# IEEE P1904.2™/D0.X
# Draft Standard for Management Channel for Customer-Premises Equipment Connected to Ethernet-based Subscriber Access Networks

Sponsor

**Standards Development Board**
of the
**IEEE Communications Society**

Approved <XX MONTH 20XX>

**IEEE-SA Standards Board**

**Abstract:** This standard TBD

**Keywords:** TBD

The Institute of Electrical and Electronics Engineers, Inc.
3 Park Avenue, New York, NY 10016-5997, USA



**PDF: ISBN 978-0-XXXX-XXXX-X STDXXXXX**
**Print: ISBN 978-0-XXXX-XXXX-X STDPDXXXXX**

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

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

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

The existence of an IEEE Standard does not imply that there are no other ways to produce, test, measure, purchase, market, or provide other goods and services related to the scope of the IEEE Standard. Furthermore, the viewpoint expressed at the time a standard is approved and issued is subject to change brought about through developments in the state of the art and comments received from users of the standard. Every IEEE Standard is subjected to review at least every five years for revision or reaffirmation, or every ten years for stabilization. When a document is more than five years old and has not been reaffirmed, or more than ten years old and has not been stabilized, it is reasonable to conclude that its contents, although still of some value, do not wholly reflect the present state of the art. Users are cautioned to check to determine that they have the latest edition of any IEEE Standard.

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

Interpretations: Occasionally questions may arise regarding the meaning of portions of standards as they relate to specific applications. When the need for interpretations is brought to the attention of IEEE, the Institute will initiate action to prepare appropriate responses. Since IEEE Standards represent a consensus of concerned interests, it is important to ensure that any interpretation has also received the concurrence of a balance of interests. For this reason, IEEE and the members of its societies and Standards Coordinating Committees are not able to provide an instant response to interpretation requests except in those cases where the matter has previously received formal consideration. A statement, written or oral, that is not processed in accordance with the IEEE-SA Standards Board Operations Manual shall not be considered the official position of IEEE or any of its committees and shall not be considered to be, nor be relied upon as, a formal interpretation of the IEEE. At lectures, symposia, seminars, or educational courses, an individual presenting information on IEEE standards shall make it clear that his or her views should be considered the personal views of that individual rather than the formal position, explanation, or interpretation of the IEEE.

Comments for revision of IEEE Standards are welcome from any interested party, regardless of membership affiliation with IEEE. Suggestions for changes in documents should be in the form of a proposed change of text, together with appropriate supporting comments. Recommendations to change the status of a stabilized standard should include a rationale as to why a revision or withdrawal is required. Comments and recommendations on standards, and requests for interpretations should be addressed to:

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

Authorization to photocopy portions of any individual standard for internal or personal use is granted by The Institute of Electrical and Electronics Engineers, Inc., provided that the appropriate fee is paid to Copyright Clearance Center. To arrange for payment of licensing fee, please contact Copyright Clearance Center, Customer Service, 222 Rosewood Drive, Danvers, MA 01923 USA; +1 978 750 8400. Permission to photocopy portions of any individual standard for educational classroom use can also be obtained through the Copyright Clearance Center.

# Introduction

This introduction is not part of IEEE P1904.2/D0.1

This standard TBD …

# Notice to users

## Laws and regulations

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

## Copyrights

This document is copyrighted by the IEEE. It is made available for a wide variety of both public and private uses. These include both use, by reference, in laws and regulations, and use in private self-regulation, standardization, and the promotion of engineering practices and methods. By making this document available for use and adoption by public authorities and private users, the IEEE does not waive any rights in copyright to this document.

## Updating of IEEE documents

Users of IEEE Standards documents should be aware that these documents may be superseded at any time by the issuance of new editions or may be amended from time to time through the issuance of amendments, corrigenda, or errata. An official IEEE document at any point in time consists of the current edition of the document together with any amendments, corrigenda, or errata then in effect. In order to determine whether a given document is the current edition and whether it has been amended through the issuance of amendments, corrigenda, or errata, visit the IEEE-SA Website at http://standards.ieee.org/index.html or contact the IEEE at the address listed previously. For more information about the IEEE Standards Association or the IEEE standards development process, visit the IEEE-SA Website at http://standards.ieee.org/index.html.

## Errata

Errata, if any, for this and all other standards can be accessed at the following URL: http://standards.ieee.org/findstds/errata/index.html. Users are encouraged to check this URL for errata periodically.

## Interpretations

Current interpretations can be accessed at the following URL: http://standards.ieee.org/findstds/interps/index.html.

## Patents

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 by the IEEE with respect to the existence or validity of any patent rights in connection therewith. If a patent holder or patent applicant has filed a statement of assurance via an Accepted Letter of Assurance, then the statement is listed on the IEEE-SA website http://standards.ieee.org/about/sasb/patcom/patents.html. Letters of Assurance may indicate whether the Submitter is willing or unwilling to grant licenses under patent 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.

Essential Patent Claims may exist for which a Letter 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 provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or nondiscriminatory. Users of this standard are expressly advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility. Further information may be obtained from the IEEE Standards Association.

# Participants

At the time this draft standard was submitted to the IEEE-SA Standards Board for approval, the following is a place holder:

, *Working Group Chair*
, *Editor*

The following individuals submitted technical contributions or commented on the draft standard at various stages of the project development.

Name

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

***(to be supplied by IEEE)***

| | | |
|---|---|---|
| Balloter1 | Balloter4 | Balloter7 |
| Balloter2 | Balloter5 | Balloter8 |
| Balloter3 | Balloter6 | Balloter9 |

When the IEEE-SA Standards Board approved this standard on <XX MONTH 20XX>, it had the following membership:

***(to be supplied by IEEE)***

**<Name>,** *Chair*
**<Name>,** *Vice Chair*
**<Name>,** *Past President*
**<Name>,** *Secretary*

SBMember1
SBMember2
SBMember3
SBMember4
SBMember5
SBMember6
SBMember7
SBMember8
SBMember9

*Member Emeritus

Also included are the following nonvoting IEEE-SA Standards Board liaisons:

<Name>, *NRC Representative*
<Name>, *DOE Representative*
<Name>, *NIST Representative*

<Name>
*IEEE Standards Program Manager, Document Development*

<Name>
*IEEE Standards Program Manager, Technical Program Development*

## Contents

# 1 Overview

## 1.1 Scope

This standard TBD ...

## 1.2 Purpose

The purpose of this standard is to TBD …

## 1.3 Coverage

This specification provides TBD ...

## 1.4 Overview of clauses

This subclause provides an overview of the scope of individual clauses included in this specification, namely:

— TBD ...

# 2 Normative references

The following referenced documents are indispensable for the application of this document (i.e., they must be understood and used, so each referenced document is cited in text and its relationship to this document is explained). For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments or corrigenda) applies.

# 3 Definitions, acronyms, and abbreviations

## 3.1 Definitions

For the purposes of this document, the following terms and definitions apply. The IEEE Standards Dictionary Online should be consulted for terms not defined in this clause.[1]

TBD

## 3.2 Acronyms and abbreviations

UMT - Universal Management Tunnel

UMTDP - Universal Management Tunnel Discovery Protocol

## 3.3 Special Terms

**Term**: Definition

## 3.4 Notation for state diagrams

All the state diagrams used in this standard meet the set of requirements included in the following subclauses.

### 3.4.1 General conventions

The operation of any protocol defined in this standard can be described by subdividing the protocol into a number of interrelated functions. The operation of the functions can be described by state diagrams. Each diagram represents the domain of a function and consists of a group of connected, mutually exclusive states. Only one state of a function is active at any given time (see Figure 3-1).

[1] IEEE Standards Dictionary Online subscription is available at http://www.ieee.org/portal/innovate/products/standard/standards_dictionary.html.

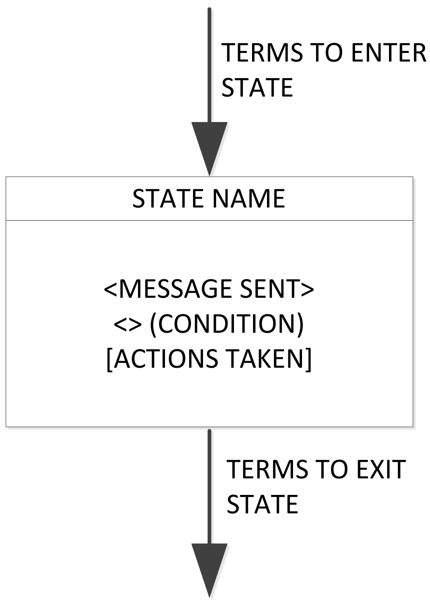



**Figure 3-1—State diagram notation example**

#### 3.4.1.1 Representation of states

Each state that the function can assume is represented by a rectangle. These are divided into two parts by a horizontal line. In the upper part the state is identified by a name in capital letters. The lower part contains the body of the given state, containing description of the actions taken in this state, as defined in 3.4.3.

#### 3.4.1.2 Transitions

All permissible transitions between the states of a function are represented graphically by arrows between them. A transition that is global in nature (for example, an exit condition from all states to the IDLE or RESET state) is indicated by an open arrow (an arrow with no source block). Global transitions are evaluated continuously whenever any state is evaluating its exit conditions. When the condition for a global transition becomes true, it supersedes all other transitions, including Unconditional Transition (UCT), returning control to the block pointed to by the open arrow.

Labels on transitions are qualifiers that are required to be fulfilled before the transition is taken. The label UCT designates an unconditional transition. Qualifiers described by short phrases are enclosed in parentheses.

The following terms are valid transition qualifiers:

- Boolean expressions
- An event such as the expiration of a timer: timer_done
- An event such as the reception of a message: MAC_DATA.indication

— An unconditional transition: UCT

— A branch taken when other exit conditions are not satisfied: ELSE

State transitions occur instantaneously. No transition in the state diagram can cross another transition. When possible, any two transitions with different logical conditions are not joined together into a single transition line.

### 3.4.2 State diagrams and accompanying text

State diagrams take precedence over text.

### 3.4.3 Actions inside state blocks

The actions inside a state block execute instantaneously. Actions inside state blocks are atomic (i.e., uninterruptible).

After performing all the actions listed in a state block one time, the state diagram then continuously evaluates exit conditions for the given state block until one is satisfied, at which point control passes through a transition arrow to the next block. While the state awaits fulfillment of one of its exit conditions, the actions inside do not implicitly repeat.

Valid state actions may include generation of *indication* and *request* primitives.

No actions are taken outside of any blocks of the state diagram.

### 3.4.4 State diagram variables

Once set, variables retain their values as long as succeeding blocks contain no references to them.

Setting the parameter of a formal interface message assures that, on the next transmission of that message, the last parameter value set is transmitted.

