13 Extended OAM for Nx25G-EPON

13.1 Introduction

The EPON system supports all OAM functions as specified in IEEE Std 802.3, Clause 57, and an eOAM-based management suite providing the following functions and features:

- eOAM device discovery and device capability discovery
- Querying and setting DBA-related parameters, including the format of *REPORT* MPCPDU and associated queue structures
- Querying and setting QoS-related parameters, including flow marking, cataloguing, classification rules, etc.
- Configuration of service ports and their associated management
- Configuration and management of VLANs
- Configuration of multicast flows and associated management parameters
- Performing management actions on the ONU devices and subsystems, including port status control, ONU device status verification, etc.
- Software upgrade process that allows operators to remotely modify the software loaded on the ONU.

13.2 Requirements

13.2.1 Frame size requirements

The eOAMPDU supports the maximum frame size with a payload of 1500 octets. The size of OAMPDUs is specified in IEEE Std 802.3, 57A.2.

13.2.2 Frame rate requirements

The maximum combined frame rate for the IEEE 802.3 (Clause 57) OAMPDUs and eOAMPDUs per ONU is configurable, as specified in IEEE Std 802.3, Clause 57 and 57A.2. The default frame rate of 10 frames/sec may be changed using the *aLlidOamFrameRate* (0xDB/0x00-0D) attribute (see 14.4.1.11).

13.2.3 Timing requirements

Some eOAMPDUs require a response from the ONU. The ONU shall generate an *eOAM_Get_Response* or *eOAM_Set_Response* eOAMPDU within 1 second of the reception of the corresponding *eOAM_Get_Request* or *eOAM_Set_Request* eOAMPDU from the OLT. This requirement covers all types of management requests issued by the OLT for the specific ONU: reading data from specific variable(s), setting values to specific variable(s), performing indicated action(s). The OLT may discard all ONU responses received after the expiration of the 1-second window without processing.

If an ONU cannot respond to the OLT request before the expiration of 1-second window, the ONU shall generate the ONU Busy alarm (see 13.4.4.2.6). The reception of the ONU Busy alarm at the OLT represents a non-critical abnormality. The handling of this condition is implementation specific. The raise of the ONU Busy alarm is generally not a reason to deregister that ONU.

13.2.4 Logical link requirements

An ONU shall transmit all OAMPDUs in envelopes tagged with Management Link ID (MLID) that was assigned to the ONU during the MPCP registration (i.e., the primary MLID).

The ONU shall be able to receive OAMPDUs in evelopes tagged with either the primary MLID or any additional unidirectional MLID assigned to the ONU via the *acConfigLlid* (0xDD/0x01-20) management action (see 14.6.2.8).

13.2.5 Virtual Link Control support

An ONU may optionally support IEEE Std 1904.2 Virtual Link Control (VLC), which allows OAM management messages to be carried over VLC tunnels. An IEEE Std 1904.2 compliant ONU may receive OAM messages as native OAMPDUs and eOAMPDUs or as VLCPDUs with subtype `SUBTYPE_OAM`. If the ONU receives an OAM request as an OAMPDU, it shall reply with an OAMPDU OAM response message. If the ONU receives an OAM request as an VLCPDU, it shall generate OAM reponse as VLCPDU as well. In the latter case, the VLCPDU destination address shall be equal to the source address in the received VLCPDU OAM request.

To initiate the OAM discovery, the IEEE Std 1904.2-compliant ONU waits for the Information TLV from the OLT first. If that TLV is received in an OAMPDU, the ONU transmits its own Information TLVs (including the Extended Information TLVs) also using an OAMPDU. Otherwise, if the Information TLV is received from the OLT in a VLCPDU, the ONU transmits its Information TLVs in an VLCPDUs as well.

The ONU shall send the OAM event notification messages using the same PDU type as was used by the last OAM message it received from the OLT.

The ONU shall transmit all VLCPDUs in envelopes with MLID tag.

13.3 Device discovery and capability discovery

13.3.1 MPCP/OAM discovery process

Figure 13-1 shows the relationship between the process of registration, initialization, and negotiation in EPON prior to establishing the data plane connectivity. First, the MPCP discovery and registration process is executed, as defined in IEEE Std 802.3ca, 144.3.7. Next, the process of OAM discovery, as defined in IEEE Std 802.3, Clause 57, and eOAM discovery, as defined in the following subclauses, is executed.

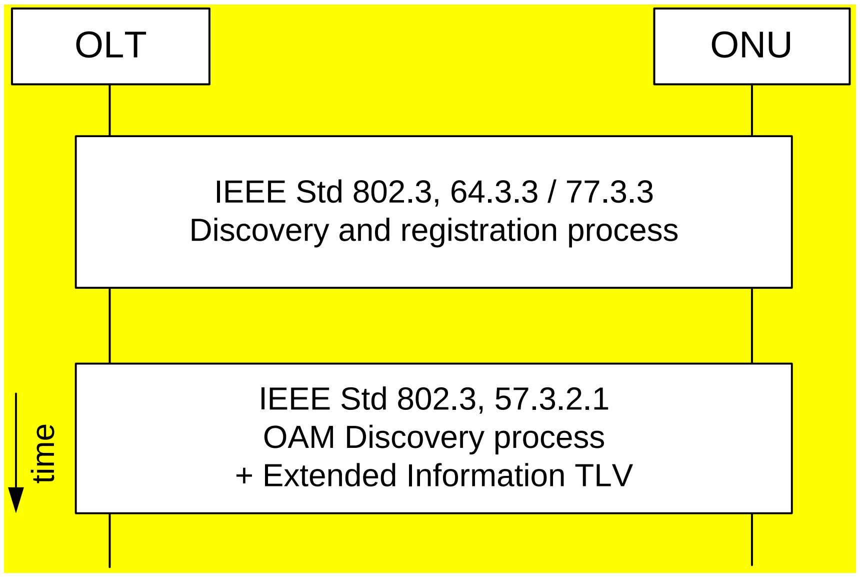
OLT
ONU
IEEE Std 802.3, 64.3.3 / 77.3.3
Discovery and registration process
IEEE Std 802.3, 57.3.2.1
OAM Discovery process
+ Extended Information TLV
time

Figure 13-1—MPCP/OAM discovery process

13.3.2 eOAM discovery process

The eOAM discovery process in the EPON is used to identify whether the given connected ONU supports the specific subtype of the Organization Specific OAM extensions (as identified by the OUI) and further to identify the capabilities of such an ONU device in terms of the supported OAM functions.

Editorial Note (to be removed prior to publication): The following statement may need to be revised after the PON protection spec is finalized and accepted

The eOAM discovery proces is executed once per physical ONU device.

13.3.2.1 Requirements

The EPON system shall implement the eOAM discovery process and the eOAM Capability Notification mechanism, using the Organization Specific extensions to the *Information* TLV specified in IEEE Std 802.3, 57.5.2.3.

The OLT shall enable any data services for the given ONU only upon the successful completion of the eOAM discovery process, as defined in 13.3, and after the completion of the authentication process if enabled by the operator.

The OLT shall deregister any ONU that does not complete the eOAM discovery process, as defined in 13.3, within five seconds of the time when the OLT sends the first *Extended Information* TLV to this specific particular ONU. The OLT shall deregister any ONU that does not participate in the eOAM discovery process, as defined in 13.3.

The ONU and OLT shall implement the eOAM discovery process by exchanging the *Organization Specific Information* TLV, as defined in IEEE Std 802.3, 57.5.2.3, and further specified in 13.2.4.1, referred to as *Extended Information* TLV. The *Extended Information* TLV is embedded in the *Information* OAMPDU, as defined in IEEE Std 802.3, 57.4.3.1. The format of the *Extended Information* TLV is defined in 13.2.4.1. An ONU shall include the *Extended Information* TLV in all *Information* OAMPDUs exchanged during the eOAM discovery process. An ONU shall start the eOAM discovery process not later than five seconds after the successful completion of the MPCP discovery and registration process.

The presence of the *Extended Information* TLV, indicating support for a specific version of the eOAM management suite, embedded in the *Information* OAMPDU transmitted by the ONU during the eOAM discovery process, indicates support of 13.3, Clause 13, and Clause 14. The lack of such an *Extended Information* TLV is treated as a lack of support for the requirements set forth in 13.3, Clause 13, and Clause 14, and consequently the OLT deregisters such an ONU as indicated above.

13.3.2.2 Ordering of *Organization Specific Information* TLVs

13.3.2.2.1 Source OAM Client requirements

A single IEEE Std 802.3, Clause 57, compliant *Information* OAMPDU may carry more than one *Organization Specific Information* TLV. To simplify both the reception and transmission processes, a specific order of transmission of such TLVs is required. In such a case, the *Local Information* TLV (IEEE Std 802.3, 57.5.2.1) and *Remote Information* TLV (IEEE Std 802.3, 57.5.2.2) shall be transmitted first, followed by the series of *Organization Specific Information* TLVs.

There are no specific transmission order requirements for *Organization Specific Information* TLVs. The *Extended Information* TLV as defined in 13.4.4.1 may be transmitted as the first *Organization Specific Information* TLV, followed by other *Organization Specific Information* TLVs, if present.

13.3.2.2.2 Destination OAM Client requirements

The destination OAM Client shall support the processing of multiple *Information* TLVs in a single *Information* OAMPDU, including *Local Information* TLV, *Remote Information* TLV, and at least one *Organization Specific Information* TLV.

The destination OAM Client shall process all received *Information* TLVs in the order of their reception, discarding any *Information* TLVs that are either malformed or unsupported. A malformed *Information* TLV is considered to have an invalid length and/or unexpected type value. An unsupported *Information* TLV follows the *Information* TLV format requirements but is marked with an OUI not supported by the given destination OAM Client.

13.3.2.3 Message flow during eOAM discovery process

The message flow during the eOAM discovery process is very simple. After the MPCP discovery and registration process is successfully completed, an ONU starts sending periodically *Information* OAMPDUs carrying the *Extended Information* TLV as defined in 13.4.4.1. Such an *Extended Information* TLV indicates the support of 13.3, Clause 13, and Clause 14. The ONU may also send additional *Organization Specific Information* TLVs if it supports other versions of management software. Their interpretation is outside the scope of this standard.

Having received the *Extended Information* TLV from an ONU, the OLT retrieves the version of the eOAM management suite supported by the device and proceeds to poll the ONU for more information, including software version and number of supported LLIDs, and configure it as needed.

The OLT is expected to support all the versions of the eOAM management suite that are supported by the fielded ONUs connected to its ports. The ONU does not learn, at any time during the eOAM discovery process, the eOAM management suite version supported by the OLT.

13.3.3 OAM and eOAM keep-alive process

During the OAM keep-alive process, *Information* OAMPDUs are exchanged between the OAM Clients to indicate that both link peers are operational. This process is defined in IEEE Std 802.3, 57.2.4. The *Information* OAMPDU, exchanged after the completion of the eOAM discovery process as defined in 13.3.2, may carry the *Extended Information* TLV or any other *Organization Specific Information* TLVs. An

ONU should include the *Extended Information* TLV in all *Information* OAMPDUs exchanged during the eOAM keep-alive process.

The failure of the OAM keep-alive process, as defined above, is treated as a critical link condition. If the ONU detects an OAM keep-alive failure, the ONU shall go through the MPCP deregistration process, as defined in IEEE Std 802.3ca, 144.3.7. If the OLT detects an OAM keep-alive failure for the given ONU, the OLT shall deregister the ONU following the MPCP deregistration process, as defined in IEEE Std 802.3ca, 144.3.7.

13.4 eOAMPDU structure

This subclause defines the internal structure of the eOAMPDU frame, i.e., the size and meaning of individual fields, and describes a TLV-oriented approach to packaging data in the eOAMPDU. Two specific types of the TLVs are also specified, namely the Variable Descriptor and the Variable Container.

