12 Discovery and maintenance

12.1 Introduction

Clause 12 focuses on ONU identity and capability discovery processes, together with maintenance mechanisms. The described maintenance mechanisms include the definition of a software upgrade process, which allows operators to remotely modify the software loaded on the ONU.

12.2 Device discovery and capability discovery

~~12.2.1 DPoE eOAM management~~

~~12.2.2~~12.2.1 MPCP/OAM discovery process

Figure 12-1~~Figure 12-8~~ 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~~64.3.3 for 1G-EPON and 77.3.3 for 10G-EPON~~. 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.

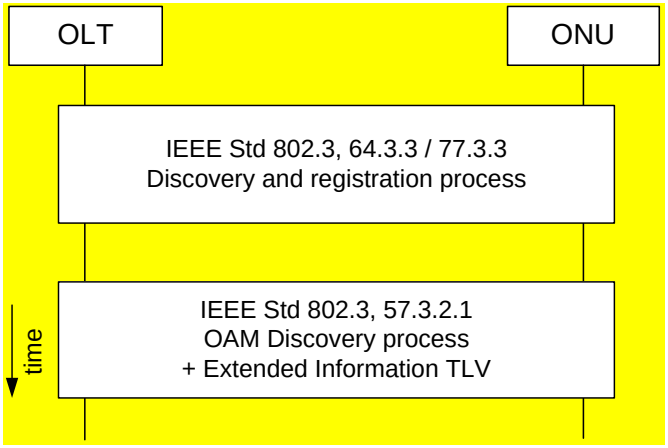

Figure 12-1—MPCP/OAM discovery process

~~12.2.3~~12.2.2 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.

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

~~12.2.3.1~~12.2.2.1 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 12.2~~12.2.3~~, 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 12.2~~12.2.3~~, 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 12.2~~12.2.3~~.

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 ~~13.4.1.3.1~~13.2.4.1, referred to as *Extended Information* TLV. The *Extended Infomration* 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 ~~13.4.1.3.1~~13.2.4.1. An ONU ~~complying with the requirements of this profile~~ shall include the *Extended Information* TLV in all *Information* OAMPDUs exchanged during the eOAM discovery process. An ONU ~~complying with the requirements of this profile~~ 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 12.2~~12.2.3~~, ~~13.4~~Clause 13, and Clause 14~~14.4~~. The lack of such an *Extended Information* TLV is treated as a lack of support for the requirements set forth in 12.2~~12.2.3~~, Clause 13~~13.4~~, and Clause 14~~14.4~~, and consequently the OLT deregisters such an ONU as indicated above.

~~12.2.3.2~~12.2.2.2 Ordering of *Organization Specific Information* TLVs

~~12.2.3.2.1~~12.2.2.2.1 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 13.2.4.1~~13.2.2.3.1~~ may be transmitted as the first *Organization Specific Information* TLV, followed by other *Organization Specific Information* TLVs, if present.

Formatted: Highlight

~~12.2.3.2.2~~12.2.2.2.2 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.

~~12.2.3.3~~12.2.2.3 Message flow during eOAM discovery process

The message flow during the eOAM discovery process ~~for this profile~~ is very simple. After the MPCP discovery and registration process is successfully completed, an ONU ~~complying with the requirements of this profile~~ starts sending periodically *Information* OAMPDUs carrying the *Extended Information* TLV as defined in ~~13.4.1.3.1~~13.2.4.1. Such an *Extended Information* TLV indicates the support of 12.2~~12.2.3~~, Clause 13~~13.4~~, and Clause 14~~14.4~~. 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 eOAM version negotiation phase, such as the one described in 12.2.1, is not used in this profile, and t~~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.

~~12.2.4~~12.2.3 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 12.2.2~~12.2.1.2~~, may carry the *Extended Information* TLV or any other *Organization Specific Information* TLVs. An ONU ~~complying with the requirements of this profile~~ 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~~Clause 144 for 25G-EPON 50G-EPON~~. 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~~Clause 144 for 25G-EPON 50G-EPON~~.

12.3 Software updates

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.)

~~12.3.1 Software upgrade using DPoE eOAM management~~

This subclause describes the software upgrade process implemented using the extended OAM specified in Clause 13~~13.4~~ and Clause 14~~14.4~~.

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.

~~12.3.2~~12.3.1 Software image download process

The software image download process is outlined in Figure 12-2~~Figure 12-20~~ and is further defined in the state diagram shown in Figure 12-3~~Figure 12-21~~ for the ONU and in the state diagram shown in Figure 12-4~~Figure 12-22~~ for the OLT.