Testing the parameter of a formal interface message tests the value of that message parameter that was received on the last transmission of said message. Message parameters may be assigned default values that persist until the first reception of the relevant message.

### 3.4.5 Operators

The state diagram operators are shown in Table 3-1.

**Table 3-1—State diagram operators**

| Character | Meaning |
|---|---|
| AND | Boolean AND |
| OR | Boolean OR |
| XOR | Boolean XOR |
| ! | Boolean NOT |
| < | Less than |
| > | More than |
| ≤ | Less than or equal to |
| ≥ | More than or equal to |
| == | Equals (a test of equality) |
| != | Not equals |
| () | Indicates precedence |

| Character | Meaning |
|---|---|
| = | Assignment operator |
| \| | Concatenation operation that combines several sub-fields or parameters into a single aggregated field or parameter |
| else | No other state condition is satisfied |
| true | Designation of a Boolean value of TRUE |
| false | Designation of a Boolean value of FALSE |

#### 3.4.6 Timers

Some of the state diagrams use timers for various purposes, e.g., measurement of time, and confirmation of activity. All timers operate in the same fashion.

A timer is reset and starts counting upon entering a state where [start x_timer, x_timer_value] is asserted. Time "x" after the timer has been started, "x_timer_done" is asserted and remains asserted until the timer is reset. At all other times, "x_timer_not_done" is asserted.

When entering a state where [start x_timer, x_timer_value] is asserted, the timer is reset and restarted even if the entered state is the same as the exited state.

Any timer can be stopped at any time upon entering a state where [stop x_timer] is asserted, which aborts the operation of the "x_timer" asserting "x_timer_not_done" indication until the timer is restarted again.

#### 3.4.7 Hexadecimal notation

Numerical values designated by the 0x prefix indicate a hexadecimal notation of the corresponding number, with the least significant bit shown on the right. For example: 0x0F represents an 8-bit hexadecimal value of the decimal number 15; 0x00-00-00-00 represents a 32-bit hexadecimal value of the decimal number 0; 0x11-AB-11-AB represents a 32-bit hexadecimal value of the decimal number 296423851.

#### 3.4.8 Binary notation

Numerical values designated by the 0b prefix indicate a binary notation of the corresponding number, with the least significant bit shown on the right. For example: 0b0001000 represents an 8-bit binary value of the decimal number 8.

### 3.5 Notation for PICS

The supplier of a device implementation that is claimed to conform to this standard is required to complete a protocol implementation conformance statement (PICS) proforma.

A completed PICS proforma is the PICS for the implementation in question. The PICS is a statement of which capabilities and options of this standard have been implemented. The PICS can be used for a variety of purposes by various parties, including the following:

a) As a checklist by the protocol implementer, to reduce the risk of failure to conform to the standard through oversight;

b) As a detailed indication of the capabilities of the implementation, stated relative to the common basis for understanding provided by the standard PICS proforma, by the supplier and acquirer, or potential acquirer, of the implementation;

c) As a basis for initially checking the possibility of interworking with another implementation by the user, or potential user, of the implementation (note that, while interworking can never be guaranteed, failure to interwork can often be predicted from incompatible PICS);

d) As the basis for selecting appropriate tests against which to assess the claim for conformance of the implementation, by a protocol tester.

Each PICS entry is uniquely identified by an item number, with the following form: [Package][Device]-[Feature][Number], where:

— [Package] is the designation of the given Package,

— [Device] identifies whether the given PICS item describes the ONU (U) or OLT (T) requirements,

— [Feature] is the identification of individual features, and finally,

— [Number] is a number allocated to each subsequent PICS entry. This item may have one of two possible formats: a decimal number or a decimal number followed by a lower-case letter. The first format is used to designate PICS with functionally distinct requirements. The latter format is used to designate PICS with functionally similar requirements.

For example, CU-LPTK3a represents a PICS entry for an ONU compliant with Package C for the “optical link protection, trunk type” feature, item 3, subitem a.

### 3.5.1 Abbreviations and special symbols

The following symbols are used in the PICS proforma:

| | |
|---|---|
| M | mandatory field/function |
| ! | negation |
| O | optional field/function |
| O.<n> | optional field/function, but at least one of the group of options labeled by the same numeral <n> is required |
| O/<n> | optional field/function, but one and only one of the group of options labeled by the same numeral <n> is required |
| X | prohibited field/function |
| <item>: | simple-predicate condition, dependent on the support marked for <item> |
| <item1>*<item2>: | AND-predicate condition, the requirement needs to be met if both optional items are implemented |

### 3.5.2 Instructions for completing the PICS proforma

The first part of the PICS proforma, Implementation Identification and Protocol Summary, is to be completed as indicated with the information necessary to identify fully both the supplier and the implementation.

The main part of the PICS proforma is a fixed-format questionnaire divided into subclauses, each containing a group of items. Answers to the questionnaire items are to be provided in the right-most column, either by simply marking an answer to indicate a restricted choice (usually Yes, No, or Not Applicable), or by entering a value or a set or range of values. (Note that there are some items where two or more choices from a set of possible answers can apply; all relevant choices are to be marked.)

Each item is identified by an item reference in the first column; the second column contains the question to be answered; the third column contains the reference or references to the material that specifies the item in the main body of the standard; the fourth column contains values and/or comments pertaining to the question to be answered. The remaining columns record the status of the items—whether the support is mandatory, optional or conditional—and provide the space for the answers.

The supplier may also provide, or be required to provide, further information, categorized as either Additional Information or Exception Information. When present, each kind of further information is to be provided in a further subclause of items labeled A<i> or X<i>, respectively, for cross-referencing purposes, where <i> is any unambiguous identification for the item (e.g., simply a numeral); there are no other restrictions on its format or presentation.

A completed PICS proforma, including any Additional Information and Exception Information, is the protocol implementation conformance statement for the implementation in question.

Note that where an implementation is capable of being configured in more than one way, according to the items listed under Major Capabilities/Options, single PICS may be able to describe all such configurations. However, the supplier has the choice of providing more than one PICS, each covering some subset of the implementation's configuration capabilities, if that would make presentation of the information easier and clearer.

### 3.5.3 Additional information

Items of Additional Information allow a supplier to provide further information intended to assist the interpretation of the PICS. It is not intended or expected that a large quantity be supplied, and the PICS can be considered complete without any such information. Examples might be an outline of the ways in which a (single) implementation can be set up to operate in a variety of environments and configurations; or a brief rationale, based perhaps upon specific application needs, for the exclusion of features that, although optional, are nonetheless commonly present in implementations.

References to items of Additional Information may be entered next to any answer in the questionnaire, and may be included in items of Exception Information.

### 3.5.4 Exception information

It may occasionally happen that a supplier wishes to answer an item with mandatory or prohibited status (after any conditions have been applied) in a way that conflicts with the indicated requirement. No pre-printed answer is found in the Support column for this; instead, the supplier is required to write into the Support column an X<i> reference to an item of Exception Information, and to provide the appropriate rationale in the Exception item itself.

An implementation for which an Exception item is required in this way does not conform to this standard. Note that a possible reason for the situation described above is that a defect in the standard has been reported, a correction for which is expected to change the requirement not met by the implementation.

### 3.5.5 Conditional items

The PICS proforma may contain conditional items. These are items for which both the applicability of the item itself, and its status if it does apply—mandatory, optional, or prohibited—are dependent upon whether or not certain other items are supported.

Individual conditional items are indicated by a conditional symbol of the form "<item>:<s>" in the Status column, where "<item>" is an item reference that appears in the first column of the table for some other item, and "<s>" is a status symbol, M (Mandatory), O (Optional), or X (Not Applicable).

If the item referred to by the conditional symbol is marked as supported, then:

a) the conditional item is applicable,

b) its status is given by "<s>", and

c) the support column is to be completed in the usual way.

Each item whose reference is used in a conditional symbol is indicated by an asterisk in the Item column.

# 4 Universal Management Tunnel (UMT)

Editorial Note: this Clause will describe the UMT architecture, showing a single UMT domain interconnecting multiple L2 domains with UMT switches, and showing UMT instance between two UMT end-points. Description of the individual device functions follows (tentative names are used)

## 4.1 Overview

### 4.1.1 Scope

This clause defines the Universal Management Tunnel (UMT) which is intended to be a supplemental layer in the IEEE 802 architecture. The UMT provides a mechanism for transmitting service data units for higher layer protocols across a layer-2 network in which those protocols would not normally be forwarded due to addressing conflicts or other factors.

UMT data from client entities is conveyed in frames called UMT Protocol Data Units (UMTPDUs). UMTPDUs contain the appropriate information to identify the encapsulated protocol for delivery to the correct receiving entity. UMTPDUs traverse one or more links and are passed between peer UMT entities, therefore UMTPDUs are forwarded by MAC clients (e.g. bridges or switches).

~~This standard will describe a management channel for customer-premises equipment (CPE) connected to Ethernet-based subscriber access networks. The key characteristics of the specified management channel are:~~

~~Multi-hop capabilities to allow management of various CPE devices located behind an Optical Network Unit (ONU), a Coaxial Network Unit (CNU), a Residential Gateway (RGW), etc.~~

~~Extensibility to accommodate new management protocols and/or new types of CPE devices.~~

~~Broadcast/multicast capabilities to allow simultaneous (synchronized) configuration of multiple devices.~~

~~Encryption capabilities to ensure secure access to managed CPE devices by the network operators.~~

~~The standard will describe the message format as well as processing operations and forwarding rules at the intermediate nodes.~~

### 4.1.2 Summary of objectives and major concepts

This subclause provides details and functional requirements for the UMT objectives:

a) Bridge/Switch traversal: A mechanism is defined to forward UMTPDUs across bridges and switches.

b) Allow a single UMT Client entity to send messages to one or more peer entities simultaneously using multicast or broadcast messages.

c) Allow the protocol to be extended to accommodate new client protocols to be supported in the future.

### 4.1.3 Summary of non-objectives

This subclause explicitly lists certain functions that are not addressed by UMT. These functions, while valuable, do not fall within the scope of this standard.

a) Tunnel state/status: This standard does not define a tunnel state or status maintenance method. UMT is a stateless protocol.

b) UMT Peer discovery: Discovery of UMT peers in the UMT network is out of scope of this standard. This standard does not define a UMT-specific method to discover or detect UMT peers.

c) Router traversal: Router traversal is out of scope of this standard. This standard does not define methods to forward UMTPDUs across Network-Layer clients (e.g. IP routers, IP hosts).

### 4.1.4 Positioning of UMT in the IEEE 802 Architecture

UMT comprises an optional sublayer between a superior sublayer (e.g., MAC Client) and a subordinate sublayer (e.g., MAC or optional MAC Control sublayer). UMT is also composed of a shim between the MAC and MAC Relay entities. Figure 4-1 shows the architectural relationship of the UMT layer to the MAC, MAC Clients and UMT Clients.