The eOAMPDU format is derived from the IEEE 802.3 (Clause 57) OAM frame format, through the use of Organization Specific Extension mechanism, as described in detail in IEEE Std 802.3, 57.4.2 (Figure 57-9), and further extended in the following subclauses.

13.4.1 Extended OAM organizationally-unique identifier (OUI)

EPON devices shall implement OAM-based management protocols per IEEE Std 802.3, Clause 57. This standard defines eOAM mechanisms for managing various features defined in this standard. The eOAM mechanisms utilize the *Organization Specific* OAMPDU, as defined in IEEE Std 802.3, 57.4.3.6. OAMPDUs defined are identified by the IEEE Std 1904.4-specific organizationally unique identifier (OUI) value, as defined in Table 13-1. Note that the OUI value is shared with Package A defined in IEEE Std 1904.1.

Table 13-1—OUI values for SIEPON.4 features

OUI	Description	Value
OUI_1904_4	OUI value identifying IEEE Std 1904.4 OAMPDUs	0x58-D0-8F

13.4.2 eOAMPDU frame format

Size of the individual fields in the eOAMPDU and their meanings shall be as shown in Table 13-2 and meet the requirements included in the following description.

Table 13-2—Structure of the eOAMPDU frame

Size (octets)	Field (name)	Value	Notes
6	Destination Address	0x01-80-C2-00-00-02	eOAMPDU header
6	Source Address	Varies	
2	Length/Type	0x88-09 (Slow Protocol)	
1	Subtype	0x03 (OAM)	
2	Flags	Varies	
1	Code	0xFE (Organization Specific extensions)	
3	OUI	OUI_1904_4	
1	Opcode	Varies	
38 to 1492	Data + Pad	Varies	
4	FCS	Varies	

The eOAMPDU comprises the following fields:

a) `Destination Address` (`DA`). The `DA` in the eOAMPDUs is the Slow_Protocols_Multicast address (i.e., 0x01-80-C2-00-00-02). Its use and encoding are specified in IEEE Std 802.3, Annex 57A.

b) `Source Address` (`SA`). The `SA` in eOAMPDUs carries the individual MAC address associated with the port through which the eOAMPDU is transmitted.

c) `Length/Type`. The eOAMPDUs are always Type encoded and carry the Slow_Protocols_Type field value.

d) `Subtype`. The `Subtype` field identifies the specific Slow Protocol being encapsulated.

e) `Flags`. The `Flags` field contains status bits as defined in IEEE Std 802.3, 57.4.2.1.

f) `Code`. The `Code` field identifies the specific type of the OAMPDU. The use and encoding of this field are specified in IEEE Std 802.3, Table 57–4.

g) `OUI`. This field carries the organizationally unique identifier assigned to given organization.

h) `Opcode`. This field carries the value of the operation code, identifying a type of the eOAMPDU (see 13.4.6.1).

i) `Data/Pad`. This field contains the eOAMPDU data and any necessary padding. The data portion of the `Data/Pad` field carries a number of TLVs used to encode specific operations/values.

   The `Pad` field is as defined in IEEE Std 802.3, Clause 3.

j) `FCS`. This field is the frame check sequence, as defined in IEEE Std 802.3, Clause 4.

The fields described in item a) through item g) are jointly referred to as the eOAMPDU header.

13.4.3 TLV-oriented structure

The `Data/Pad` field in the eOAMPDU may carry at least one TLV, used to encode a specific set of values/operations. Individual TLV types for the eOAMPDU, i.e., Variable Descriptor and Variable Container, are defined in the following subclauses.

A series of TLVs carried in any of the *eOAM_Get_Request*, *eOAM_Get_Response*, *eOAM_Set_Request*, or *eOAM_Set_Response* eOAMPDUs shall be terminated with the Variable Descriptor with values carried in the `Branch` and `Leaf` fields equal to 0.

13.4.3.1 Variable Descriptor TLV

eOAM_Get_Request eOAMPDUs (see 13.4.6.2) use the list of Variable Descriptor TLVs and permit the management system to request the value of one or more managed objects hosted on the ONU, both defined in IEEE Std 802.3, Clause 30.

A Variable Descriptor TLV has the form of a 3-octet 2-tuple, comprising a 1-octet `Branch` code and a 2-octet `Leaf` code (together comprising the `Type` identifier), which unequivocally identifies the target attribute/action.

The structure of a Variable Descriptor shall be as presented in Table 13-3.

Table 13-3—Structure of the Variable Descriptor

Size (octets)	Field (name)		Value range
1	Branch	Type	0x00: End of list of TLVs 0x01 to 0xFF
2	Leaf		0x00-00 to 0xFF-FF

13.4.3.2 Variable Container TLV

eOAM_Get_Response eOAMPDUs (see 13.4.6.3), *eOAM_Set_Request* eOAMPDUs (see 13.4.6.4), and *eOAM_Set_Response* eOAMPDUs (see 13.4.6.5) use the list of Variable Container TLVs and permit

— The ONU to respond to the *eOAM_Get_Request* eOAMPDU; or

— The management system to set the value of one or more target managed attributes hosted on the ONU; or

— The ONU to respond to the *eOAM_Set_Request* eOAMPDU.

A Variable Container TLV has the form of a variable-length 4-tuple, comprising the following fields:

— A 1-octet-wide `Branch` field

— A 2-octet-wide `Leaf` field

— A 1-octet-wide `Length` field

— A variable-length `Value` field, the size of which is defined by the value carried in the `Length` field

The `Branch`/`Leaf` 2-tuple represents the `Type` field and unequivocally identifies the target attribute/action.

The `Length` code identifies the size of the following `Value` field, with additional restrictions:

— When the `Length` field value is in the range of 0x00 to 0x7F, it represents the length of the `Value` field, expressed in units of octets, where the value of 0x00 represents the length of the field equal to 128 octets and length in the range of 1 to 127 octets is mapped directly into the range of 0x01 to 0x7F (see IEEE Std 802.3, 57.6.2.1).

— When the `Length` field value is in the range of 0x80 to 0xFF, the value carried in this field represents the eOAMPDU return code and implies a zero length of the `Value` field (no additional value is carried in that case). The eOAMPDU return codes are defined in 13.4.7.

The `Value` field is present only if the `Length` field value is in the range of 0x00 to 0x7F and represents the following:

— In the case of *eOAM_Get_Response* eOAMPDUs, the value stored in the requested managed attribute

— In the case of *eOAM_Set_Request* eOAMPDUs, the value to be written into the target managed attribute

The structure of a Variable Container TLV shall be as presented in Table 13-4.

Table 13-4—Structure of the Variable Container

Size (octets)	Field (name)		Value	Comments
1	Branch	Type	0x00 to 0xFF	—
2	Leaf		0x00-00 to 0xFF-FF	—
1	Length		0x00 0x01 to 0x7F 0x80 to 0xFF	`Value` field is 128 octets long. `Value` field is 1 to 127 octets long. `Value` field is zero octets long, `Length` field represents the return code per 13.4.7.
Varies	Value		Variable	Present only when the value in the `Length` field is greater than zero; format as defined for the branch/leaf code

Variable Containers may contain data of a few common types, as defined below:

— **Integer**: An integer carried in a Variable Container shall be represented in the two's-complement form, with the Most Significant Octet (MSO) first. Note that Variable Containers are of variable length; as a result, attributes that are integers do not have a fixed width. The source OAM client may suppress leading zero octets of integers. The target OAM client shall accept an integer in a Variable Container of any legal width (1–128 octets). If a Variable Container is smaller (shorter) than the representation used by the target OAM client, the value is extended to match the representation used by the target OAM client as necessary. If the Variable Container is larger than the representation used by the target OAM client, the resulting action on the received value is implementation dependent.

— **Enumerated value**: An enumerated value carried in a Variable Container is a set of values with predefined meanings. Enumerated values always have a fixed length, regardless of the number of trailing zeros—effectively, enumerated values always have the maximum possible size anticipated for the particular managed attribute. Examples of valid enumerated values include (in binary format) 0b0000-0001-1000 or 0b0000-0000-0000. The source OAM client shall not suppress trailing zeros for enumerated values, and the target OAM client shall not add trailing zeros to the received enumerated values.

— **Sequence list**: A sequence list carried in a Variable Container has the form of a series (sequence) of values, typically of the enumerated value type. All elements in the sequence list shall be of the same length. The number of elements in the sequence list shall be determined from the size of the given Variable Container (value of the `Length` field).

13.4.3.3 TLVs carrying large values

The maximum length of data that can fit into a single Variable Container is equal to 128 octets. Some attribute values may be larger than the 128 octets, requiring a series of TLVs to transfer them between the source OAM client and the target OAM client, using a repeated branch/leaf tuple for the attribute in question. Such a series of TLVs is terminated with a TLV with the same branch/leaf tuple, and a length of zero, to indicate the end of multi-TLV value.

The value of such a large attribute is segmented into a number of individual TLVs, where each TLV in such a sequence of TLV carries a fragment of the large attribute and has the size meeting the requirements stipulated in 13.4.3.2. As a result, the value of such a large attribute is broken into a number of blocks, each with the size of at most 128 octets, and each such block is then carried in a dedicated TLV. For example, assume that the ONU has 23 MAC addresses stored in the dynamic MAC address table. The OLT requests the current list of MAC addresses learned by the ONU, in response to which the ONU needs to list all such MAC addresses using the respective TLVs. A single TLV can hold at most 21 whole MAC addresses (21 × 6 = 126 octets). In this case the first TLV would carry 21 MAC address, the second TLV would carry the remaining 23−21 = 2 MAC addresses, and this series of TLVs would be followed by a TLV with the same branch/leaf tuple but of zero size to indicate the end of the large value sequence.

13.4.4 TLVs for 802.3 OAMPDUs

13.4.4.1 *Extended Information* TLV

The *Information* OAMPDU defined in IEEE Std 802.3, Clause 57, can contain the Organization Specific Information as *Information* TLV (IEEE Std 802.3, 57.5.2.3). Presence of this *Extended Information* TLV in the *Information* OAMPDU during the OAM discovery process indicates that the OLT or the ONU supports the extended OAM.

The format of the *Extended Information* TLV shall be as specified in Table 13-5 and described in the following text.

Table 13-5—Structure of the *Extended Information* TLV

Size (octets)	Field (name)	Value
1	Type	0xFE (*Organization Specific Information* TLV)
1	Length	0x05
3	OUI	OUI_1904_4
1	InfoType	0x00
1	Version	This field identifies the version of the eOAM. Bits [7:4] represent the major version number Bits [3:0] represent the minor version number The following values are defined: 0x30: OAM compliant with IEEE Std 1904.4-202x, Other values are reserved and ignored on reception.

The following fields comprise the *Extended Information* TLV:

a) `Type`: this field is used to indicate the data type held in the given TLV. In the case of the *Extended Information* TLV, this field carries the value of 0xFE, according to IEEE Std 802.3, Table 57–6, indicating the *Organization Specific Information* TLV.

b) `Length`: this field is used to indicate the length of the TLV, expressed in units of octets.

c) `OUI`: this field is used to identify the organization to which the given *Information* TLV belongs. At least one of the *Organization Specific Information* TLVs exchanged between the ONU and the OLT during the eOAM discovery process shall be of *Extended Information* TLV type, containing the OUI_1904_4.

d) `InfoType`: this field is used to identify the subtype of the *Extended Information* TLV.

e) `Version`: this field is used to indicate the version of the eOAM supported by the given device. The internal format of this field is as follows: aaaa.bbbb (4 bits followed by 4 bits), where “aaaa” represents the major version number, and “bbbb” represents the minor version number. For example, a `Version` field carrying the value of 0b0010.0000 represents a major version 2, and a minor version 0.

13.4.4.2 *Event Notification* TLV

The basic structure of the *Organization Specific Event* TLV shall be as specified in IEEE Std 802.3, 57.5.3.5. Specific fields in the *Organization Specific Event* TLV shall be as shown in Figure 13-2 and specified below.

