by its ceiling.

Annex M (informative) Bibliography

[M1]: Allan, David W., Ashby, Neil, and Hodge, Cliff, “Fine-tuning Time in the Space Age,” *IEEE Spectrum*, March 1998.

[M2] Allan, David W., Ashby,Neil, Hodge, Clifford C. “The Science of Timekeeping”, Hewlett Packard Application Note 1289, 1997

[M3]: Balasubramanian, Sivaram, Harris Kendal R., and Moldovansky, Anatoly “A Frequency Compensated Clock for Precision Synchronization using IEEE 1588 Protocol and its Application to Ethernet*”, Workshop on IEEE 1588*, Gaithersburg, MD, USA, 2003.

[M4]: Eidson, John C. et al., “Method for recognizing events and synchronizing clocks,” U.S. Patent 5,566,180, October 15, 1996.

[M5]: IEEE EUI-64, IEEE EUI-48, and IEEE MAC-48 assigned numbers may be obtained from the IEEE Registration Authority, http://standards.ieee.org/regauth/ Tutorials on these assigned numbers may be found on this web site.

[M6]: IETF RFC-1305 (1992) “Network Time Protocol (Version 3)”, Mills, David L., March 1992. http://ietfreport.isoc.org/rfc/rfc1305.txt

[M7]: IETF RFC 1589 (1994) “A Kernel Model for Precision Timekeeping”, Mills, David L., March 1994. http://ietfreport.isoc.org/rfc/rfc1589.txt

[M8]: IETF RFC 1624 (1994) “Computation of the Internet Checksum via Incremental Update”, Rijsinghani, A. Ed., May 1994. http://ietfreport.isoc.org/rfc/rfc1624.txt

[M9]: IETF RFC 2104 (1997) “HMAC: Keyed-Hashing for Message Authentication”, Krawczyk, H., Bellare, M., and Canetti, R., February 1997. http://ietfreport.isoc.org/rfc/rfc2104.txt

[M10]: IETF RFC 2404 (1998), “The Use of HMAC-SHA-1-96 within ESP and AH”, Madson, C. November 1998. http://ietfreport.isoc.org/rfc/rfc2404.txt

[M11]: IETF RFC 2460 (1998) “Internet Protocol, Version 6 (IPv6) Specification”, Deering, S. and Hinden, R. December 1998. http://ietfreport.isoc.org/rfc/rfc2460.txt

[M12]: IETF RFC 2783 (2000), “Pulse-Per-Second API for UNIX-like Operating Systems”, Mogul, J. and Stone, J. March 2000. http://ietfreport.isoc.org/rfc/rfc2783.txt

[M13]: IETF RFC 4291 (2006), “IP Version 6 Addressing Architectre”, Hinden, R. and Deering, S. February 2006. http://www.ietf.org/rfc/rfc4291.txt

[M14]: IETF RFC 768 (1980) “User Datagram Protocol”, Postel, J. August 1980. http://ietfreport.isoc.org/rfc/rfc768.txt

[M15]: IETF RFC 791 (1981) “Internet Protocol”, Postel, J. September 1981. http://ietfreport.isoc.org/rfc/rfc791.txt

[M16]: ISO/IEC 9945:2003 Information technology — Portable Operating System Interface (POSIX®)

[M17]: ISO8601: ISO 8601:2004, Data elements and interchange formats– Information interchange– Representation of dates and times

[M18]: Items and pointers on <http://tycho.usno.navy.mil/time.html>, a web page maintained by the U.S. Naval Observatory.

[M19]: ITU-T Recommendation G.810, *Definitions and Terminology for Synchronization Networks*, ITU-T, Geneva, August, 1996, Corregendum 1, November, 2001.

[M20]: National Institute of Standards and Technology: Secure Hash Signature Standard (SHS) (FIPS PUB 180-2) http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf or http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

[M21]: National Institute of Standards and Technology: The Keyed-Hash Message Authentication Code (HMAC) (FIPS PUB 180-2), http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf.

[M22]: Perlman, Radia, *Interconnections, Bridges and Routers*, Addison-Wesley, 1992 ISBN 0-201-56332-0.

[M23]: SERVICE DE LA ROTATION TERRESTRE, OBSERVATOIRE DE PARIS, 61, Av. De l'Observatoire 75014 PARIS (France).

[M24]: Sullivan, D.B., Allan, D.W., Howe, D.A., Walls, F.L. editors, "Characterization of clocks and oscillators," *NIST technical note 1337*, March 1990.

[M25]: Wang, Sihai, Cho, Jaehun and Garner, Geoffrey M., "Improvements to boundary Clock Based Time Synchronization through Cascaded Switches", *2006 Conference on IEEE 1588*, Gaithersburg, MD, USA, October 2 – 4, 2006.

[M26](International Vocabulary of Basic and General Terms in Metrology (VIM), BIPM, IEC, IFCC, ISO, IUPAC, IUPAP, OIML, 2nd ed., 1993, definition 6.10) (Editor: I will put this in order after the edits are complete)