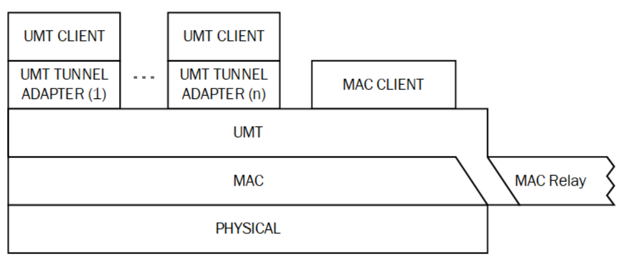



**Figure 4-1 - UMT relationship to the IEEE 802 model**

### 4.1.5 Compatibility Considerations

#### 4.1.5.1 Application

UMT is intended for use in IEEE 802 networks. Nothing in this standard disallows implementation of UMT on non-IEEE 802 networks, but description of such implementation is out of the scope of this standard.

A conformant implementation may implement the UMT layer for some ports within a system while not implementing it for other ports on the same system.

#### 4.1.5.2 Interoperability between UMT capable DTEs

A DTE is able to determine whether or not a remote DTE has UMT functionality enabled. The optional UMT Discovery mechanism described in the annex discovers the presence of UMT peers and their configured parameters, such as maximum allowable UMTPDU size, and supported UMT Client protocols.

#### 4.1.5.3 Interface to MAC Clients

The UMT Layer described in this standard implements a transparent pass-through for MAC clients that generate the MA_DATA.request service primitive (and expect the MA_DATA.indication service primitive). In some cases, such as OAM described in IEEE Std. 802.3 Clause 57, a protocol might be required to operate as a MAC client and as a UMT Client.

This standard may describe in text or depict in figures such protocols as having multiple instances – one at the native position in the protocol stack and another at the UMT Client position in the protocol stack. This depiction is intended only to clarify the intended operation of the protocol with respect to UMT and not to specify the method of implementation.

Similarly, it is out of the scope of this standard to describe the position and operation of all possible MAC clients and MAC functions (such as MAC Control) relative to UMT. Where this standard is silent on the operation of a protocol relative to UMT's transparent pass-through functionality for MAC clients, that protocol shall conform to its specification and operate as if UMT were not present.

## 4.2 Functional Specifications

### 4.2.1 Interlayer Service Interfaces

Figure 4-2 depicts the usage of interlayer interfaces by the UMT layer.

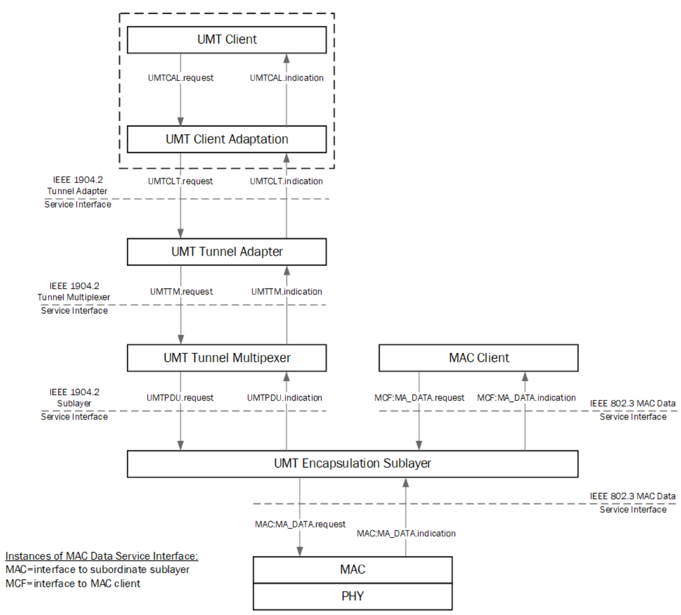



**Figure 4-2 - UMT interlayer service interfaces**

### 4.2.2 Principles of Operation

UMT employs the following principles and concepts:

a) UMTPDUs traverse a single bridging domain and are passed between UMT Layer entities. UMTPDUs are forwarded by intermediate bridges according to IEEE Std. 802.1Q and IEEE Std. 802.1D.

b) UMT is a stateless/connectionless transmission method.

c) The UMT Layer presents a standard IEEE 802.3 MAC service interface to the superior sublayer, which is the MAC Client.

d) The UMT Layer employs a standard IEEE 802.3 MAC service interface to the subordinate sublayer. Subordinate sublayers include MAC and MAC Control.

e) Frames from superior sublayers are multiplexed within the UMT Layer with UMTPDUs.

f) The UMT Layer parses received frames and passes UMTPDUs to the UMT Tunnel Multiplexer. Non-UMTPDUs are passed to the superior sublayer.

g) Knowledge of the underlying Physical Layer device is not required by the UMT Layer.

h) The UMT Tunnel Multiplexer parses received UMTPDUs and passes them to an appropriate UMT Tunnel Adapter based on tunnel identifying fields, the source MAC address and destination MAC address.

i) The UMT Tunnel Adapter in UMT unicast operation emulates a point-to-point link to the remote UMT peer.

j) The UMT Tunnel Adapter parses received UMTPDUs and passes the UMT Client service data to the appropriate UMT Client.

k) The optional UMT Client Adaptation layer is an abstract layer that adapts the UMT Client service interface to the UMT Tunnel Adapter service interface.

l) The UMT Client can be any protocol layer that could normally exist above MAC Control.

### 4.2.3 Instances of the MAC data service interface

A superior sublayer such as the MAC client communicates with the UMT Layer using the standard MAC data service interface specified in IEEE Std. 802.3 Clause 2. Similarly, the UMT Layer communicates with a subordinate sublayer such as the MAC Control or MAC using the same standard service interfaces.

This clause uses two instances of the MAC data service interface, therefore it is necessary to introduce a notation convention so that the reader can be clear as to which interface is being referred to at any given time. A prefix is therefore assigned to each service primitive, indicating which of the two interfaces is being invoked, as depicted in Figure 4-2. The prefixes are as follows:

a) MCF:, for primitives issued on the interface between the superior sublayer and the UMT Layer (MCF is an abbreviation for MAC client frame)

b) MAC:, for primitives issued on the interface between the underlying subordinate sublayer (e.g., MAC) and the UMT Layer

### 4.2.4 UMT Client

The UMT Client is the functional block that uses the UMT to forward data across the UMT network.

#### 4.2.4.1 Responsibilities of the UMT Client

The UMT Client makes requests to the UMT Client Adaption layer to send data across the UMT. The UMT Client also listens to the UMT Client Adaptation Layer for incoming data. Generally, interactions with the peer UMT Client is out of the scope of this standard. Informative annexes have been included to guide implementors in the use of UMT. In some cases, the annex has been made normative.

#### 4.2.4.2 UMT Client Interactions

The UMT Client entity communicates with the UMT Client Adaptation using the following interlayer service interfaces:

UMTCAL.request

UMTCAL.indication

The UMTCAL.request and UMTCAL.indication, service primitives described in this subclause are mandatory.

##### 4.2.4.2.1 UMTCAL.request

###### 4.2.4.2.1.1 Function

This primitive defines the transfer of data from an UMT Client entity to the UMT Client Adaptation entity. This primitive is specific to the UMT Client protocol and is implementation specific. Informative annexes have been included to guide implementors. In some cases, the annex has been made normative.

###### 4.2.4.2.1.2 Semantics of the service primitive

The semantics of the primitive are implementation specific.

###### 4.2.4.2.1.3 When Generated

This primitive is generated by the UMT Client entity whenever a client PDU is to be transferred to a peer entity.

###### 4.2.4.2.1.4 Effect of Receipt

The receipt of this primitive will cause the UMT Client Adaptation entity to perform any required parsing and transformations of the received parameters necessary to send the UMT Client PDU over the UMT. After performing these actions, the UMT Client Adaptation entity asserts the UMTCLT.request primitive to the UMT Tunnel Adapater according to the procedures described in 4.2.5.2.1.

##### 4.2.4.2.2 UMTCAL.indication

###### 4.2.4.2.2.1 Function

This primitive defines the transfer of data from a UMT Client Adaptation entity to a UMT Client entity. This primitive is specific to the UMT Client protocol and is implementation specific. Informative annexes have been included to guide implementors. In some cases, the annex has been made normative.

###### 4.2.4.2.2.2 Semantics of the service primitive

The semantics of the primitive are implementation specific.

##### 4.2.4.2.2.3 When Generated

This primitive is passed from the UMT Client Adaptation entity to the UMT Client entity to indicate the arrival of a UMTPDU to the local UMT Client. Such UMTPDUs are reported only if they are validly formed and received without error.

##### 4.2.4.2.2.4 Effect of Receipt

The effect of receipt of this primitive by the UMT Client is unspecified.

### 4.2.5 UMT Client Adaptation

#### 4.2.5.1 Responsibilities of the UMT Client Adaptation

The UMT Client Adaptation is an intermediate layer that adapts the UMT Client interfaces to the UMT Tunnel Adapter interfaces. The UMT Client Adaptation receives transmit requests from the UMT Client via the UMTCAL.request primitive, transforms those requests as needed, and passes the results into the UMT Tunnel Adapter via the UMTCLT.request primitive. In similar fashion, the UMT Client Adaptation receives incoming data via the UMTCLT.indication primitive, transforms those requests as needed, and passes the results into the UMT Client via the UMTCAL.indication primitive.

The required transformations are specific to the UMT Client entity and are left unspecified. Example UMT Client Adaptations are provided in the Annex.

#### 4.2.5.2 UMT Client Adaptation Interactions

The UMT Client Adaptation entity communicates with the UMT Tunnel Adapter using the following interlayer service interfaces:

UMTCLT.request

UMTCLT.indication

The UMTCLT.request and UMTCLT.indication, service primitives described in this subclause are mandatory.

##### 4.2.5.2.1 UMTCLT.request

###### 4.2.5.2.1.1 Function

This primitive defines the transfer of data from an UMT Client Adaptation entity to a UMT Tunnel Adapter entity.

###### 4.2.5.2.1.2 Semantics of the service primitive

The semantics of the primitive are as follows:

```
UMTCLT.request (
                umt_subtype,
                umt_client_sdu
                )
```

The umt_subtype is used to identify the intended UMT Client entity and is used to populate the Subtype field of the UMTPDU. The umt_client_sdu parameter is used to create the Data field within the UMTPDU to be transmitted.

##### 4.2.5.2.1.3 When generated

This primitive is generated by the UMT Client Adaptation entity whenever a client PDU is to be transferred to a peer entity using UMT.

##### 4.2.5.2.1.4 Effect of Receipt

The receipt of this primitive will cause the UMT Tunnel Adapter entity to multiplex the request with requests from other UMT Client entities and assert the UMTTM.request primitive to the UMT Tunnel Multiplexer according to the procedures described in 4.2.7.3.1.