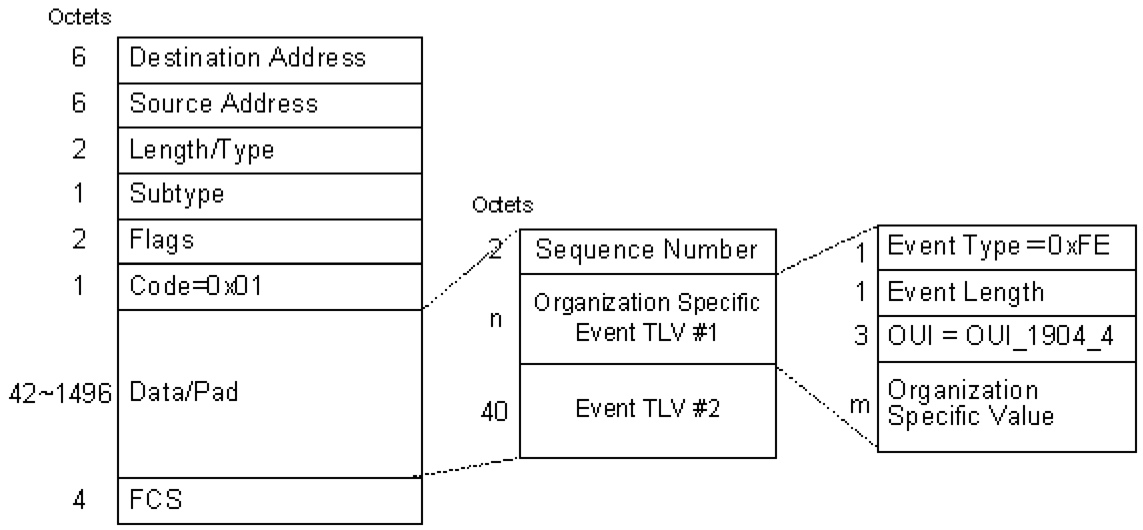

Figure 13-2—Relationship between *Organization Specific Event* TLV and the *Event Notification* OAMPDU

a) `Event Type` = 0xFE, according to the encoding of this field as defined in IEEE Std 802.3, Table 57–12.

b) `Event Length`. This one-octet field indicates the length (in octets) of this TLV-tuple.

c) `OUI` value, equal to OUI_1904_4.

d) `Organization Specific Value` carries the specific set of event-associated information. Further, the structure of the `Organization Specific Value` shall be as specified in Table 13-6 and described below.

Table 13-6—Internal structure of the `Organization Specific Value` field

Octet(s)	Field	Notes
1	EventCode	This field identifies the type of alarm that was identified by the source OAM client. See Table 13-7 for definition of individual values for the `EventCode` field. These alarm codes are grouped into link faults, critical events, and Dying Gasp alarm types, with code values numbered accordingly. Only the values listed in the table are supported. Other values are reserved and ignored on reception.
1	EventRaised	This field indicates whether the given event was raised. The following values are supported: 0x00: The given event was cleared. 0x01: The given event was raised. Other values are reserved and ignored on reception.
2	ObjectType	This field identifies the object element generating the alarm in question.

Octet(s)	Field	Notes
2 or 4	ObjectInstance	This field identifies the object element instance generating the alarm in question.

— `ObjectType` field identifies the object that generated the given event, as defined in 14.2.1.1. Other values of the `ObjectType` are reserved and ignored on reception.

— `ObjectInstance` field identifies the specific instance of the object that generated the given event, as defined in 14.2.1.2.

Table 13-7—Code points for the `EventCode` field

Event Group	Event Name	Code	Defined in
Link Fault Alarms	LoS	0x11	13.4.4.2.1
	Key Exchange Failure	0x12	13.4.4.2.2
Critical Event Alarms	Port Disabled	0x21	13.4.4.2.3
Dying Gasp Alarms	Power Failure	0x41	13.4.4.2.4
Other Alarms	Statistics Alarm	0x81	13.4.4.2.5
	ONU Busy	0x82	13.4.4.2.6
	MAC Table Overflow	0x83	13.4.4.2.7
	PON_IF_Switch	0x84	13.4.4.2.8

13.4.4.2.1 LoS (0x11)

For the PON port, a loss of signal (LoS) condition is detected by lack of incoming optical power or loss of clock and data recovery lock to the downstream bit clock. The transceiver status monitoring for the ONU and the OLT is as specified in 9.1.3. On any of the UNI ports, the LoS condition corresponds to the Link Down condition detected by the UNI port PHY.

13.4.4.2.2 Key Exchange Failure (0x12)

The Key Exchange Failure alarm indicates that a scheduled key exchange has failed. Encryption continues with the previous key for another key exchange interval. Another key exchange is attempted at the next key exchange time.

13.4.4.2.3 Port Disabled (0x21)

The Port Disabled event indicates that one of the ONU ports has been disabled by management action. If the PON port is disabled, then this event notification is not transmitted, and this alarm is visible only locally on the ONU.

13.4.4.2.4 Power Failure (0x41)

A Power Failure alarm indicates that the ONU lost power and is imminently going to be removed from the EPON. An ONU makes every attempt to send this *Event Notification* TLV when it detects loss of power. An ONU may not be able to actually send this *Event Notification* TLV if the required transmission grants are not allocated by the OLT before the ONU runs out of power.

13.4.4.2.5 Statistics Alarm (0x81)

The Statistics Alarm indicates a crossing of predefined thresholds on a specific statistic, as indicated by the *Alarm* TLV, as defined in Table 13-8. Typically, these thresholds would be set for counters for error conditions such as CRC errors.

Table 13-8—*Alarm* TLV structure

Size (octets)	Field (name)	Value
1	Branch	Branch of statistic that crossed threshold
2	Leaf	Leaf of statistic that crossed threshold

13.4.4.2.6 ONU Busy (0x82)

The ONU Busy alarm may be raised by an ONU to inform the OLT that it has been busy for an extended period and may have problems responding to any further OAM requests in the usual timely fashion.

13.4.4.2.7 MAC Table Overflow (0x83)

The MAC Table Overflow alarm is raised by an ONU to inform the OLT that an ingress MAC address has not been learned because the total number of MAC addresses has been exceeded. For example, if the ONU was provisioned to allow four MAC addresses on a particular UNI port, then the first four addresses seen would be learned; the fifth address would cause this alarm to be raised.

13.4.4.2.8 PON_IF_Switch (0x84)

The PON_IF_Switch alarm is raised by the ONU to inform the OLT that the PON interface on the ONU was switched from the active interface to backup interface, according to the tree protection mechanism defined in 9.3.4.

13.4.5 Multipart eOAMPDU response sequence

In some cases, responses to a single eOAMPDU transmitted by the OLT, received from the ONU, may not fit into a single eOAMPDU, especially when reading values from tables with the total size exceeding the payload of an eOAMPDU, e.g., MAC address tables. In this case, the ONU splits its response across multiple eOAMPDUs. The ONU shall inform the OLT that the complete response to the original request was not sent in a single eOAMPDU, but rather in a series of eOAMPDUs. In addition, the OLT shall be able to detect any missing eOAMPDUs in the series of eOAMPDUs comprising a complete response from the ONU.

To indicate that additional eOAMPDUs comprising a complete response from the ONU are forthcoming, the ONU shall add an instance of the *Sequence* TLV (0xDB/0x00-01) to the response eOAMPDU to denote the response sequence. The ONU should not add the *Sequence* TLV (0xDB/0x00-01) to an eOAMPDU not being part of the response sequence.

To send a multiple part response requiring *N* eOAMPDUs, the ONU does the following:

- For the first eOAMPDU in the response sequence:
  - Set the value in the `Sequence#` field to 0x00.
- For the last eOAMPDU in the response sequence:
  - Set bit 15 in the `Sequence#` field to 1.
- For all eOAMPDUs in the response sequence:
  - Add the *Sequence* TLV (0xDB/0x00-01) to the eOAMPDU.
  - Increment the value of the `Sequence#` field.

Figure 13-3 presents examples of various eOAMPDU exchange sequences.

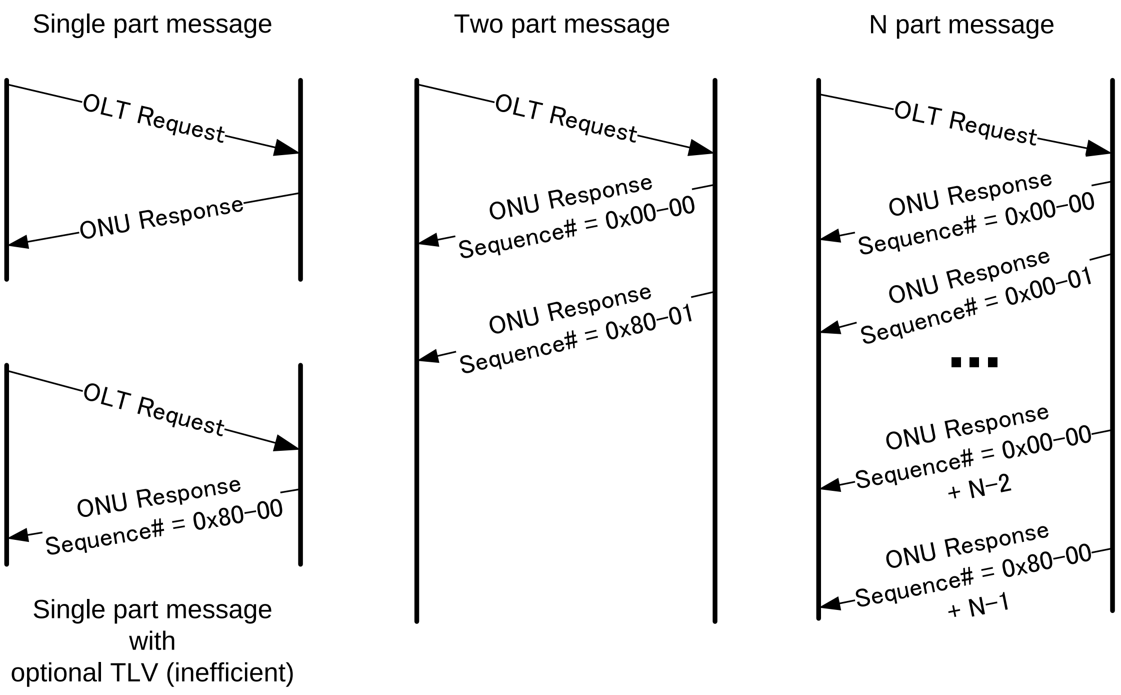

Figure 13-3—Example of various eOAMPDU exchange sequences

13.4.6 eOAMPDU types

Most management functions required for the proper operation of EPON are carried out through the process of reading and writing individual attributes of the managed objects hosted in the ONU. As an example, setting the operating speed for an UNI port requires writing an appropriate value into the speed attribute of the proper port object. Likewise, information can be read from the respective managed objects hosted on the ONU using dedicated eOAMPDUs.

It is also possible to cause the ONU to perform certain actions, e.g., disable specific UNI ports, reset counters, by setting appropriate values in the stored managed objects.

13.4.6.1 eOAMPDU codes

eOAMPDUs shall be as defined in Table 13-9. These eOAMPDUs use the Organization Specific Extension mechanisms defined in IEEE Std 802.3, Clause 57. Other values are reserved and ignored on reception.