#### 4.2.5.2.2 UMTCLT.indication

##### 4.2.5.2.2.1 Function

This primitive defines the transfer of data from an UMT Tunnel Adapter entity to an UMT Client Adaptation Layer entity.

##### 4.2.5.2.2.2 Semantics of the service primitive

The semantics of the primitive are as follows:

```
UMTCLT.indication (
                destination_address,
                source_address,
                umt_subtype,
                umt_client_sdu
                )
```

The value of the destination_address parameter is copied from the destination_address parameter received in the UMTTM.indication primitive. The value of the source_address parameter is copied from the source_address parameter received in the UMTTM.indication primitive. The value of the umt_client_sdu parameter is copied from the umt_client_sdu parameter received in the UMTPDU.indication primitive.

##### 4.2.5.2.2.3 When generated

This primitive is passed from the UMT Tunnel Adapter entity to the UMT Client Adaptation entity to indicate the arrival of a UMTPDU to the local UMT Client. Such UMTPDUs are reported only if they are validly formed and received without error.

##### 4.2.5.2.2.4 Effect of Receipt

The receipt of this primitive will cause the UMT Client Adaptation entity to perform any required parsing and transformations. After performing these actions, the UMT Client Adaptation entity asserts the UMTCAL.indication primitive to the UMT Client Adaptation entity according to the procedures described in 4.2.4.2.2.

#### 4.2.6 UMT Tunnel Adapter

##### 4.2.6.1 Responsibilties of the UMT Tunnel Adapter

The UMT Tunnel Adpater multiplexes requests from multiple UMT Clients and passes them to the UMT Tunnel Multiplexer by asserting the UMTTM.request primitive. Similarly, the UMT Tunnel Adapter layer receives UMTPDUs from the UMT Tunnel Multiplexer via the UMTTM.indication primitive and parses the UMTPDUs for delivery to the UMT Client designated by the Subtype field. Delivery to the UMT Tunnel Client Adaptation entity occurs via assertion of the UMTCLT.indication primitive.

##### 4.2.6.2 Block Diagram

Figure 4-3 depicts the major blocks within the UMT Tunnel Adapater and their interrelationships with one another and external entities.

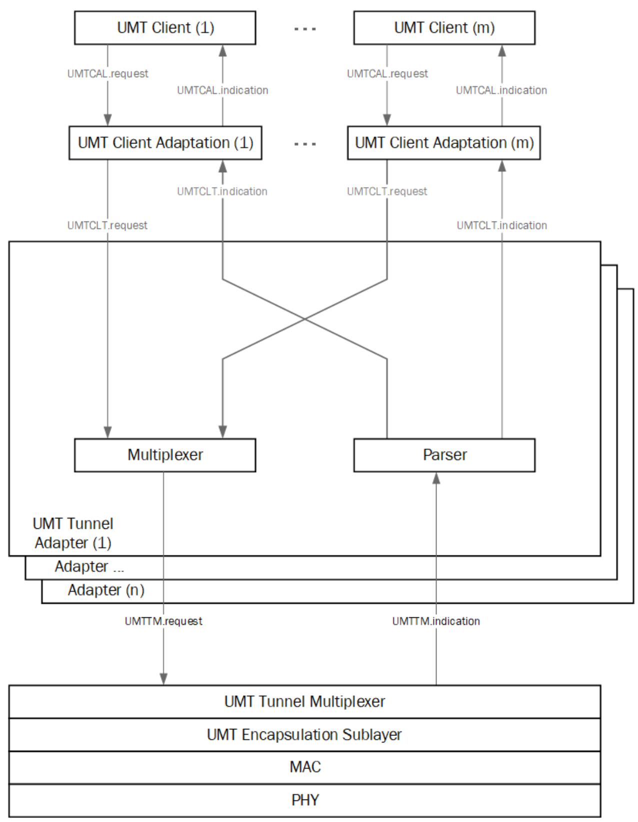



**Figure 4-3 - UMT Tunnel Adapter Block Diagram**

#### 4.2.6.3 UMT Tunnel Adpater Interactions

The UMT Tunnel Adapter entity communicates with the UMT Tunnel Multiplexer using the following interlayer service interfaces:

UMTTM.request

UMTTM.indication

The UMTTM.request and UMTTM.indication service primitives described in this subclause are mandatory.

##### 4.2.6.3.1 UMTTM.request

###### 4.2.6.3.1.1 Function

This primitive defines the transfer of data from an UMT Tunnel Adapater entity to the UMT Tunnel Multiplexer entity.

###### 4.2.6.3.1.2 Semantics of the service primitive

The semantics of the primitive are as follows:

UMTTM.request (

umt_tunnel_id

umt_subtype,

umt_client_sdu

)

The umt_tunnel_id parameter identifies the specific UMT Tunnel Adapter instance asserting the service primitive and is used by the UMT Tunnel Multiplexer to assign the source MAC address and destination MAC address. The umt_client_data parameter is used to create the Data field within the UMTPDU to be transmitted. The umt_subtype and umt_client_sdu are copied from the UMTCLT.request primitive.

###### 4.2.6.3.1.3 When Generated

This primitive is generated by the UMT Tunnel Adapter entity whenever an UMTPDU is to be transferred to a peer entity.

###### 4.2.6.3.1.4 Effect of Receipt

The receipt of this primitive will cause the UMT Tunnel Multiplexer entity to multiplex the request with requests from other UMT Tunnel Adapter entities and assert the UMTPDU.request primitive to the UMT Encapsulation Sublayer according to the procedures described in 4.2.7.3.1.

##### 4.2.6.3.2 UMTTM.indication

###### 4.2.6.3.2.1 Function

This primitive defines the transfer of data from the UMT Tunnel Multiplexer entity to a single instance of a UMT Tunnel Adapter entity.

#### 4.2.6.3.2.2 Semantics of the service primitive

UMTTM.indication (

umt_tunnel_id,

destination_address,

source_address,

umt_subtype,

umt_client_sdu

)

The umt_tunnel_id parameter identifies the specific UMT Tunnel Adapter instance to which the service primitive is being addressed. The value of the destination_address parameter is copied from the destination_address parameter received in the UMTPDU.indication primitive. The value of the source_address parameter is copied from the source_address parameter received in the UMTPDU.indication primitive. The value of the umt_client_sdu parameter is copied from the umt_client_sdu parameter received in the UMTPDU.indication primitive.

#### 4.2.6.3.2.3 When Generated

This primitive is passed from the UMT Tunnel Multiplexer entity to a single instance of a UMT Tunnel Adapter entity to indicate the arrival of a UMTPDU to a local UMT Client. Such UMTPDUs are reported only if they are validly formed and received without error.

#### 4.2.6.3.2.4 Effect of Receipt

The receipt of this primitive by a UMT Tunnel Adapter will cause the UMT Tunnel Adapter parser function to pass the UMT Client data to the intended UMT Client via the UMT Client Adaptation entity based on the subtype received in the UMTPDU by asserting the UMTCLT.indication primitive according the procedures in 4.2.5.2.2.

### 4.2.7 UMT Tunnel Multiplexer

#### 4.2.7.1 Responsibilities of the UMT Tunnel Multiplexer

The UMT Tunnel Multiplexer multiplexes requests from multiple UMT Tunnel Adapters and passes them to the UMT Encapsulation Sublayer by asserting the UMTPDU.request primitive. Similarly, the UMT Tunnel Multiplexer layer receives UMTPDUs from the UMT Encapsulation Sublayer via the UMTPDU.indication primitive and parses the UMTPDUs for delivery to the UMT Tunnel Adapter designated by the SA and DA fields. Delivery to the UMT Tunnel Adapter entity occurs via assertion of the UMTTM.indication primitive.

#### 4.2.7.2 Block Diagram

Figure 4-4 depicts the major blocks within the UMT Tunnel Multiplexer and their interrelationships with one another and external entities.

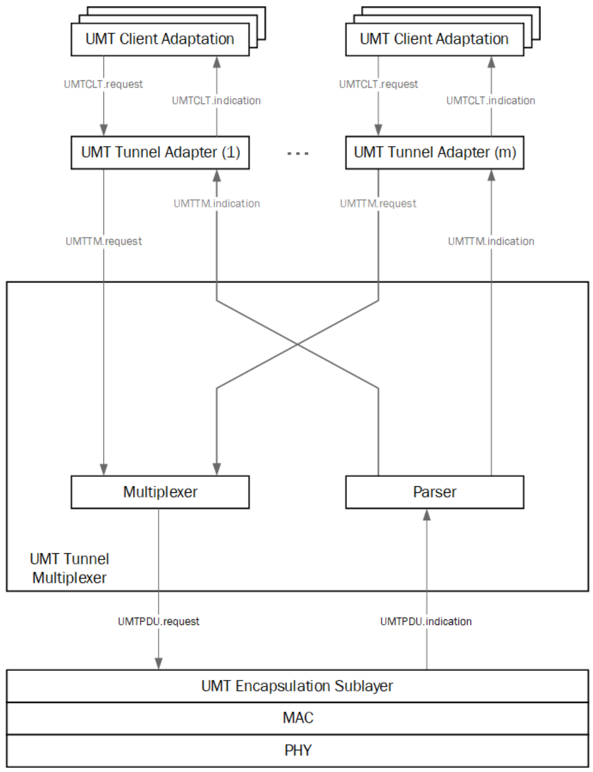



**Figure 4-4 - UMT Tunnel Multiplexer Block Diagram**

#### 4.2.7.3 UMT Tunnel Multiplexer Interactions

The UMT Tunnel Multiplexer entity communicates with the UMT Encapsulation Sublayer using the following interlayer service interfaces:

UMTPDU.request

UMTPDU.indication

The UMTPDU.request and UMTPDU.indication service primitives described in this subclause are mandatory.

##### 4.2.7.3.1 UMTPDU.request

###### 4.2.7.3.1.1 Function

This primitive defines the transfer of data from an UMT Tunnel Multiplexer entity to the UMT Encapsulation Sublayer entity.

##### 4.2.7.3.1.2 Semantics of the service primitive

The semantics of the primitive are as follows:

UMTPDU.request (

destination_address,

source_address,

umt_type,

umt_subtype,

umt_client_sdu

)

The destination_address parameter may specify either an individual or a group MAC entity address and designates the intended UMT destination peer. The source_address parameter, if present, must specify an individual MAC address. If the source_address parameter is omitted, the local MAC sublayer entity will insert a value associated with that entity.

The umt_type corresponds directly to the Length/Type parameter that is defined by IEEE Std. 802.3. The umt_subtype and umt_client_sdu are copied from the UMTTM.request primitive.

##### 4.2.7.3.1.3 When Generated

This primitive is generated by the UMT Tunnel Multiplexer entity whenever an UMTPDU is to be transferred to a peer entity.

##### 4.2.7.3.1.4 Effect of Receipt

The receipt of this primitive will cause the UMT Encapsulation Sublayer entity to insert all UMTPDU specific fields, including DA, SA, Length/Type and Subtype, and pass the properly formed UMTPDU to the lower protocol layers for transfer to the peer UMT entity according to the procedures described in IEEE Std. 802.

#### 4.2.7.3.2 UMTPDU.indication

##### 4.2.7.3.2.1 Function

This primitive defines the transfer of data from an UMT Encapsulation Sublayer entity to a UMT Tunnel Multiplexer entity.

##### 4.2.7.3.2.2 Semantics of the service primitive

The semantics of the primitive are as follows:

UMTPDU.indication (

destination_address,

source_address,

umt_type,

umt_subtype,

umt_client_sdu

)

The destination_address parameter is the MAC destination address of the incoming UMTPDU. The source_address parameter is the MAC source address of the incoming UMTPDU. The umt_type parameter contains the value of the Length/Type field from the received UMTPDU. The umt_subtype and umt_client_sdu parameters are the Subtype and Data fields, respectively, from the incoming UMTPDU.

###### 4.2.7.3.2.3 When Generated

This primitive is passed from the UMT Encapsulation Sublayer entity to the UMT Tunnel Multiplexer entity to indicate the arrival of a UMTPDU to the local UMT Encapsulation Sublayer entity that is destined for a local UMT Client. Such UMTPDUs are reported only if they are validly formed and received without error.

###### 4.2.7.3.2.4 Effect of Receipt

The receipt of this primitive by the UMT Tunnel Multiplexer will cause the UMT Multiplexer parser function to pass the UMT Client data to the intended UMT Tunnel Adapter by asserting the UMTTM.indication primitive according to the procedures in 4.2.6.3.2.

### 4.2.8 UMT Encapsulation Sublayer

#### 4.2.8.1 Responsibilities of the UMT Encapsulation Sublayer

The UMT Encapsulation Sublayer is the intermediate layer that multiplexes requests from the UMT Tunnel Control layer with requests from the MAC Client. The UMT Encapsulation Sublayer passes these requests on to the MAC layer by asserting the MAC:MA_DATA.request primitive. Similarly, the UMT Encapsulation Sublayer receives PDUs from the MAC layer via the MAC:MA_DATA.indication primitive and parses the received PDUs for delivery to the UMT Tunnel Control layer or MAC Client based on the Type/Length field. Delivery to the MAC Client occurs via the MCF:MA_DATA.indication primitive. Delivery to the UMT Tunnel Control layer occurs via the UMTPDU.indication primitive.

#### 4.2.8.2 Block Diagram

Figure 4-5 depicts the major blocks within the UMT Encapsulation Sublayer and their interrelationships with one another and external entities.

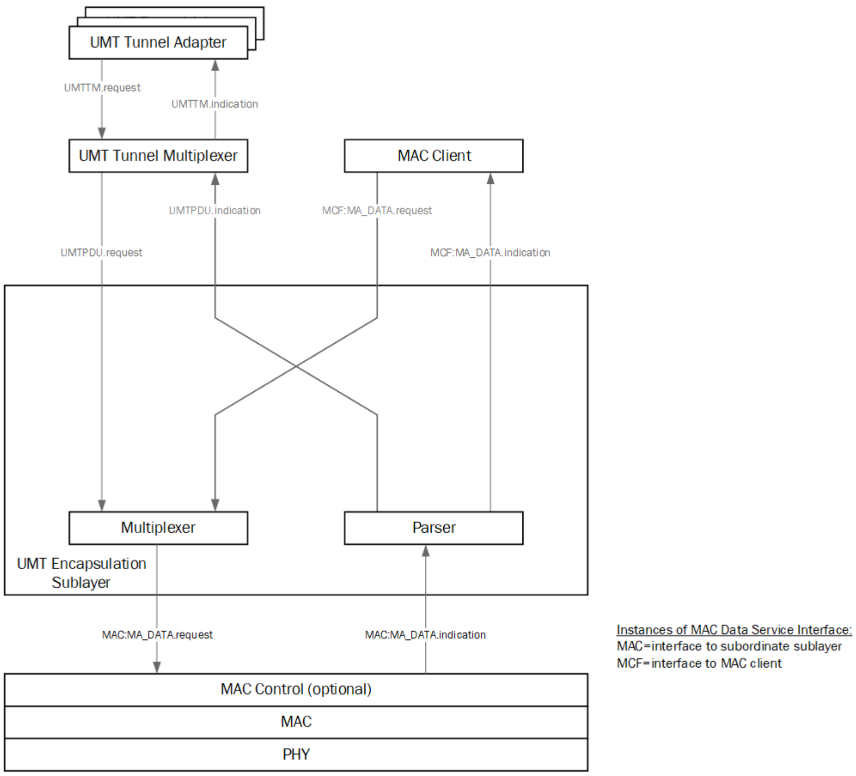



**Figure 4-5 - UMT Encapsulation Sublayer Block Diagram**

#### 4.2.8.3 UMT Encapsulation Sublayer Interactions

The UMT Encapsulation Sublayer entity communicates with the MAC layer using the following interlayer service interfaces:

MAC:MA_DATA.request

MAC:MA_DATA.indication

The UMT Encapsulation Sublayer entity communicates with the MAC Client using the following interlayer service interfaces:

MCF:MA_DATA.request

MCF:MA_DATA.indication

Operation of the MA_DATA.request and MA_DATA.indication primitives is defined in IEEE Std. 802.3 Clause 2. The following sections describe their operation in the context of UMT. Where there is any conflict between this standard and IEEE Std. 802.3 Clause 2, the latter takes precedence.

#### 4.2.8.3.1 MCF:MA_DATA.request

##### 4.2.8.3.1.1.1 Function

See IEEE Std. 802.3 Clause 2.3.1.1

##### 4.2.8.3.1.1.2 Semantics of the service primitive

See IEEE Std. 802.3 Clause 2.3.1.2

##### 4.2.8.3.1.1.3 When generated

See IEEE Std. 802.3 Clause 2.3.1.3

##### 4.2.8.3.1.1.4 Effect of receipt

The receipt of this primitive by the UMT Encapsulation Sublayer will cause the UMT Encapsulation Sublayer to call the MAC sublayer MAC:MA_DATA.request service primitive with its parameters identical to the MCF:MA_DATA.request primitive.

#### 4.2.8.3.2 MCF:MA_DATA.indication

##### 4.2.8.3.2.1.1 Function

See IEEE Std. 802.3 Clause 2.3.2.1

##### 4.2.8.3.2.1.2 Semantics of the service primitive

See IEEE Std. 802.3 Clause 2.3.2.2

##### 4.2.8.3.2.1.3 When generated

This primitive is generated by the UMT Encapsulation Sublayer to indicate to the superior MAC client entity the arrival of a non-UMT PDU. The MCF:MA_DATA.indication primitive is called with its parameters identical to the MAC:MA_DATA.indication primitive.

##### 4.2.8.3.2.1.4 Effect of receipt

See IEEE Std. 802.3 Clause 2.3.2.4

#### 4.2.8.3.3 MAC:MA_DATA.request

##### 4.2.8.3.3.1.1 Function

See IEEE Std. 802.3 Clause 2.3.1.1

##### 4.2.8.3.3.1.2 Semantics of the service primitive

See IEEE Std. 802.3 Clause 2.3.1.2

##### 4.2.8.3.3.1.3 When generated

This primitive is generated by the UMT Encapsulation Sublayer when the superior MAC client asserts the MCF:MA_DATA.request primitive. The MAC:MA_DATA.request primitive is called with its parameters identical to the MCF:MA_DATA.request primitive.

The MAC:MA_DATA.request primitive is also called when a UMTPDU.request primitive is received from the UMT Tunnel Multiplexer layer. In this case, the UMT Encapsulation Sublayer copies the destination_address and source_address into the destination_address and source_address field of the MAC:MA_DATA.request primitive. Further, the UMT Encapsulation Sublayer assembles the mac_service_data_unit field by concatenating the umt_type, umt_subtype, and umt_client_sdu parameters received in the UMTPDU.request primitive.

###### 4.2.8.3.3.1.4 Effect of receipt

See IEEE Std. 802.3 Clause 2.3.1.4

#### 4.2.8.3.4 MAC:MA_DATA.indication

###### 4.2.8.3.4.1.1 Function

See IEEE Std. 802.3 Clause 2.3.2.1

###### 4.2.8.3.4.1.2 Semantics of the service primitive

See IEEE Std. 802.3 Clause 2.3.2.2

###### 4.2.8.3.4.1.3 When generated

See IEEE Std. 802.3 Clause 2.3.2.3

###### 4.2.8.3.4.1.4 Effect of receipt

The receipt of this primitive by the UMT Encapsulation Sublayer will cause the UMT Encapsulation Sublayer to parse the incoming frame. Based on the value of the Length/Type field, the UMT Encapsulation Sublayer will determine whether the frame is destined for the UMT Tunnel Multiplexer or the MAC Client.

If the frame is destined for the MAC Client, the UMT Encapsulation Sublayer will generate an MCF:MA_DATA.indication service primitive with its parameters identical to the MAC:MA_DATA.indication primitive.

If the frame is destined for the UMT Tunnel Multiplexer, the UMT Encapsulation Sublayer parses the UMTPDU to find the Type, Subtype, and Data fields. After parsing the UMTPDU, the UMT Encapsulation Sublayer asserts the UMTPDU.indication primitive with the umt_type parameter copied from the Type field, the umt_subtype parameter copied from the Subtype field, and the umt_client_sdu parameter copied from the Data field.

## 4.3 Detailed functions and state diagrams

### 4.3.1 State Diagram Variables

#### 4.3.1.1 Constants

UMT_Subtype

**The value of the Subtype field for UMTPDUs (see**

Table 4-2).

UMT_Protocol_Type

The value of the UMT Protocol Length/Type field. (see Table 4-1).

NULL

The value used to indicate the empty set or the non-existence of an entity.

#### 4.3.1.2 Variables

BEGIN

A variable that resets the functions within UMT.

Values: TRUE; when any of the component UMT Encapsulation Sublayers is reset.

FALSE; When (re-)initialization has completed.

ind_DA

ind_SA

ind_mac_service_data_unit

ind_reception_status

The parameters of the MA_DATA.indication service primitive, as defined in IEEE Std. 802.3 Clause 2.

ind_umt_tid

The value of the umt_tunnel_id parameter passed to the UMT Tunnel Adapter in the UMTTM.indication primitive.

Value: Integer

ind_Length/Type

The value of the Length/Type field in a received MAC protocol frame (see Table 4-1) and is passed to the UMT Tunnel Multiplexer in the umt_type parameter of the UMTPDU.indication primitive.