Table 13-9—eOAMPDUs and assignment of Opcode values

Opcode	eOAMPDUs	Defined in
0x00	Reserved, ignored on reception	
0x01	*eOAM_Get_Request*	13.4.6.2
0x02	*eOAM_Get_Response*	13.4.6.3
0x03	*eOAM_Set_Request*	13.4.6.4
0x04	*eOAM_Set_Response*	13.4.6.5
0x05	Reserved, ignored on reception	
0x06	Reserved, ignored on reception	
0x07	Reserved, ignored on reception	
0x08	*eOAM_KeyExchange*	13.4.6.7
0x09	*eOAM_Software*	13.4.6.6
0x0A	Reserved, ignored on reception	
0x0B	Reserved, ignored on reception	

Opcode	eOAMPDUs	Defined in
0x0C	Reserved, ignored on reception	
0xFC	*eOAM_Early_WakeUpOLT*	13.4.6.8
0xFD	*eOAM_Early_WakeUpONU*	13.4.6.9
0xFE	*eOAM_Sleep_Allowed*	13.4.6.10

13.4.6.2 *eOAM_Get_Request* eOAMPDU

The *eOAM_Get_Request* eOAMPDU permits the management system to request the value of one or more attributes hosted on the ONU, both defined in IEEE Std 802.3, Clause 30. The `Data` field of the *eOAM_Get_Request* eOAMPDU contains a series of Variable Descriptors and *Object Context* TLVs, if needed. The size (and presence) of the `Pad` field depends on the number of individual Variable Descriptors and *Object Context* TLVs. The structure of the Variable Descriptor is defined in 13.4.3.1. The structure of the *Object Context* TLV is defined in 14.2.1.

Functionally, the *eOAM_Get_Request* eOAMPDU is identical to the Variable Request OAMPDU as defined in IEEE Std 802.3, 57.4.3.3.

The structure of the *eOAM_Get_Request* eOAMPDU shall be as specified in Table 13-10 and as described in more detail below.

Table 13-10—Structure of the *eOAM_Get_Request* eOAMPDU

Size (octets)	Field (name)	Value
21	eOAMPDU header	Varies
1	Opcode	0x01
Varies	Data	Varies, a series of Variable Descriptors and *Object Context* TLV
Varies	Pad	0x00-…-00
4	FCS	Varies

eOAMPDU header, `Opcode`, `Data`, `Pad`, and `FCS` fields are defined in 13.4.2.

13.4.6.3 *eOAM_Get_Response* eOAMPDU

The *eOAM_Get_Response* eOAMPDU permits the ONU to respond to the *eOAM_Get_Request* eOAMPDU and contains a series of Variable Containers and *Object Context* TLVs, if needed.. The size (and presence) of the `Pad` field depends on the number of individual Variable Containers and *Object Context* TLVs. Each Variable Container may carry the value of the requested variable or a return code (per 13.4.7) if the variable reading process fails for any reason. The structure of the Variable Container is defined in 13.4.3.2. The structure of the *Object Context* TLV is defined in 14.2.1.

Functionally, the *eOAM_Get_Response* eOAMPDU is identical to the Variable Response OAMPDU as defined in IEEE Std 802.3, 57.4.3.4.

The structure of the *eOAM_Get_Response* eOAMPDU shall be as specified in Table 13-11 and as described in more detail below.

Table 13-11—Structure of the *eOAM_Get_Response* eOAMPDU

Size (octets)	Field (name)	Value
21	eOAMPDU header	Varies
1	Opcode	0x02
Varies	Data	Varies, a series of Variable Containers and *Object Context* TLVs
Varies	Pad	0x00-…-00

Size (octets)	**Field (name)**	**Value**
4	FCS	Varies

eOAMPDU header, `Opcode`, `Data`, `Pad`, and `FCS` fields are defined in 13.4.2.

The size of individual Variable Container(s) ranges between 5 and 132 octets, with the maximum size limited by the `Length` field encoding used in the Variable Container, as defined in 13.4.3.2.

13.4.6.4 *eOAM_Set_Request* eOAMPDU

The *eOAM_Set_Request* eOAMPDU permits the management system to set the value of one or more attributes hosted on the ONU, both defined in IEEE Std 802.3, Clause 30. The `Data` field of the *eOAM_Set_Request* eOAMPDU contains a series of Variable Containers and *Object Context* TLVs, if needed. The size (and presence) of the `Pad` field depends on the number of individual Variable Containers and *Object Context* TLVs. The structure of the Variable Container is defined in 13.4.3.2. The structure of the *Object Context* TLV is defined in 14.2.1.

The *eOAM_Set_Request* eOAMPDU does not have a functional equivalent in the OAMPDU defined in IEEE Std 802.3, Clause 57. IEEE 802.3 OAM does not support operations related to setting attributes and actions.

The structure of the *eOAM_Set_Request* eOAMPDU shall be as specified in Table 13-12 and as described in more detail below.

Table 13-12—Structure of the *eOAM_Set_Request* eOAMPDU

Size (octets)	**Field (name)**	**Value**
21	eOAMPDU header	Varies
1	Opcode	0x03
Varies	Data	Varies, a series of *M* Variable Containers and *Object Context* TLVs
Varies	Pad	0x00-…-00
4	FCS	Varies

eOAMPDU header, `Opcode`, `Data`, `Pad`, and `FCS` fields are defined in 13.4.2.

Each Variable Container included in the *eOAM_Set_Request* eOAMPDU may map into an attribute (e.g., for branches 0x07 or 0xDB) or an action (e.g., for branches 0x09 or 0xDD), as indicated by the `Branch`/`Leaf` 2-tuple. The `Value` field provides a new value to be assigned to the target attribute or a parameter for the target action.

Actions instruct the target eOAM client to execute a procedure, e.g., rebooting the ONU. The management actions specified in IEEE Std 802.3, Clause 30, are not supported in the IEEE Std 802.3 (Clause 57) OAMPDUs. The OAM extensions allow the source eOAM client to request execution of both actions defined by IEEE Std 802.3, Clause 30, and actions. Some of the actions are expressed using the Variable Container, where the parameters for this action are carried in the body of the Variable Container. Actions that do not have parameters are represented with a Variable Container of zero length (`Length` value of 0x80).

13.4.6.5 *eOAM_Set_Response* eOAMPDU

The *eOAM_Set_Response* eOAMPDU permits the ONU to respond to management requests (read variable[s], set variable[s], or perform action[s]) and contains a series of Variable Containers and *Object*

Context TLVs, if needed. The size (and presence) of the `Pad` field depends on the number of individual Variable Containers and *Object Context* TLVs. Each Variable Container carries a return code (see 13.4.7) together with the `Branch`/`Leaf` identification of the target attribute/action. The structure of the Variable Container is defined in 13.4.3.2. The structure of the *Object Context* TLV is defined in 14.2.1.

The structure of the *eOAM_Set_Response* eOAMPDU shall be as specified in Table 13-13 and as described in more detail below.

Table 13-13—Structure of the *eOAM_Set_Response* eOAMPDU

Size (octets)	Field (name)	Value
21	eOAMPDU header	Varies
1	Opcode	0x04
Varies	Data	Varies, a series of Variable Containers and *Object Context* TLVs
Varies	Pad	0x00-…-00
4	FCS	Varies

eOAMPDU header, `Opcode`, `Data`, `Pad`, and `FCS` fields are defined in 13.4.2.

13.4.6.6 *eOAM_Software* eOAMPDU

This subclause provides the definition of the generic *eOAM_Software* eOAMPDU, together with the specific eOAMPDU subtypes required to implement the software update mechanism. The software update mechanism is specified in 12.3.3.

13.4.6.6.1 *eOAM_Software* eOAMPDU structure

The *eOAM_Software* eOAMPDU is a specific type of the generic eOAMPDU, as defined in Table 13-8.

The generic structure of the *eOAM_Software* eOAMPDU shall be as presented in Table 13-14 and as described in more detail below.

Table 13-14—Structure of the *eOAM_Software* eOAMPDU

Size (octets)	Field (name)	Value + notes
21	eOAMPDU header	Varies
1	Opcode	0x09
1	FileTransferOpcode	Indicates the type of the *eOAM_Software* eOAMPDU, per Table 13-15.
Varies	FileTransferBody	Carries the actual data portion of the *eOAM_Software* eOAMPDU, depending on the value of the `FileTransferOpcode` field.
Varies	Pad (optional)	0x00-…-00
4	FCS	Varies

Table 13-15—Values of the `FileTransferOpcode` field

`FileTransferOpcode` value	Value
Reserved	0x00
eOAM_Software_WriteRequest	0x01
eOAM_Software_FileTransferData	0x02
eOAM_Software_FileTransferAck	0x03

a) eOAMPDU header, as defined in 13.4.2.

b) `Opcode`, as defined in 13.4.2. This field carries the value of 0x09 for *eOAM_Software* eOAMPDU.

c) `FileTransferOpcode` indicates the type of the *eOAM_Software* eOAMPDU. Three types of *eOAM_Software* eOAMPDUs are defined:

   0x01: *eOAM_Software_WriteRequest* eOAMPDU is used by the OLT to initiate the ONU software image transfer process.

   0x02: *eOAM_Software_FileTransferData* eOAMPDU is used by the OLT to transfer a fragment of the given ONU software image between the OLT and the ONU.

   0x03: *eOAM_Software_FileTransferAck* eOAMPDU is used by the ONU to provide a return code to the OLT, indicating the current status of the ONU software image transfer process.

   Other values are reserved and ignored on reception.

d) `FileTransferBody` carries the actual information related to the given software upgrade process. There are several supported messages types, as specified by the `FileTransferOpcode` field.

   Individual *eOAM_Software* eOAMPDUs (*eOAM_Software_WriteRequest* eOAMPDU, *eOAM_Software_FileTransferData* eOAMPDU, and *eOAM_Software_FileTransferAck* eOAMPDU) are further defined in the following subclauses.

   The size of this field is variable and depends on the eOAMPDU subtype as indicated in the `Type` field.

e) `Pad`, as defined in 13.4.2. The length of this field is variable and depends on the size of the total size of the `FileTransferOpcode` and `FileTransferBody` fields.

f) `FCS`, as defined in 13.4.2.

13.4.6.6.2 *eOAM_Software_WriteRequest* eOAMPDU

The *eOAM_Software_WriteRequest* eOAMPDUs is used by the OLT to initiate a file transfer from the OLT to the selected ONU and deliver the name of the ONU software filename to be stored in the *aOnuFwFileName* (0xDB/0x01-0E) attribute. After this eOAMPDU is received, the ONU prepares for the reception of the software image.

The structure of the *eOAM_Software_WriteRequest* eOAMPDU shall be as specified in Table 13-16 and as described in more detail below.

Table 13-16—Structure of the *eOAM_Software_WriteRequest* eOAMPDU

Size (octets)	Field	Value + notes
21	eOAMPDU header	Varies
1	Opcode	0x09
1	FileTransferOpcode	0x01
Varies	FileName	null-terminated ASCII string
Varies	Pad (optional)	0x00-…-00
4	FCS	Varies

a) eOAMPDU header, `Opcode`, `FileTransferOpcode`, `Pad`, and `FCS` fields are defined in 13.4.6.6.1.

b) `FileTransferOpcode` identifies the *eOAM_Software_WriteRequest* eOAMPDU.

c) `FileName` represents the ONU software filename, to be stored at the ONU in the *aOnuFwFileName* (0xDB/0x01-0E) attribute.

13.4.6.6.3 *eOAM_Software_FileTransferData* eOAMPDU

The *eOAM_Software_FileTransferData* eOAMPDUs are used to carry individual fragments of the ONU software image file. Each eOAMPDU carries the block number (`BlockNumber` field) and data fragment size indicator (`BlockWidth` field), specifying the number of file data octets to follow.

The structure of the *eOAM_Software_FileTransferData* eOAMPDU shall be as specified in Table 13-17 and as described in more detail below.