Value: Integer

ind_umt_subtype

**The value of the Subtype field in a received UMT protocol frame (see**

Table 4-2) and is passed to the UMT Tunnel Multiplexer in the umt_subtype parameter of the UMTPDU.indication primitive.

Value: Integer

ind_umt_client_sdu
The value of the Data field in a received UMT protocol frame and is passed to the UMT Tunnel Multiplexer in the umt_client_sdu paremeter of the UMTPDU.indication primitive.

req_DA

req_SA

req_mac_service_data_unit

req_reception_status

The parameters of the MA_DATA.request service primitive, as defined in IEEE Std. 802.3 Clause 2.

req_umt_tid

The value of the umt_tunnel_id parameter passed to the UMT Tunnel Multiplexer in the UMTTM.request primitive.

Value: Integer

req_umt_type

The value of the umt_type parameter passed to the UMT Encapsulation Sublayer in the UMTPDU.request primitive.

Value: Integer

req_umt_subtype

The value of the umt_subtype parameter passed to the UMT Client in the UMTCLT.request primitive.

Value: Integer

req_umt_client_sdu

The value of the umt_client_sdu parameter passed to the UMT Client in the UMTCLT.request primitive.

#### 4.3.1.3 Messages

MAC:MA_DATA.indication

MCF:MA_DATA.indication

The service primitives used to pass a received frame to a client with the specified parameters.

MAC:MA_DATA.request

MCF:MA_DATA.request

The service primitives used to transmit a frame with the specified parameters.

UMTPDU.indication

UMTCLT.indication

UMTTM.indication

UMTCAL.indication

The service primitives used to pass a received UMTPDU to a client with the specified parameters.

UMTPDU.request

UMTCLT.request

UMTTM.request

UMTCAL.request

The service primitives used to transmit a UMTPDU with the specified parameters.

UMTPDUIND

Alias for UMTPDU.indication (ind_DA, ind_SA, ind_Length/Type, ind_umt_subtype, ind_umt_client_sdu)

UMTPDUREQ

Alias for UMTPDU.request(req_DA, req_SA, req_umt_type, req_umt_subtype, req_umt_client_sdu)

UMTCLTIND

Alias for UMTCLT.indication (ind_DA, ind_SA, ind_umt_subtype, ind_umt_client_sdu)

UMTCLTREQ

Alias for UMTCLT.request(req_umt_subtype, req_umt_client_sdu)

UMTTMREQ

Alias for UMTTM.request(req_umt_tid, req_umt_subtype, req_umt_client_sdu)

UMTTMIND

Alias for UMTTM.indication(ind_umt_tid, ind_DA, ind_SA, ind_umt_subtype, ind_umt_client_sdu)

MADR

Alias for MA_DATA.request(req_DA, req_SA, req_mac_service_data_unit, frame_check_sequence)

MADI

Alias for MA_DATA.indication(ind_DA, ind_SA, ind_mac_service_data_unit, ind_reception_status)

#### 4.3.1.4 Functions

get_sa(req_umt_tid)

This function returns the desired source MAC address to be used on the tunnel indicated by req_umt_tid. This function returns NULL if the source MAC address is to be inserted by the MAC layer. The implementation of the get_sa() function is out of scope for this standard.

get_da(req_umt_tid)

This function returns the desired destination MAC address to be used on the tunnel indicated by ind_umt_tid). It is assumed that it is not possible to call this function prior to the specified tunnel's creation and therefore it must always return a valid value. The implementation of the get_da() function is out of scope for this standard.

get_tid(ind_SA, ind_DA)

This function returns the unique identifier of the UMT Tunnel Adapter associated with the indicated source MAC address and indicated destination MAC address. This function returns NULL if there is no UMT Tunnel Adapter configured with the specified source MAC address and destination MAC. The implementation of the get_tid() function is out of scope for this standard.

length(binary_data)

This function returns the length, in bits, of the binary_data parameter.

##### 4.3.1.5 Counters

No counters are defined.

##### 4.3.1.6 Timers

No timers are defined.

### 4.3.2 UMT Client Adaptation

Refer to the annex for informative and normative descriptions of UMT Client Adaptations for specific UMT Client protocols.

### 4.3.3 UMT Tunnel Adapter

As depicted in Figure 4-3, the UMT Tunnel Adapter is comprised of the functions:

a) ***Multiplexer***. This function is responsible for multiplexing UMT Client service data units received from the UMT Client and UMT Client Adaptation entities and passing them to the UMT Tunnel Multiplexer.

b) ***Parser***. This function distinguishes among UMTPDU subtypes and passes received UMT Client service data units to the appropriate UMT ser via the associated UMT Client Adaptation entity.

#### 4.3.3.1 Multiplexer

The UMT Tunnel Adapter shall implement the multiplexer state diagram shown in Figure 4-6.

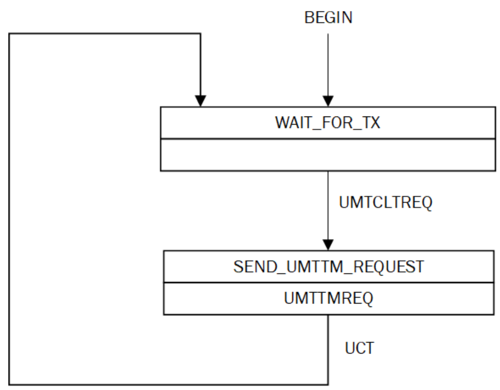



**Figure 4-6 - UMT Tunnel Adapter Multiplexer State Diagram**

#### 4.3.3.1.1 WAIT_FOR_TX State

Upon initialization, the WAIT_FOR_TX state is entered. While in the WAIT_FOR_TX state, the Multiplexer waits for the occurrence of an UMTCLT.request. The UMTCLT.request signal can be asserted by one or more UMT Client Adaptation entities.

#### 4.3.3.1.2 SEND_UMTTM_REQUEST State

**Once the Multiplexer reaches the SEND_UMTTM_REQUEST state, it shall assert the UMTTM.request signal with the required parameters. The value of req_umt_subtype shall be set by the UMT Client Adaptation entity based on the identity of the UMT Client that asserted the UMTCAL.request. The value must be taken from**

Table 4-2. The value of req_umt_tid shall be set by the UMT Tunnel Adapter based on the tunnel identifier assigned to it at the time of its creation.

### 4.3.3.2 Parser

The UMT Tunnel Adapter shall implement the parser state diagram shown in Figure 4-7.

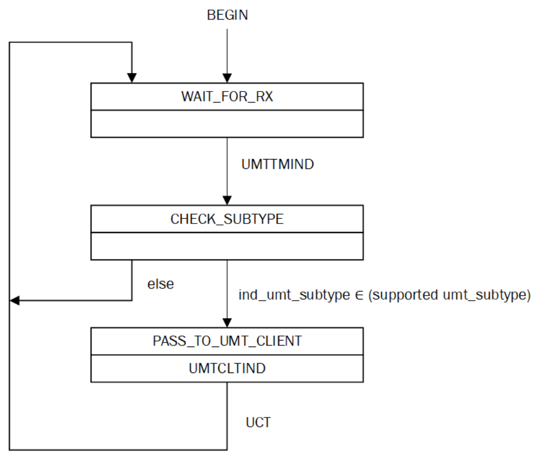



**Figure 4-7 - UMT Tunnel Adapter Parser State Diagram**

##### 4.3.3.2.1 WAIT_FOR_RX State

Upon initialization, the WAIT_FOR_RX state is entered. While in the WAIT_FOR_RX state, the parser waits for the occurrence of an UMTTM.indication. Upon assertion of UMTTM.indication the parser enters the CHECK_SUBTYPE state.

##### 4.3.3.2.2 CHECK_SUBTYPE State

In the CHECK_SUBTYPE state, the parser inspects the value of ind_umt_subtype. If the value of ind_umt_subtype is an element of the supported UMT subtypes, the parser will transition to the PASS_TO_UMT_CLIENT state. If the value of ind_umt_subtype is not a supported UMT subtype, the parser will discard the UMTPDU and move to the WAIT_FOR_RX state.

**A value of ind_umt_subtype is an element of the supported umt_subtypes if a UMT Client Adaptation entity has registered itself to use the associated tunnel with one of the UMT Subtypes found in**

Table 4-2.

##### 4.3.3.2.3 PASS_TO_UMT_CLIENT State

In the PASS_TO_UMT_CLIENT state, the parser asserts the UMTCLT.indication signal. The destination UMT Client Adaptation entity is determined by the value of ind_umt_subtype.

### 4.3.4 UMT Tunnel Multiplexer

As depicted in Figure 4-4, the UMT Tunnel Multiplexer is comprised of the following functions:

c) ***Multiplexer***. This function is responsible for multiplexing UMT Client service data units received from the UMT Tunnel Adapters and passing them to the UMT Encapsulation Sublayer.

d) ***Parser***. This function distinguishes among UMT tunnels passes received UMT Client service data units to the appropriate UMT Tunnel Adapter.

#### 4.3.4.1 Multiplexer

The UMT Tunnel Multiplexer shall implement the multiplexer state diagram shown in Figure 4-8.

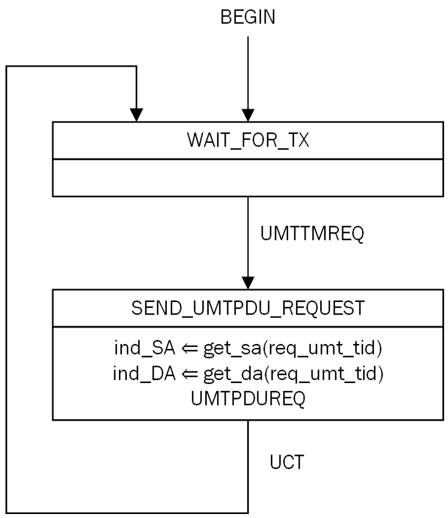



**Figure 4-8 - UMT Tunnel Multiplexer Multiplexer State Diagram**

##### 4.3.4.1.1 WAIT_FOR_TX State

Upon initialization, the WAIT_FOR_TX state is entered. While in the WAIT_FOR_TX state, the Multiplexer waits for the occurrence of an UMTTM.request. The UMTTM.request signal can be asserted by one or more UMT Tunnel Adapter entities.

##### 4.3.4.1.2 SEND_UMTPDU_REQUEST State

Once the Multiplexer reaches the SEND_UMTPDU_REQUEST state, it shall assert the UMTPDU.request signal with the required parameters. The value of req_SA req_DA are determined by calling the get_da() and get_sa() functions. The value of req_umt_subtype and req_umt_client_sdu shall be copied from the received UMTTM.request primitive parameters.

#### 4.3.4.2 Parser

The UMT Tunnel Multiplexer shall implement the parser state diagram shown in Figure 4-9.

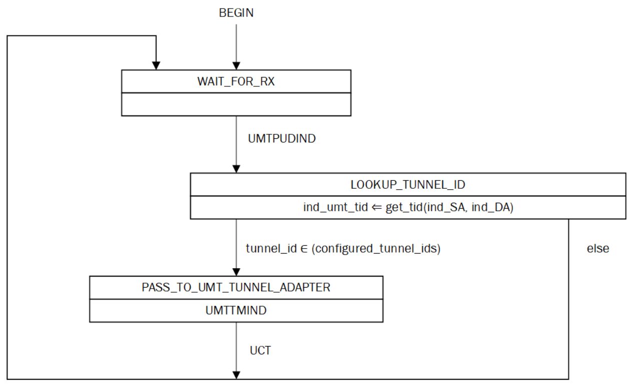



**Figure 4-9 - UMT Tunnel Multiplexer Parser State Diagram**

#### 4.3.4.2.1 WAIT_FOR_RX State

Upon initialization, the WAIT_FOR_RX state is entered. While in the WAIT_FOR_RX state, the parser waits for the occurrence of an UMTPDU.indication. Upon assertion of UMTPDU.indication the parser enters the LOOKUP_TUNNEL_ID state.

#### 4.3.4.2.2 LOOKUP_TUNNEL_ID State

In the LOOKUP_TUNNEL_ID state, the parser determines the local instance of UMT Tunnel Adapter entity to which the UMTPDU is destined by calling the get_tid() function. The parser will transition to the PASS_TO_UMT_TUNNEL_ADAPTER state if the identified tunnel is an element of the configured tunnels on the local UMT peer. If the identified tunnel is not configured on the local UMT peer, the parser will discard the UMTPDU and move to the WAIT_FOR_RX state.

A tunnel is an element of the configured tunnels if an adminsistrator has configured the tunnel on the local UMT peer. A tunnel may been configured by the administrator manually or through an automated UMT peer discovery mechanism.

#### 4.3.4.2.3 PASS_TO_UMT_TUNNEL_ADAPTER State

In the PASS_TO_UMT_TUNNEL_ADAPTER state, the parser asserts the UMTTM.indication primitive. The destination UMT Tunnel Adapter entity is determined by the ind_umt_tid value returned in the LOOKUP_TUNNEL_ID state.

### 4.3.5 UMT Encapsulation Sublayer

As depicted in Figure 4-5, the UMT Encapsulation Sublayer comprises the following functions:

e) ***Multiplexer***. This function is responsible for passing frames received from the superior sublayer (i.e., UMT client) and UMTPDUs to the subordinate sublayer (e.g., MAC sublayer).

f) ***Parser***. This function distinguishes among UMTPDUs and MAC client frames and passes each to the appropriate entity (UMT client or superior sublayer, respectively).

#### 4.3.5.1 Multiplexer

The UMT Encapsulation Sublayer entity shall implement the multiplexer state diagram shown in Figure 4-10.

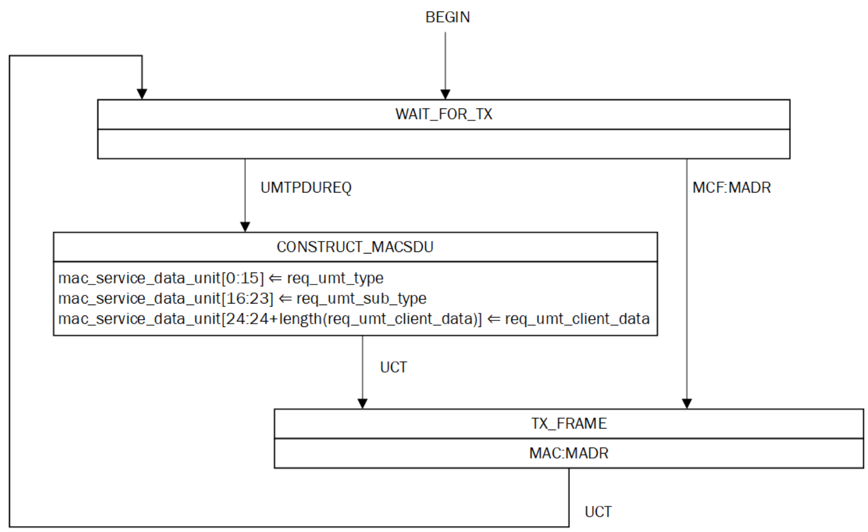



**Figure 4-10 - UMT Encapsulation Sublayer Multiplexer State Diagram**

##### 4.3.5.1.1 WAIT_FOR_TX state

Upon initialization, the WAIT_FOR_TX state is entered. While in the WAIT_FOR_TX state, the Multiplexer waits for the occurrence of a UMTPDU.request or MCF:MA_DATA.request.

##### 4.3.5.1.2 CONSTRUCT_MACSDU state

The multiplexer transitions to the CONSTRUCT_MACSDU state when a UMTPDU.request is received. In the CONSTRUCT_MACSDU state the multiplexer populates the Type/Length field with req_umt_type, the UMT subtype field with req_umt_subtype, and the remainder of the MAC Data field with req_umt_client_sdu.

##### 4.3.5.1.3 TX_FRAME state

Once the multiplexer reaches the TX_FRAME state, it shall provide transparent pass-through of frames submitted by the superior sublayer. The transmission of a UMTPDU shall not affect the transmission of a frame that has been submitted to the subordinate sublayer (i.e., the MAC's TransmitFrame function is synchronous, and is never interrupted). After the frame has been sent to the subordinate sublayer, the Multiplexer process returns to the WAIT_FOR_TX state.

#### 4.3.5.2 Parser

The UMT Encapsulation Sublayer entity shall implement the parser state diagram shown in Figure 4-11.

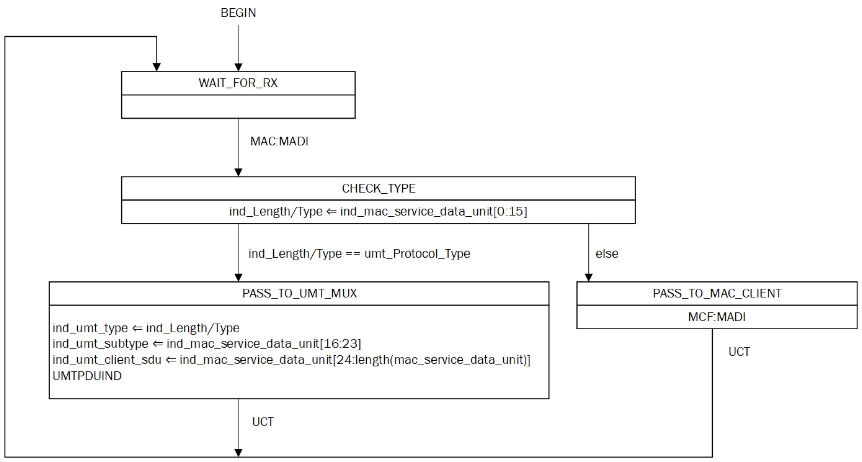



**Figure 4-11 - UMT Encapsulation Sublayer Parser State Diagram**

#### 4.3.5.2.1 WAIT_FOR_RX state

Upon initialization, the WAIT_FOR_RX state is entered. While in the WAIT_FOR_RX state, the parser waits for the occurrence of an MAC:MA_DATA.indication. Upon assertion of MAC:MA_DATA.indication the parser enters the CHECK_TYPE state.

#### 4.3.5.2.2 CHECK_TYPE state

In the CHECK_TYPE state, the parser inspects the value of the ind_Length/Type field. If the value of the ind_Length/Type equals umt_Protocol_Type, the parser will transition to the PASS_TO_UMT_MUX state. If the value of the ind_Length/Type is anything else, the parser will move to the PASS_TO_MAC_CLIENT state.

#### 4.3.5.2.3 PASS_TO_UMT_MUX

In the PASS_TO_UMT_MUX state, the parser parses the UMTPDU to find the ind_umt_subtype, and ind_umt_client_sdu and then asserts the UMTPDU.indication primitive.

#### 4.3.5.2.4 PASS_TO_MAC_CLIENT state

In the PASS_TO_MAC_CLIENT state, the parser asserts the MCF:MA_DATA.indication primitive with parameters identical to those received from the MAC:MA_DATA.indication primitive.

## 4.4 UMT PDU format

### 4.4.1 Ordering and representation of octets

All UMTPDUs comprise an integral number of octets. When the encoding of (an element of) an UMTPDU is depicted in a diagram:

a) Octets are transmitted from top to bottom within the given field.

b) Within an octet, bits are shown with bit 0 to the left and bit 7 to the right.

c) When consecutive octets are used to represent a binary number, the octet transmitted first has the more significant value.

d) When consecutive octets are used to represent a MAC address, the least significant bit of the first octet is assigned the value of the first bit of the MAC address, the next most significant bit the value of the second bit of the MAC address, and so on for all the octets of the MAC address.

When the encoding of an element of an UMTPDU is depicted in a table, the least significant bit is bit 0. The bit/octet ordering of any Organizationally Unique Identifier (OUI) or Company ID (CID) field within an UMTPDU is identical to the bit/octet ordering of the OUI portion of the Destination Address (DA)/Source Address(SA). Additional detail defining the format of OUIs and CIDs can be found in IEEE Std 802-2014, 8.2.2.

### 4.4.2 Structure

The UMTPDU structure shall be as shown in Figure 4-12.

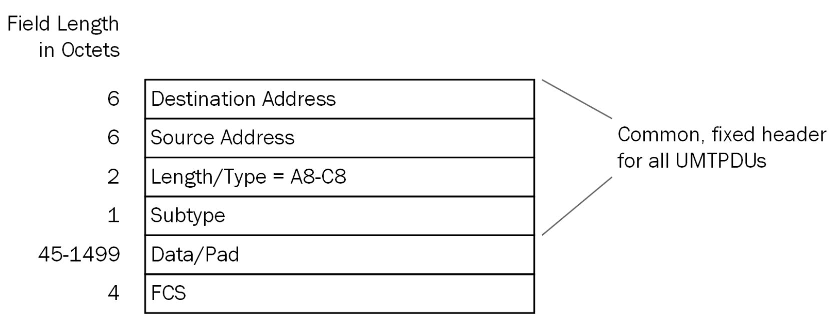



**Figure 4-12 - UMTPDU Frame Structure**

UMTPDUs shall have the following fields

a) ***Destination Address*** (DA). The DA in the UMTPDU specifies the destination addressee(s) for which the frame is intended. Its use and encoding are specified in IEEE Std 802.3, Clause 4.

b) ***Source Address*** (SA). The SA in UMTPDUs carries the individual MAC address associated with the port through which the UMTPDU is transmitted.

c) ***Length/Type***. The Length/Type in UMTPDUs carries the UMT_Protocol_Type field value as specified in Table 4-1.