Table 13-17—Structure of the *eOAM_Software_FileTransferData* eOAMPDU

Size (octets)	Field (name)	Value + notes
21	eOAMPDU header	Varies
1	Opcode	0x09
1	FileTransferOpcode	0x02
2	BlockNumber	This field reflects the sequential number of the current ONU software image fragment carried in this eOAMPDU.
2	BlockWidth	This field reflects the size of the `BlockData` field. Its value is expressed in units of octets. When the value of this field is equal to 0x00-00, this eOAMPDU is used to keep the ONU software image transfer process alive.
Varies	BlockData	This field carries the actual fragment of the ONU software image.
Varies	Pad (optional)	0x00-…-00
4	FCS	Varies

a) eOAMPDU header, `Opcode`, `FileTransferOpcode`, `Pad`, and `FCS` fields are defined in 13.4.6.6.1.

b) `FileTransferOpcode` identifies the *eOAM_Software_FileTransferData* eOAMPDU.

c) `BlockNumber` contains the sequential number of the current ONU software image fragment carried in the *eOAM_Software_FileTransferData* eOAMPDU.

d) `BlockWidth` represents the size of `BlockData`. Its value is expressed in units of octets. When the *eOAM_Software_FileTransferData* eOAMPDU is used to keep the ONU software image transfer process alive (keep-alive message), the value of this field is equal to 0x00-00.

e) `BlockData` carries the actual fragment of the ONU software image.

13.4.6.6.4 *eOAM_Software_FileTransferAck* eOAMPDU

The *eOAM_Software_FileTransferAck* eOAMPDU is used by the ONU to indicate the current status of the ONU software image transfer process. Each eOAMPDU of this type carries the block number (`BlockNumber` field) and the response code (`ResponseCode` field). The block number indicates the number of the next ONU software image data block expected by the ONU.

The *eOAM_Software_FileTransferAck* eOAMPDU is also used by the OLT to indicate the end of the ONU software image file. In this case, the *eOAM_Software_FileTransferAck* eOAMPDU carries the `BlockNumber` value of 0x00-00, together with the `ResponseCode` value indicating the end of the transfer process.

The structure of the *eOAM_Software_FileTransferAck* eOAMPDU shall be as specified in Table 13-18 and as described in more detail below.

Table 13-18—Structure of the *eOAM_Software_FileTransferAck* eOAMPDU

Size (octets)	Field (name)	Value + notes
21	eOAMPDU header	Varies
1	Opcode	0x09
1	FileTransferOpcode	0x03
2	BlockNumber	This field carries the sequential number of the software image fragment (block) that the ONU expects to receive next.
1	ResponseCode	This field carries the response code generated by the sender entity (either ONU or OLT).
Varies	Pad (optional)	0x00-…-00
4	FCS	Varies

a) eOAMPDU header, `Opcode`, `FileTransferOpcode`, `Pad`, and `FCS` fields are defined in 13.4.6.6.1.

b) `FileTransferOpcode` identifies the *eOAM_Software_FileTransferData* eOAMPDU.

c) `BlockNumber` contains the number of the software image block, as described in 13.4.6.6.3.

d) `ResponseCode` carries the response code, as defined in Table 13-19. Only the values specified in Table 13-19 are allowed. Other values are reserved and cause the *eOAM_Software_FileTransferAck* eOAMPDU to be ignored.

Table 13-19—Response Code values carried in `ResponseCode` field

Response Code	Value	Meaning
OK	0x00	No errors.
Undefined	0x01	Unknown error, or one not covered elsewhere.
Not Found	0x02	Read requested file that is not available.
No Access	0x03	Access permissions do not allow the requested read/write.
Full	0x04	Storage is full, and cannot hold the written file.
Illegal Operation	0x05	Cannot perform requested operation in current state.
Unknown ID	0x06	Requested file ID is not supported by this device.
Bad Block	0x07	Block received in error.
Timeout	0x08	No block received before timer expiration.
Busy	0x09	Cannot perform requested action due to other activity.
Incompatible File	0x0A	Received file is incompatible with this device. File incompatibility is determined by the device vendor.
Corrupted File	0x0B	File was received corrupted and is unusable by this device. File integrity is determined by the device vendor.

13.4.6.7 *eOAM_KeyExchange* eOAMPDU

The *eOAM_KeyExchange* eOAMPDU is used to implement the key exchange protocol between the OLT and the ONU.

13.4.6.7.1 *eOAM_KeyExchange* eOAMPDU structure

The *eOAM_KeyExchange* eOAMPDU is a specific type of the generic eOAMPDU, as defined in Table 13-2.

The structure of the *eOAM_KeyExchange* eOAMPDU shall be as presented in Table 13-20 and as described in more detail below.

Table 13-20—Structure of the *eOAM_KeyExchange* eOAMPDU

Size (octets)	Field (name)	Value + notes
21	eOAMPDU header	Varies
1	Opcode	0x08
1	KeyExchangeOpcode	Indicates the type of the *eOAM_KeyExchange* eOAMPDU.
Varies	KeyExchangeBody	Carries the actual data portion of the *eOAM_KeyExchange* eOAMPDU, depending on the value of the `Key Exchange Opcode` field.
Varies	Pad (optional)	0x00-…-00
4	FCS	Varies

a) eOAMPDU header, as defined in 13.4.2.

b) `Opcode`, as defined in 13.4.2. This field carries the value of 0x08 for *eOAM_KeyExchange* eOAMPDU.

c) `KeyExchangeOpcode` indicates the type of the *eOAM_KeyExchange* eOAMPDU. Two types of *eOAM_KeyExchange* eOAMPDUs are defined:

   0x00: *eOAM_KeyExchange_Assign* eOAMPDU is used to assign the encryption key.

   0x01: *eOAM_KeyExchange_ACK* eOAMPDU is used to acknowledge the assignment of the encryption key.

   Other values are reserved and ignored on reception.

d) `KeyExchangeBody` carries the actual information related key exchange process. There are several supported messages types, as specified by the `Key Exchange Opcode` field.

   Individual *eOAM_KeyExchange* eOAMPDUs (*eOAM_KeyExchange_Assign* and *eOAM_KeyExchange_ACK* eOAMPDU) are further defined in the following subclauses.

   The size of this field is variable and depends on the eOAMPDU subtype as indicated in the `Type` field.

e) `Pad`, as defined in 13.4.2. The length of this field is variable and depends on the size of the total size of the `KeyExchangeOpcode` and `KeyExchangeBody` fields.

f) `FCS`, as defined in 13.4.2.

13.4.6.7.2 *eOAM_KeyExchange_Assign* eOAMPDU

The *eOAM_KeyExchange_Assign* eOAMPDU is used to assign the new encryption key to the link peer.

The structure of the *eOAM_KeyExchange_Assign* eOAMPDU shall be as specified in Table 13-21 and as described in more detail below.

Table 13-21—Structure of the *eOAM_KeyExchange_Assign* eOAMPDU

Size (octets)	Field (name)	Value + notes
21	eOAMPDU header	Varies
1	Opcode	0x08
1	KeyExchangeOpcode	0x00

Size (octets)	Field (name)	Value + notes
2	LLID	This field carries the value of LLID (as in LLID carried in the frame preamble) for L-ONU to which this eOAMPDU applies. The supported range of values is 0x00-00 to 0x7F-FF. Other values are reserved and ignored on reception.
1	KeyNumber	This field indicates the key exchange phase. The supported range of value is 0x00 to 0x01. Other values are reserved and ignored on reception
1	KeyLength	This field indicates the length of the encryption key. The value is expressed in units of octets.
Varies	Key	This field carries the actual encryption key of the length indicates by the `Key Length` field.
Varies	Pad (optional)	0x00-…-00
4	FCS	Varies

a) eOAMPDU header, `Opcode`, `Pad`, and `FCS` fields are defined in 13.4.2.

b) `KeyExchangeOpcode` identifies the *eOAM_KeyExchange_Assign* eOAMPDU.

c) `LLID` indicates the value of L-ONU LLID to which this *eOAM_KeyExchange_Assign* eOAMPDU refers.

d) `KeyNumber` indicates the key exchange phase, indicating to the receiving link peer whether the current or previous key is to be used.

e) `KeyLength` provides information on the length of the actual encryption key, expressed in units of octets.

f) `Key` carries the actual encryption key.

13.4.6.7.3 *eOAM_KeyExchange_ACK* eOAMPDU

The *eOAM_KeyExchange_ACK* eOAMPDU is used by the link peer to confirm the assignment of the new encryption key.

The structure of the *eOAM_KeyExchange_ACK* eOAMPDU shall be as specified in Table 13-22 and as described in more detail below.

Table 13-22—Structure of the *eOAM_KeyExchange_ACK* eOAMPDU

Size (octets)	Field (name)	Value + notes
21	eOAMPDU header	Varies
1	Opcode	0x08
1	KeyExchangeOpcode	0x01
2	LLID	This field carries the value of LLID (as in LLID carried in the frame preamble) for L-ONU to which this eOAMPDU applies. The supported range of values is 0x00-00 to 0x7F-FF. Other values are reserved and ignored on reception.
1	KeyNumber	This field indicates the key exchange phase. The supported range of value is 0x00 to 0x01. Other values are reserved and ignored on reception.
Varies	Pad (optional)	0x00-…-00
4	FCS	Varies

a) eOAMPDU header, `Opcode`, `Pad`, and `FCS` fields are defined in 13.4.2.

b) `KeyExchangeOpcode` identifies the *eOAM_KeyExchange_ACK* eOAMPDU.

c) `LLID` indicates the value of L-ONU LLID to which this *eOAM_KeyExchange_ACK* eOAMPDU refers.

d) `KeyNumber` indicates the key exchange phase, indicating to the receiving link peer whether the current or previous key is to be used.

13.4.6.8 *eOAM_Early_WakeUpOLT* eOAMPDU

The OLT with enabled support for early wake-up function sends the *eOAM_Early_WakeUpOLT* eOAMPDU to request the ONU to leave the sleep state and enter the active state.

The structure of the *eOAM_Early_WakeUpOLT* eOAMPDU shall be as specified in Table 13-23 and as described in more detail below.

Table 13-23—Structure of the *eOAM_Early_WakeUpOLT* eOAMPDU

Size (octets)	Field (name)	Value
21	eOAMPDU header	Varies
1	Opcode	0xFC
38	Pad	0x00-…-00
4	FCS	Varies

eOAMPDU header, `Opcode`, `Pad`, and `FCS` fields are defined in 13.4.2.

13.4.6.9 *eOAM_Early_WakeUpONU* eOAMPDU

The ONU sends the *eOAM_Early_WakeUpONU* eOAMPDU to indicate to the OLT that it left the sleep state and entered the active state. This information allows the OLT to enable the downstream queues and resume downstream transmission to this particular ONU.

The structure of the *eOAM_Early_WakeUpONU* eOAMPDU shall be as specified in Table 13-24 and as described in more detail below.

Table 13-24—Structure of the *eOAM_Early_WakeUpONU* eOAMPDU

Size (octets)	Field (name)	Value
21	eOAMPDU header	Varies
1	Opcode	0xFD
38	Pad	0x00-…-00
4	FCS	Varies

eOAMPDU header, `Opcode`, `Pad`, and `FCS` fields are defined in 13.4.2.

13.4.6.10 *eOAM_Sleep_Allowed* eOAMPDU

The *eOAM_Sleep_Allowed* eOAMPDU is used by the OLT to request the ONU to enter the specified sleep mode (indicated by the `SleepMode` field) for a specific duration of time (indicated by the `SleepDuration` field).

The structure of the *eOAM_Sleep_Allowed* eOAMPDU shall be as specified in Table 13-25.

Table 13-25—Structure of the *eOAM_Sleep_Allowed* eOAMPDU

Size (octets)	Field (name)	Value
21	eOAMPDU header	Varies
1	Opcode	0xFE
1	SleepMode	Sleep mode requested by the OLT
4	SleepDuration	The duration of the sleep state, expressed in units of time quanta
Varies	Pad	0x00-…-00
4	FCS	Varies

eOAMPDU header, `Opcode`, `Pad`, and `FCS` fields are defined in 13.4.2.