***Subtype*. The Subtype field identifies the specific UMT Client layer being encapsulated. The Subtype field value is specified in**

d) Table 4-2.

e) ***Data***. This field contains the UMTPDU data. This field must be at least 45 octets in length to ensure that no UMTPDU is less than 64 octets in length.

**Commented [KAN1]:** This implies that UMT might need to pad the data field. How do we deal with this?

f) ***FCS***. This field is the Frame Check Sequence, as defined in IEEE Std. 802.3.

**Table 4-1 - UMT_Protocol_Type Value**

| Name | Value |
|---|---|
| UMT_Protocol_Type | A8-C8 |

**Table 4-2 - UMT Subtype Values**

| Protocol Subtype Value | Protocol Name |
|---|---|
| 0 | Reserved |
| 1 | Unassigned |
| 2 | Unassigned |
| 3 | IEEE Std. 802.3 and IEEE Std. 1904.1 Operations, Administration, and Maintenance (OAM) |
| 4-10 | Unassigned |
| 11 | IGMP |
| 12 | OMCI |
| 13 | UMT Relay |
| 14-252 | Unassigned |
| 253 | Vendor-Specific |
| 254 | UMT Peer Maintenance |
| 255 | Reserved |

### 4.4.3 UMTPDU Description

The local UMT layer communicates with the remote UMT layer via UMTPDUs. UMTPDUs are identified with a specific code. UMTPDUs are formatted as compliant IEEE 802.3 frames, where the IEEE 802.3 frame header format is described in IEEE Std. 802.3. UMTPDUs are further defined, as shown in Figure 4-12, to include a Subtype field following the IEEE 802.3 defined Length/Type field. The Data field begins in a fixed location within the UMTPDU. The Data field contents are unique to the particular UMTPDU. All received UMTPDUs are parsed by the UMT layer to determine to which superior layer the Payload is to be delivered. The UMT Subtype field shall be used to determine which superior layer will receive the Payload. UMTPDUs with reserved Subtype field values are not transmitted. A UMTPDU containing a reserved Subtype value is ignored on receipt. A UMTPDU containing a Subtype value that is unsupported by the receiving UMT layer are ignored on receipt.

#### 4.4.4 UMTPDU Addressing

A UMT tunnel is uniquely identified by the combination of the MAC Source Address and MAC Destination Address in the UMTPDU SA and DA fields. The SA shall be the MAC address of the local UMT peer and must not be a broadcast or group MAC address.

In typical operation the DA of the UMTPDU will be the unique MAC address of a UMT peer. This is referred to as unicast UMT operation.

Nothing in this standard disallows the use of a broadcast or group MAC address in the DA field of the UMTPDU. UMT broadcast mode operation refers to the case when a broadcast MAC address is used in the DA field of the UMTPDU. UMT multicast mode operation refers to the case when a group MAC address is used in the DA field of the UMTPDU.

When a UMT peer receives a UMTPDU with a broadcast or group MAC address in the DA field, the UMT Encapsulation Sublayer shall pass the UMTPDU to the UMT Tunnel Multiplexer. The UMT Tunnel Multiplexer shall lookup the tunnel id as specified in 4.3.4.2. If no tunnel id is found to match the unique (SA,DA) pair, then the UMT Tunnel Multiplexer shall drop the UMTPDU, otherwise the UMT Multipexer shall pass the UMTPDU to the corresponding UMT Tunnel Adapter. This implies that an administrator may configure a tunnel with a broadcast or group destination address, and must configure such a tunnel if broadcast or multicast UMT operation is desired.

## 4.5 Protocol implementation conformance statement (PICS) proforma

### 4.5.1 Introduction

The supplier of a protocol implementation that is claimed to conform to this standard shall complete the following protocol implementation conformance statement (PICS) proforma.

### 4.5.2 Identification

#### 4.5.2.1 Implementation identification

| | |
|---|---|
| Supplier | |
| Contact Point for inquiries about the PICS | |
| Implementation Name(s) and Version(s) | |
| Other information necessary for full identification—e.g., name(s) and version(s) for machines and/or operating systems; System Name(s) | |
| NOTE 1—Only the first three items are required for all implementations; other information may be completed as appropriate in meeting the requirements for the identification.<br><br>NOTE 2—The terms Name and Version should be interpreted appropriately to correspond with a supplier's terminology (e.g., Type, Series, Model). | |

#### 4.5.2.2 Protocol Summary

| | |
|---|---|
| Identification of protocol standard | IEEE Std. 1904.2, Univeral Management Tunnel |

| | |
|---|---|
| Identification of amendments and corrigenda to this PICS proforma that have been completed as part of this PICS | |
| Have any Exception items been required? No [ ] Yes [ ]<br><br>(The answer Yes means that the implementation does not conform to IEEE Std 1904.2.) | |
| Date of Statement | |

#### 4.5.2.3 Major Capabilities/Options

| Item | Feature | Subclause/Table | Value/Comment | Status | Support |
|---|---|---|---|---|---|
| | | | | O/M | Yes [ ]<br><br>No [ ] |
| | | | | | |

### 4.5.3 PICS proforma tables for UMT

#### 4.5.3.1 Functional Specifications

| Item | Feature | Subclause/Table | Value/Comment | Status | Support |
|---|---|---|---|---|---|
| | | | | O/M | Yes [ ]<br><br>No [ ] |
| | | | | | |

#### 4.5.3.2 UMTPDUs

| Item | Feature | Subclause/Table | Value/Comment | Status | Support |
|---|---|---|---|---|---|
| | | | | O/M | Yes [ ]<br><br>No [ ] |
| | | | | | |

## ~~4.6 UMT Architecture~~

~~A typical PON is deployed with an OLT at the local Central Office (CO) and several ONUs which are connected to the Outside Distribution Network (ODN) comprising at least one fiber spliter. The OLT acts as the management master responsible for controlling individual connected ONUs, including MPCP / OAM registration, service provisioning, etc., as defined in IEEE Std 1904.1-2013.~~

### ~~4.2.1.Single hop between Management Master and OLT~~

~~In this scenario, the UMT Management Master is collocated with the OLT within the CO, and it is has access to all information within the OLT, such as status of individual ONUs, QoS profiles assigned to individual services, device status, etc.. Physically, the UMT Management Master in this architecture would have a form of a software agent running on the OLT hardware. This architecture example is shown in Figure 4-13.~~

**~~Figure 4-13 – Single hop between Management Master and OLT~~**

### ~~4.2.2 Multiple hops between Management Master and OLT~~

~~In that example, the UMT Mangment Master does not have a direct access to the OLT, but it shares the same L2 network, providing access to information stored within the OLT via standardized interfaces. The UMT Management Master and the OLT are separated by a number of layer 2 hops. Physically, the UMT Management Master in this architecture would have the form of a software agent running on either a dedicated or virtual machine, physically separate from the OLT, but otherwise connected to the same LAN. The UMT Management Master in this case can be shared by more that one OLT, provided that all these OLTs are connected to the same LAN. This arrangement is shown in Figure 4-14.~~

**~~Figure 4-14 – Multiple hops between Management Master and OLT~~**

### ~~4.2.3 Management Master sharing L3 network with EPON OLT~~

~~In that example, the UMT Mangment Master is connected (directly on indirectly) to the core transport network of the operator and manages a number of OLTs connected (directly or indirectly) to the same core transport network. The UMT Management Master is provided access to information stored within the OLT via standardized interfaces. Physically, the UMT Management Master in this architecture would have the form of a software agent running on either a dedicated or virtual machine, physically separate from the OLT, but otherwise reachable via IP level connectivity. The UMT Management Master in this case can be shared by more that one OLT, provided that all these OLTs are connected at the IP level. This arrangement is shown in Figure 4-15.~~

**~~Figure 4-15 – Management Master sharing L3 network with EPON OLT~~**

## ~~4.7 UMT Interfaces~~

### ~~4.7.1 UMT Layering~~

~~Figure 4-16- UMT Layering diagram~~

### ~~4.7.2 4.2 Frame transformation architecture~~

~~Figure 4-17- Frame Transformation layers architecture~~

### ~~4.7.3 States Diagram~~

~~Figure 4-18- Parser state diagram~~

~~Figure 4-19 - UMT Multiplexer state diagram~~

## ~~4.8 UMT Device Functions~~

## ~~4.9 Examples of UMT Use Cases~~

# ~~5 UMT Discovery Protocol (UMTDP)~~

## ~~5.1 Definition of UMTDP Data Unit~~

## ~~5.2 UMTDP Operation~~

## ~~5.3 State diagrams and variable definitions~~

### ~~5.3.1 Variables~~

### ~~5.3.2 Times~~

### ~~5.3.3 Functions~~

### ~~5.3.4 Primitives~~

### ~~5.3.5 State diagrams~~

# ~~6 PICS~~

# ~~7 Examples: Header 1~~

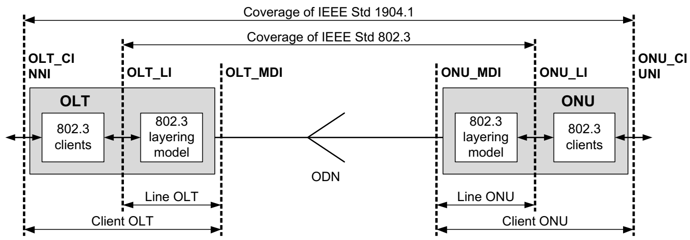
## ~~7.1 Examples: Header 2~~

~~Example of a paragraph of text.~~

~~Example of a table is shown below.~~

**~~Table 7-1—Table Template~~**

| ~~Column1~~ | ~~Column2~~ | ~~Column3~~ |
|---|---|---|
| ~~Value1~~ | ~~Value2~~ | ~~Value3~~ |
| ~~Value1~~ | ~~Value2~~ | ~~Value3~~ |
| ~~Value1~~ | ~~Value2~~ | ~~Value3~~ |




b) OLT and ONU without service-specific functions

**~~Figure 7-1—Example of a figure~~**

~~Example of a bulleted list:~~

- ~~Line 1; and~~
- ~~Line 2.~~

### ~~7.1.1 Examples: Header 3~~

#### ~~7.1.1.1 Examples: Header 4~~

##### ~~7.1.1.1.1 Examples: Header 5~~

has two functions in the UMT stack. The first is to manage the instantiation of Tunnel Adapters which may be configured by the administrator or automatically discovered. The second is to is the intermediate layer that controls the instantiation of emulates a point-to-point link between the local UMT Peer and each remote UMT Peer. UMT Peers may be configured by the administrator or automatically discovered. The UMT Tunnel Control layer presents a unique Tunnel Adapter entity to the UMT Clients for each remote UMT Peer.