13.4.7 eOAMPDU return codes

The eOAMPDU generated by the ONU in response to OLT-side query eOAMPDU may carry return codes when the `Length` field value in the Variable Container is in the range of 0x80 to 0xFF. Specific return codes shall be as specified in Table 13-26. Other values are reserved and ignored on reception.

Per the definition of the Variable Container (see 13.4.3.2), when bit 7 in the `Length` field is set, the `Value` field is not present in the Variable Container.

Table 13-26—Return codes

Code	Name	Description
0x80	No Error	The operation was successfully completed.
0x81	Too Long	Length of result exceeded eOAMPDU data field available.
0x86	Bad Parameters	Parameters for the requested action fail error checking.
0x87	No Resources	The device does not currently have the resources (table entries, memory, etc.) to perform the requested action.
0x88	System Busy	The device is not currently in the proper state to perform the requested action.
0xA0	Undetermined Error	Unknown or unlisted attribute error.
0xA1	Unsupported	An attribute requested is not supported on this device.
0xA2	May Be Corrupted	The value of an attribute counter may be invalid due to reset.
0xA3	Hardware Failure	An attribute hardware error prevented the operation from completing.
0xA4	Overflow	The requested attribute experienced overflow error.

NOTE—Specific return codes may be carried in either *eOAM_Set_Response* eOAMPDU or *eOAM_Get_Response* eOAMPDU.

The OLT at its own discretion may send multiple TLVs in a single *eOAM_Set_Request* eOAMPDU or *eOAM_Get_Request* eOAMPDU, covering multiple attributes and/or actions.

The ONU shall provide exactly one TLV with the return code for each attribute/action TLV included in the received *eOAM_Set_Request* eOAMPDU. The ONU shall provide either exactly one TLV with the return code or at least one TLV with the value of the requested attribute for each attribute TLV included in the received *eOAM_Get_Request* eOAMPDU. The number of *eOAM_Set_Response* or *eOAM_Get_Response* eOAMPDUs generated by the ONU depends on the number of response TLVs generated by the ONU in response to attribute/action TLVs in the received *eOAM_Set_Request* or *eOAM_Get_Request* eOAMPDU.

If a TLV in the *eOAM_Set_Request* eOAMPDU requires the accompanying *Object Context* TLV, the return code in the *eOAM_Set_Response* eOAMPDU shall be preceded by the same *Object Context* TLV. If the series of return codes to the given TLVs in the *eOAM_Set_Request* eOAMPDU does not fit into one *eOAM_Set_Response* eOAMPDU, the remaining part of the series of return codes shall be preceded by the appropriate *Object Context* TLV.

13.5 Software update

The software upgrade mechanism allows an ONU to receive a new software image from the OLT, verify it, and switch to using the new image (i.e., load and execute the new software image upon the reboot.)

In this subclause, the following terms are used extensively:

— **software**: software and/or firmware. The process of upgrading ONU software and/or firmware is the same, and specific separation of the downloaded software image is at the discretion of the given system provider. It is also outside the scope of this standard.

— **committed image**: the software image stored in the ONU’s permanent memory and marked to be used (i.e., loaded and executed) upon the ONU restart.

— **committing**: the process involving storing the software image in the ONU’s permanent memory, verifying the integrity of the stored image, and marking the image to be used on next ONU restart. The committing process does not invoke an automatic ONU restart. However, on subsequent restarts, the image marked as committed is used.

13.5.1 Software image download process

The software image download process is outlined in Figure 13-4 and is further defined in the state diagram shown in Figure 13-5 for the ONU and in the state diagram shown in Figure 13-6 for the OLT.

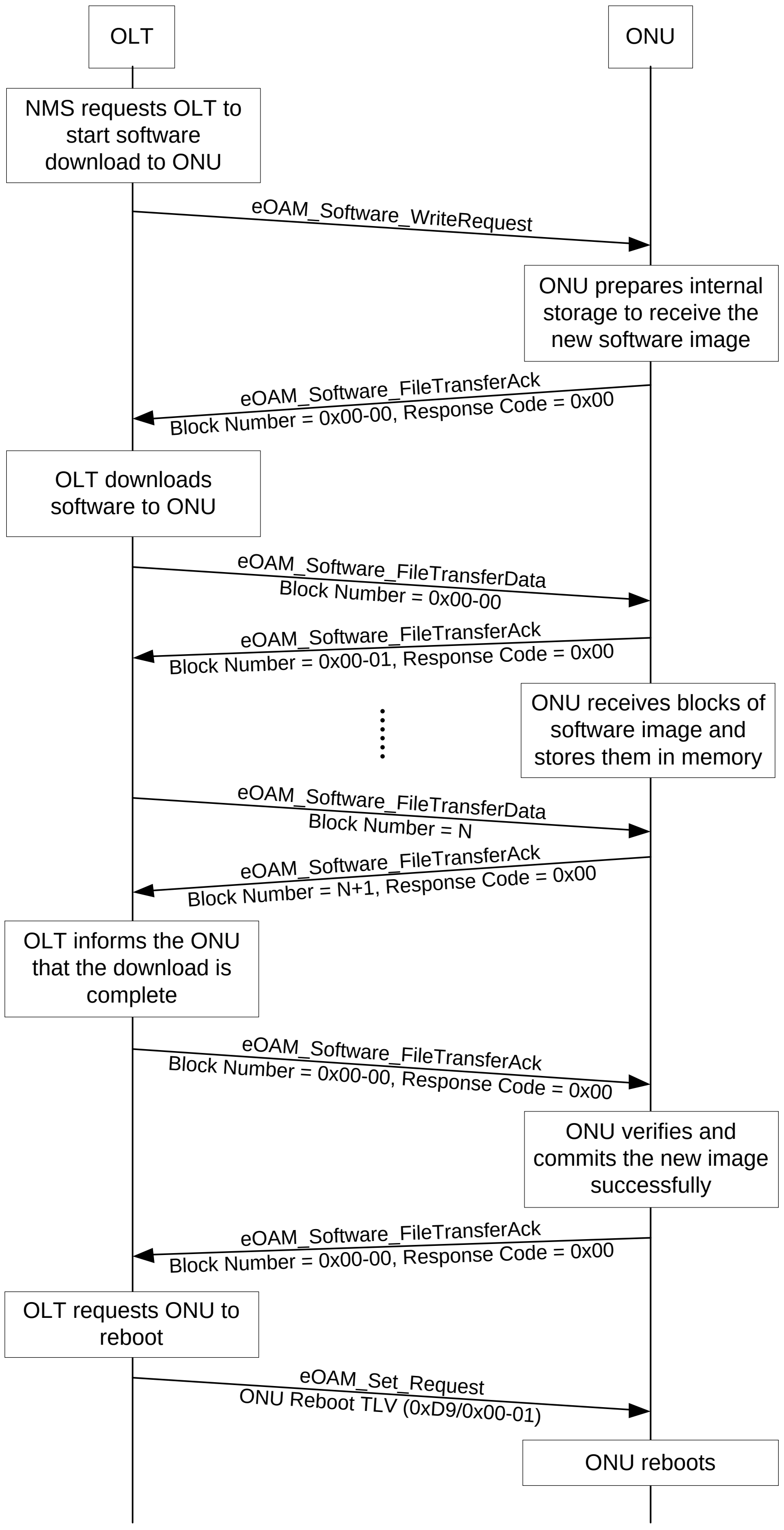
OLT
ONU
NMS requests OLT to start software download to ONU
eOAM_Software_WriteRequest
ONU prepares internal storage to receive the new software image
eOAM_Software_FileTransferAck
Block Number = 0x00-00, Response Code = 0x00
OLT downloads software to ONU
eOAM_Software_FileTransferData
Block Number = 0x00-00
eOAM_Software_FileTransferAck
Block Number = 0x00-01, Response Code = 0x00
ONU receives blocks of software image and stores them in memory
eOAM_Software_FileTransferData
Block Number = N
eOAM_Software_FileTransferAck
Block Number = N+1, Response Code = 0x00
OLT informs the ONU that the download is complete
eOAM_Software_FileTransferAck
Block Number = 0x00-00, Response Code = 0x00
ONU verifies and commits the new image successfully
eOAM_Software_FileTransferAck
Block Number = 0x00-00, Response Code = 0x00
OLT requests ONU to reboot
eOAM_Set_Request
ONU Reboot TLV (0xD9/0x00-01)
ONU reboots

Figure 13-4—Data flow during a successful software image download and committing process

During the eOAM discovery process, the OLT learns the basic information about the ONU, including type, chipset version, ONU capabilities, software version, etc. This information is used by the NMS to determine whether the software in the given registering ONU needs to be upgraded.

The decision to perform a software upgrade for the given ONU remains at the sole discretion of the NMS. The NMS requests the OLT to initiate the software upgrade process for a given ONU as outlined in Figure 13-4.

The software image download process has the form of a file transfer from the OLT to the ONU. This process is similar to TFTP, but it includes a number of EPON-specific optimizations:

— The transfer protocol operates over the IEEE 802.3 OAM channel instead of IP channel. It uses variable-length eOAMPDUs specified in 13.4.6.6.
— The transfer mode includes only binary data encoding.
— The ONU responses indicate the next block that the ONU expects to receive rather than acknowledge the last received block.

The software upgrade process comprises the following steps:

— Download initiation
— Download
— Verification
— Committing

13.5.1.1 Download initiation step

The software image download step is started by the NMS by requesting the OLT to download the updated software image for the selected ONU. In response to this request, the OLT sends the *eOAM_Software_WriteRequest* eOAMPDU (as specified in 13.4.6.6.2), containing the ONU software filename to be stored in the *aOnuFwFileName* (0xDB/0x01-0E) attribute. The ONU responds to this request by sending either

a) *eOAM_Software_FileTransferAck* eOAMPDU with `BlockNumber` = 0x00-00 and `ResponseCode` = 0x00 (see 13.4.6.6.4), when the ONU is ready to accept the forthcoming software image; or

b) *eOAM_Software_FileTransferAck* eOAMPDU, with `BlockNumber` = 0x00-00 and a specific value in the `ResponseCode` field, indicating the type and additional description of the encountered error (see Table 13-19). In this case, the software image download process is interrupted and may be reinitialized in the future, if needed.

13.5.1.2 Download step

Once the *eOAM_Software_FileTransferAck* eOAMPDU with `BlockNumber` = 0x00-00 and `ResponseCode` = 0x00 is received by the OLT, the OLT starts transmission of individual blocks of the software image using the series of *eOAM_Software_FileTransferData* eOAMPDUs (as specified in 13.4.6.6.3).

For the transfer, the software image is divided into a number of data blocks, where each block is at most 1400 octets long and the last block may be smaller than 1400 octets. Each of the transmitted software image blocks is accompanied by a sequentially increasing block number carried in the `BlockNumber` field in the *eOAM_Software_FileTransferData* eOAMPDU. This block number is used in the acknowledgment mechanism: each transmitted block is acknowledged by the ONU by sending the *eOAM_Software_FileTransferAck* eOAMPDU with the value in the `BlockNumber` field equal to the value carried in the `BlockNumber` field in the *eOAM_Software_FileTransferData* eOAMPDU plus one, indicating the number of the next software image block expected by the ONU. The OLT sends the next software image block only after the next software image block was requested by the ONU.

Once the file transfer begins, the OLT sends at least one *eOAM_Software_FileTransferData* eOAMPDU every second. If the OLT does not receive an *eOAM_Software_FileTransferAck* eOAMPDU from the ONU requesting the next block within one second of sending the last block, the OLT sends a keep-alive message (*eOAM_Software_FileTransferData* eOAMPDU with the `BlockWidth` equal to 0x00). Upon sending three keep-alive messages, the OLT aborts the software download process.

If the ONU fails to receive an *eOAM_Software_FileTransferAck* eOAMPDU every second, a timeout is counted, and the ONU sends an *eOAM_Software_FileTransferAck* eOAMPDU. This message contains the timeout error code and the sequence number indicating the desired block of the software image. Upon detecting three successive timeouts, the ONU aborts the software download process.

If the software image downloading process is aborted by either the OLT or the ONU due to three successive timeouts or other reasons, the ONU shall retain the software image that existed in the ONU prior to the failed download attempt.

13.5.1.3 Verification step

Once the OLT receives the *eOAM_Software_FileTransferAck* eOAMPDU confirming the successful reception of the last software image block, the OLT sends the *eOAM_Software_FileTransferAck* eOAMPDU with `BlockNumber` = 0x00-00 and `ResponseCode` = 0x00, requesting the ONU to verify that the software image was received, assembled, and stored correctly in the internal ONU storage.

The verification step uses the software image ICS. The downloaded software image contains an embedded ICS value (its location relative to the beginning of the software image is outside the scope of this specification). The ONU calculates the ICS for the downloaded software image and compares it with the ICS embedded in the software image. Other methods of software image verification are also allowed, but remain outside the scope of this standard.

If the verification was successful, the `ResponseCode` field holds the value of 0x00. In this case, the software download process is complete, and the ONU automatically starts the software image committing step. Otherwise, the ONU responds to the OLT with the *eOAM_Software_FileTransferAck* eOAMPDU (see 13.4.6.6.4) with the appropriate value of the `ResponseCode` field (per Table 13-19).

13.5.1.4 Committing step

The software image committing step is used to make the newly downloaded software image a default boot image from the next ONU restart onward. The OLT does not provide additional signaling for the ONU to start the software image committing process.

The software image committing step starts automatically once the ONU successfully verifies the received software image. This step involves writing a new software image into the permanent memory on the ONU and verifying the integrity of the written software image.

Once the software image committing step has successfully completed, the ONU sends the *eOAM_Software_FileTransferAck* eOAMPDU with `BlockNumber` = 0x00-00 and `ResponseCode` = 0x00, indicating the success of the software image committing step. If the committing process is successful, the ONU uses the newly committed software image on next restart of the device. ONU restart is not performed automatically as part of the committing step.

In the case of any errors detected during the process of committing the newly downloaded software image, the ONU sends the *eOAM_Software_FileTransferAck* eOAMPDU with `BlockNumber` = 0x00-00 and `ResponseCode` field that holds any of the values specified in Table 13-19, indicating a problem with the software image committing step. If the ONU is unable to complete the committing step (due to power interruption or other reasons) or if the integrity of the image written to the permanent storage is not confirmed, the ONU shall retain the previous version of the software image.

When the OLT receives the *eOAM_Software_FileTransferAck* eOAMPDU with `BlockNumber` = 0x00-00 and `ResponseCode` = 0x00, confirming the successful committal of the software image, it issues the *eOAM_Set_Request* eOAMPDU (see 13.4.6.4) with the *ONU Reboot* TLV (0xDD/0x00-01), as defined in 14.6.1.1, instructing the ONU to restart. The ONU loads the new software image as part of the restart process.

13.5.2 State diagrams

This subclause specifies the state diagrams for the software download process for the ONU and the OLT.

The software image download process on the OLT side is driven by the NMS. The OLT returns the completion result of individual steps to the NMS using the appropriate NMSI type primitives, as defined in 13.5.2.5.

13.5.2.1 Constants

`receiveTimeout`

TYPE: 16-bit unsigned integer

This constant represents the duration of the time interval between reception of subsequent messages at the given ONU. This constant is expressed in units of milliseconds.

VALUE: 0x03-E8 (1 second)

`retryLimit`

TYPE: 8-bit unsigned integer

This constant represents the maximum number of retransmission attempts for a single message. Once the `retryLimit` transmission attempts fail, the given device reacts per Figure 13-5 for the ONU and Figure 13-6 for the OLT.

VALUE: 3

`storeTimeout`

TYPE: 16-bit unsigned integer

This constant represents the duration of the time interval between subsequent reattempts of the software image committing process for the downloaded software image. This constant is expressed in units of milliseconds.

VALUE: 0x3A-98 (15 seconds)

`transmitTimeout`

TYPE: 16-bit unsigned integer

This constant represents the duration of the time interval between subsequent retransmissions of the same software image block encapsulated in the *eOAM_Software_FileTransferData* eOAMPDU to the given ONU. This constant is expressed in units of milliseconds.

VALUE: 0x03-E8 (1 second)

13.5.2.2 Variables

`blockData`

TYPE: bit array

This bit array contains the software image fragment (block) carried in the *eOAM_Software_FileTransferData* eOAMPDU. The size of the `blockData` bit array is derived from the `BlockWidth` field of this eOAMPDU.

`blockNumber`

TYPE: 16-bit unsigned integer

This variable identifies the software image block number in the sequence of transmission.

`blockSize`

TYPE: 16-bit unsigned integer

This variable identifies the maximum size of a single software image block that may be delivered to the ONU. This variable is set locally by the OLT prior to starting the software download process.

`blockWidth`

TYPE: 16-bit unsigned integer

This variable represents the size of the software image block, as extracted from the *eOAM_Software_FileTransferData* eOAMPDU.

`commitDone`

TYPE: Boolean

The value of `true` indicates that the software image committing process successfully completed operations in the COMMIT_IMAGE or ACK_BUSY states. Otherwise, the value of `commitDone` is set to `false`.

`fileName`

TYPE: null-terminated ASCII string

This variable represents the ONU software filename, as indicated by the NMS.

`imageSize`

TYPE: 32-bit unsigned integer

This variable represents the size of the software image, expressed in units of octets.

`lastBlock`

TYPE: 16-bit unsigned integer

This variable represents the index (sequence number) of the last software image block in the software image received from the NMS.

`localTimeoutCount`

TYPE: 8-bit unsigned integer

This variable counts the number of local timeout events observed by the OLT during the software download and committing process for a single message, as shown in Figure 13-6.

`nextBlock`

TYPE: 16-bit unsigned integer

This variable represents the index (sequence number) of the software image block that the ONU expects next. The first image block has an index of 0x00-00.

`remoteTimeoutCount`

TYPE: 8-bit unsigned integer

This variable counts the number of remote timeout events observed by the OLT during the software download and committing process for a single message, as shown in Figure 13-6.

`resultCode`

TYPE: 16-bit unsigned integer

This variable represents the numeric error code for the given error event as returned by the function.

`retryCount`

TYPE: 8-bit unsigned integer

This variable represents the current number of retransmission attempts for the given message.

`storageDone`

TYPE: Boolean

The value of `true` indicates that the `verifyStorage()` function in the ONU has completed preparing storage for the new software image. Otherwise, the value of `storageDone` is set to `false`.

13.5.2.3 Timers

`retryTimer`

This timer is used to measure the time interval between reception of subsequent messages from the ONU or the OLT. If three consecutive timeouts are announced, the given part of the software download process is aborted.

13.5.2.4 Functions

`clearStorage(newImage)`

This function deletes the partially downloaded image when the download process is aborted.

`commitImage(newImage)`

This function is used to write a new software image into the permanent memory on the ONU and verify the integrity of the written software image. On completion, this function sets the `commitDone` variable to `true`. This function returns the values as defined in Table 13-19.

`getBlock(image, block#)`

This function extracts the software image block number `block#` from the software image pointed to by `image` and returns it in the form of a bit array saved into the `blockData` variable. The retrieved software image block is then delivered to the ONU.

`verifyImage(imageId)`

This function verifies the integrity of the downloaded and assembled software image, identified by the parameter `imageId`. This function verifies that the ICS calculated for the stored software image, identified by a parameter `imageId`, matches the ICS embedded in the software image itself. The location of the embedded ICS code in the downloaded software image is implementation dependent. In addition, this function may also check for implementation-dependent criteria, such as verification of correct image type or version, verification of correct product ID vendor ID, etc. This function returns the values as defined in Table 13-19.

`verifyStorage(storageId)`

This function prepares the memory storage identified by the parameter `storageId` and verifies that it is ready to receive a new software image. This function may perform flash memory erasure and other necessary operations. On completion, this function sets the `storageDone` variable to `true`. This function returns the values as defined in Table 13-19.

`write(image, block)`

This function is used to write a data fragment (block) passed in the `block` parameter into the selected software image, identified by the `image` parameter. This function returns the values as defined in Table 13-19.

13.5.2.5 Primitives

`eOAMI_Any`

This primitive represents the reception of any eOAMPDU related to the software download protocol (defined in 13.4.6.6). It replaces the following logical condition:

```
OPI(source_address, flags, code, Opcode) AND
code                    == 0xFE AND
Opcode                  == 0x09
```

`eOAMI_FTA(block, code)`

Acronym for *eOAM_Software_FileTransferAck* eOAMPDU, as defined in 13.4.6.6.4. It replaces the following logical condition:

```
OPI(source_address, flags, code, OUI_1904_4 | Opcode |
FileTransferOpcode | BlockNumber | ResponseCode) AND
code                    == 0xFE AND
Opcode                  == 0x09 AND
FileTransferOpcode      == 0x03 AND
BlockNumber             == block AND
ResponseCode            == code
```

`eOAMI_FTA_End`

Acronym for `eOAMI_FTA(0x00-00, OK)`. The value `OK` is defined in Table 13-19.

`eOAMI_FTA_Error`

Acronym for *eOAM_Software_FileTransferAck* eOAMPDU, as defined in 13.4.6.6.4. It replaces the following logical condition:

```
OPI(source_address, flags, code, OUI_1904_4 | Opcode |
FileTransferOpcode | BlockNumber | ResponseCode) AND
code                    == 0xFE AND
Opcode                  == 0x09 AND
FileTransferOpcode      == 0x03 AND
ResponseCode            != 0x00 AND
ResponseCode            != 0x08
```

`eOAMI_FTA_ErrorCommit`

Acronym for *eOAM_Software_FileTransferAck* eOAMPDU, as defined in 13.4.6.6.4. excluding eOAMPDUs carrying response codes 0x00 (`OK`), 0x08 (`Timeout`), and 0x09 (`Busy`). The values for the response codes are defined in Table 13-19. This acronym replaces the following logical condition:

```
eOAMI_FTA_Error AND ResponseCode != 0x09
```

```
eOAMI_FTA_OK
```

Acronym for *eOAM_Software_FileTransferAck* eOAMPDU, as defined in 13.4.6.6.4. It replaces the following logical condition:

```
OPI(source_address, flags, code, OUI_1904_4 | Opcode |
FileTransferOpcode | BlockNumber | ResponseCode) AND
code                    == 0xFE AND
Opcode                  == 0x09 AND
FileTransferOpcode      == 0x03 AND
ResponseCode            == 0x00
```

```
eOAMI_FTA_Timeout
```

Acronym for *eOAM_Software_FileTransferAck* eOAMPDU, as defined in 13.4.6.6.4. It replaces the following logical condition:

```
OPI(source_address, flags, code, OUI_1904_4 | Opcode |
FileTransferOpcode | BlockNumber | ResponseCode) AND
code                    == 0xFE AND
Opcode                  == 0x09 AND
FileTransferOpcode      == 0x03 AND
ResponseCode            == 0x08
```

```
eOAMI_FTD
```

Acronym for *eOAM_Software_FileTransferData* eOAMPDU, as defined in 13.4.6.6.3. It replaces the following logical condition:

```
OPI(source_address, flags, code, OUI_1904_4 | Opcode |
FileTransferOpcode) AND
code                    == 0xFE AND
Opcode                  == 0x09 AND
FileTransferOpcode      == 0x02
```

```
eOAMI_WR
```

Acronym for *eOAM_Software_WriteRequest* eOAMPDU, as defined in 13.4.6.6.2. It replaces the following logical condition:

```
OPI(source_address, flags, code, OUI_1904_4 | Opcode |
FileTransferOpcode | FileName) AND
code                    == 0xFE AND
Opcode                  == 0x09 AND
FileTransferOpcode      == 0x01
```

```
eOAMR_FTA(block, code)
```

Acronym for *eOAM_Software_FileTransferAck* eOAMPDU, as defined in 13.4.6.6.4. It replaces the following code:

```
code                    = 0xFE
Opcode                  = 0x09
FileTransferOpcode      = 0x03
BlockNumber             = block
```

```
ResponseCode            = code
source_address          = OLT or ONU MAC
OPR(source_address, flags, code, OUI_1904_4 | Opcode |
FileTransferOpcode | BlockNumber | ResponseCode)
```

`eOAMR_FTA_Code(code)`

Acronym for `eOAMR_FTA(0x00-00, code)`. The argument `code` represents an 8-bit unsigned integer that can take on values defined in Table 13-19.

`eOAMR_FTA_End`

Acronym for `eOAMR_FTA(0x00-00, 0x00)`.

`eOAMR_FTD(blockData, blockNumber)`

Acronym for *eOAM_Software_FileTransferData* eOAMPDU, as defined in 13.4.6.6.3. It replaces the following logical condition:

```
code                    = 0xFE
Opcode                  = 0x09
FileTransferOpcode      = 0x03
BlockNumber             = blockNumber
BlockWidth              = size of blockData parameter in units of octets
BlockData               = blockData
source_address          = OLT MAC
OPR(source_address, flags, code, OUI_1904_4 | Opcode |
FileTransferOpcode | BlockNumber | BlockWidth | BlockData)
```

`eOAMR_FTD_KeepAlive`

Acronym for *eOAM_Software_FileTransferData* eOAMPDU, as defined in 13.4.6.6.3. It replaces the following logical condition:

```
code                    = 0xFE
Opcode                  = 0x09
FileTransferOpcode      = 0x03
BlockNumber             = 0x00-00
BlockWidth              = 0x00
source_address          = OLT MAC
OPR(source_address, flags, code, OUI_1904_4 | Opcode |
FileTransferOpcode | BlockNumber | BlockWidth)
```

`eOAMR_Reboot`

Acronym for *eOAM_Set_Request* eOAMPDU, as defined in 13.4.6.4, containing *ONU Reboot* TLV (0xDD/0x00-01), as defined in 14.6.1.1. It replaces the following logical condition:

```
code                    = 0xFE
Opcode                  = 0x03
ONU_Reboot_TLV.Branch   = 0xDD
ONU_Reboot_TLV.Leaf     = 0x00-01
source_address          = OLT MAC
OPR(source_address, flags, code, OUI_1904_4 | Opcode |
ONU_Reboot_TLV)
```

`eOAMR_WR(fileName)`

Acronym for *eOAM_Software_WriteRequest* eOAMPDU, as defined in 13.4.6.6.2. It replaces the following code:

```
code                    = 0xFE
Opcode                  = 0x09
FileTransferOpcode      = 0x01
FileName                = fileName
source_address          = OLT MAC
OPR(source_address, flags, code, OUI_1904_4 | Opcode |
FileTransferOpcode | FileName)
```

`NMSI(commit, status)`

This primitive is used to notify the NMS about the result of the software image committing process, where the `status` parameter corresponds to the return code, as defined in Table 13-19.

When `status` is equal to `OK`, this primitive is used to notify the NMS about the successful completion of the software image committing process per Figure 13-6.

`NMSI(download, status)`

This primitive is used to notify the NMS about the result of the software image download process, where the `status` parameter corresponds to the return code, as defined in Table 13-19.

When `status` is equal to `OK`, this primitive is used to notify the NMS about the successful completion of the software image download process per Figure 13-6.

`NMSR(download, imageData, imageSize, fileName)`

This primitive is used by the NMS to request the OLT to start the software image download process per Figure 13-6, where the software image is delivered to the OLT within the `imageData` parameter and the size of the received image is provided in the `imageSize` parameter. The ONU software filename is provided in the `fileName` parameter.

13.5.2.6 State diagrams

The ONU shall implement the software image download process as shown in Figure 13-5.

The OLT shall implement the software image download process as shown in Figure 13-6. The state diagram defined in Figure 13-6 is instantiated for each ONU and is operated independently as requested by the NMS.

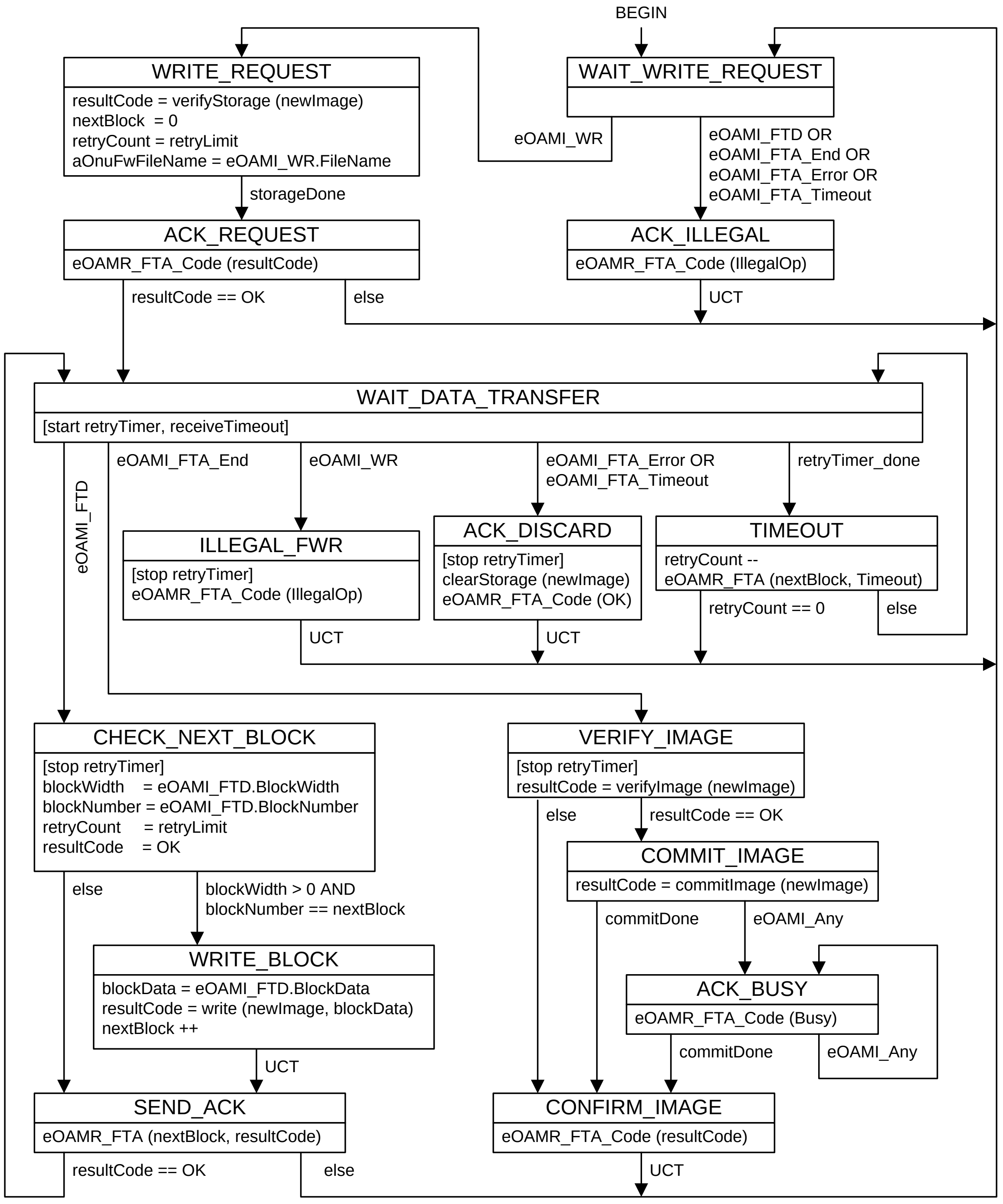

Figure 13-5—ONU software image download and committing process state diagram

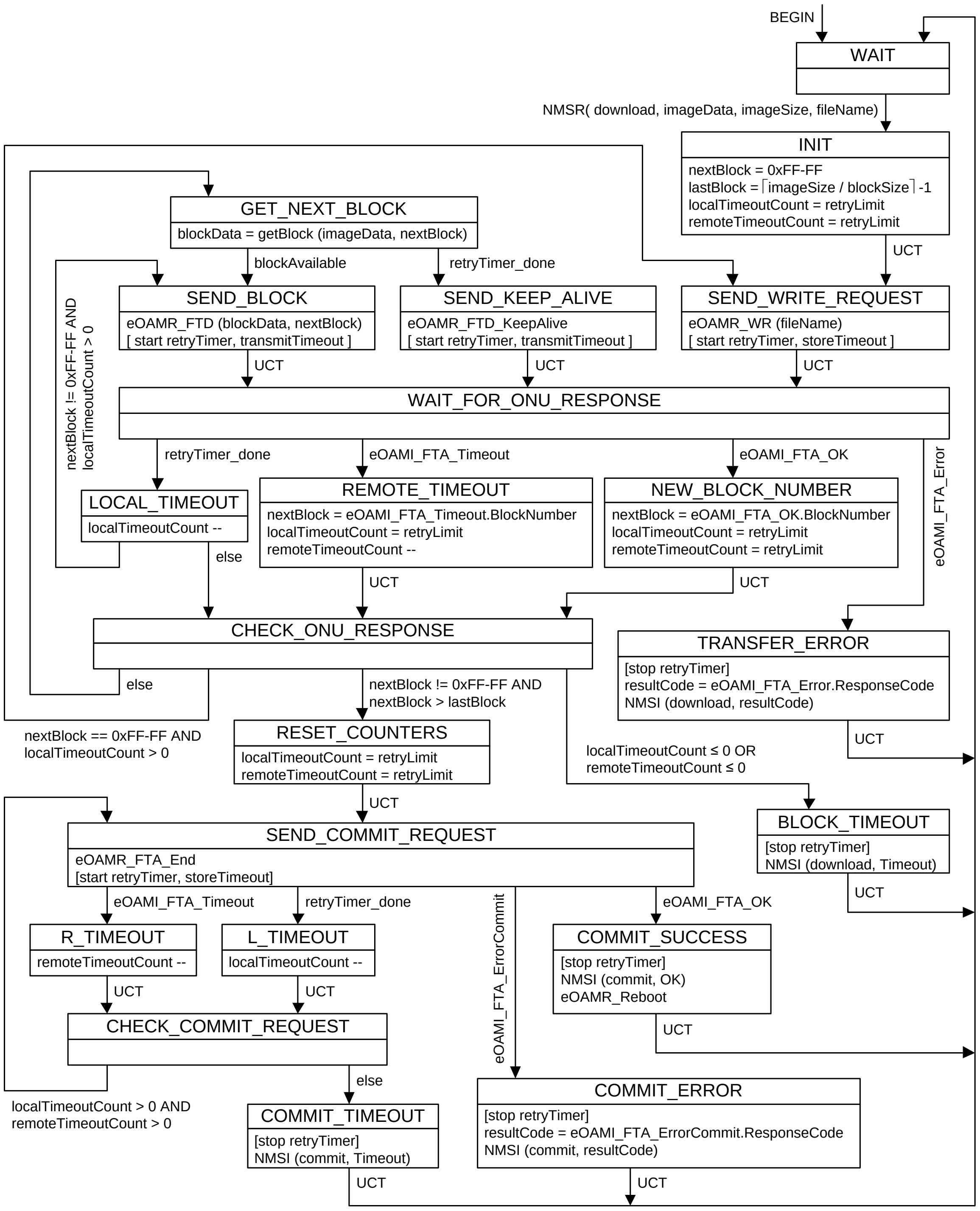

Figure 13-6—OLT software image download process state diagram