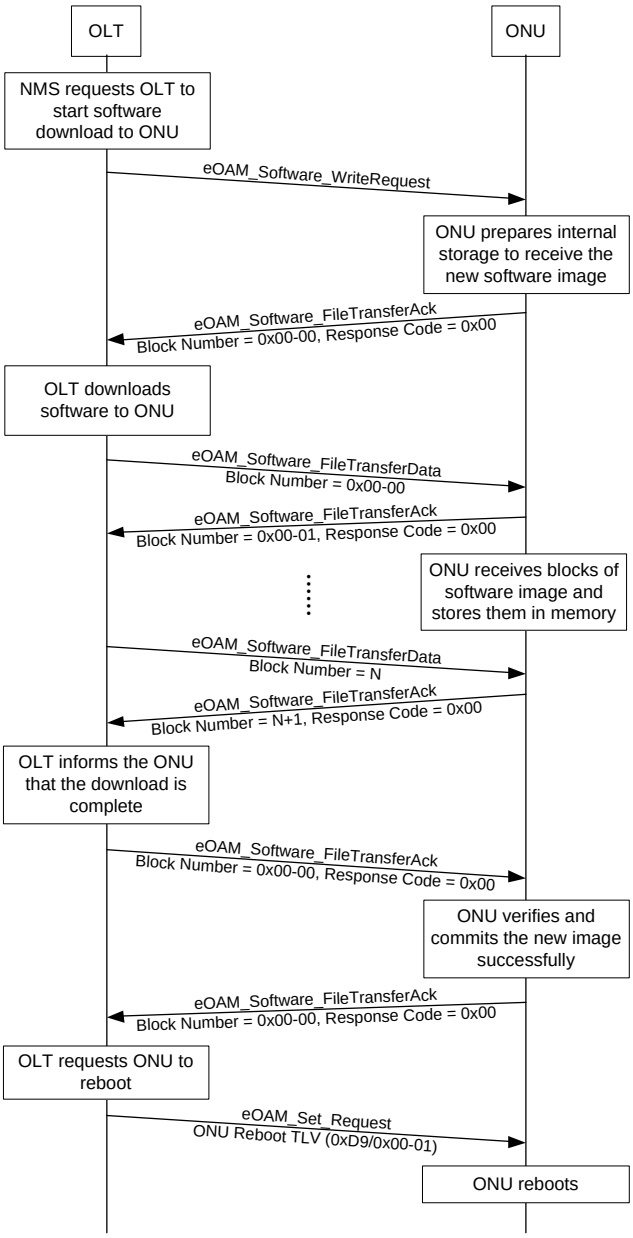

Figure 12-2—Data flow during a successful software image download and committing process

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 12-2~~Figure 12-20~~.

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 ~~13.4.2.10~~13.3.6.

— 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

~~12.3.2.1~~12.3.1.1 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 13.3.6.2), 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 13.3.6.4), 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 13-101~~Table 13-18). In this case, the software image download process is interrupted and may be reinitialized in the future, if needed.

~~12.3.2.2~~12.3.1.2 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 13.3.6.3).

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.

~~12.3.2.3~~12.3.1.3 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 13.3.6.4) with the appropriate value of the `ResponseCode` field (per ~~Table 13-101~~Table 13-18).

~~12.3.2.4~~12.3.1.4 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 13-101~~Table 13-18, 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 ~~13.4.2.4~~13.3.4) with the *ONU Reboot* TLV (0xDD/0x00-01), as defined in ~~14.4.5.1.1~~14.6.1.1, instructing the ONU to restart. The ONU loads the new software image as part of the restart process.

~~12.3.3~~12.3.2 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 12.3.2.5~~12.3.3.2.5~~.

~~12.3.3.1~~12.3.2.1 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 12-3~~Figure 12-21~~ for the ONU and Figure 12-4~~Figure 12-22~~ 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)

~~12.3.3.2~~12.3.2.2 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 12-4~~Figure 12-22~~.

`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 12-4~~Figure 12-22~~.

`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`.

~~12.3.3.3~~12.3.2.3 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.

~~12.3.3.4~~12.3.2.4 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 13-101~~Table 13-18.

`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 13-101~~Table 13-18.

`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 13-101~~Table 13-18.

`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 13-101~~Table 13-18.

~~12.3.3.5~~12.3.2.5 Primitives

`eOAMI_Any`

This primitive represents the reception of any eOAMPDU related to the software download protocol (defined in ~~13.4.2.10~~13.3.6). 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 13.3.6.4. 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 13-101~~Table 13-18.

`eOAMI_FTA_Error`

Acronym for *eOAM_Software_FileTransferAck* eOAMPDU, as defined in 13.3.6.4. 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 13.3.6.4. excluding eOAMPDUs carrying response codes 0x00 (`OK`), 0x08 (`Timeout`), and 0x09 (`Busy`). The values for the response codes are defined in ~~Table 13-101~~Table 13-18. 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 13.3.6.4. 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 13.3.6.4. 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 13.3.6.3. 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 13.3.6.2. 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 13.3.6.4. 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 13-101~~Table 13-18.

`eOAMR_FTA_End`

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

`eOAMR_FTD(blockData, blockNumber)`

Acronym for *eOAM_Software_FileTransferData* eOAMPDU, as defined in 13.3.6.3. 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 13.3.6.3. 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 ~~13.4.2.4~~13.3.4, containing *ONU Reboot* TLV (0xDD/0x00-01), as defined in ~~14.4.5.1.1~~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 13.3.6.2. 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 13-101~~Table 13-18.

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 12-4~~Figure 12-22~~.

```
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 13-101~~Table 13-18.

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 12-4~~Figure 12-22~~.

```
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 12-4~~Figure 12-22~~, 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.

~~12.3.3.6~~12.3.2.6 State diagrams

The ONU shall implement the software image download process as shown in Figure 12-3~~Figure 12-21~~.

The OLT shall implement the software image download process as shown in Figure 12-4~~Figure 12-22~~. The state diagram defined in Figure 12-4~~Figure 12-22~~ is instantiated for each ONU and is operated independently as requested by the NMS.

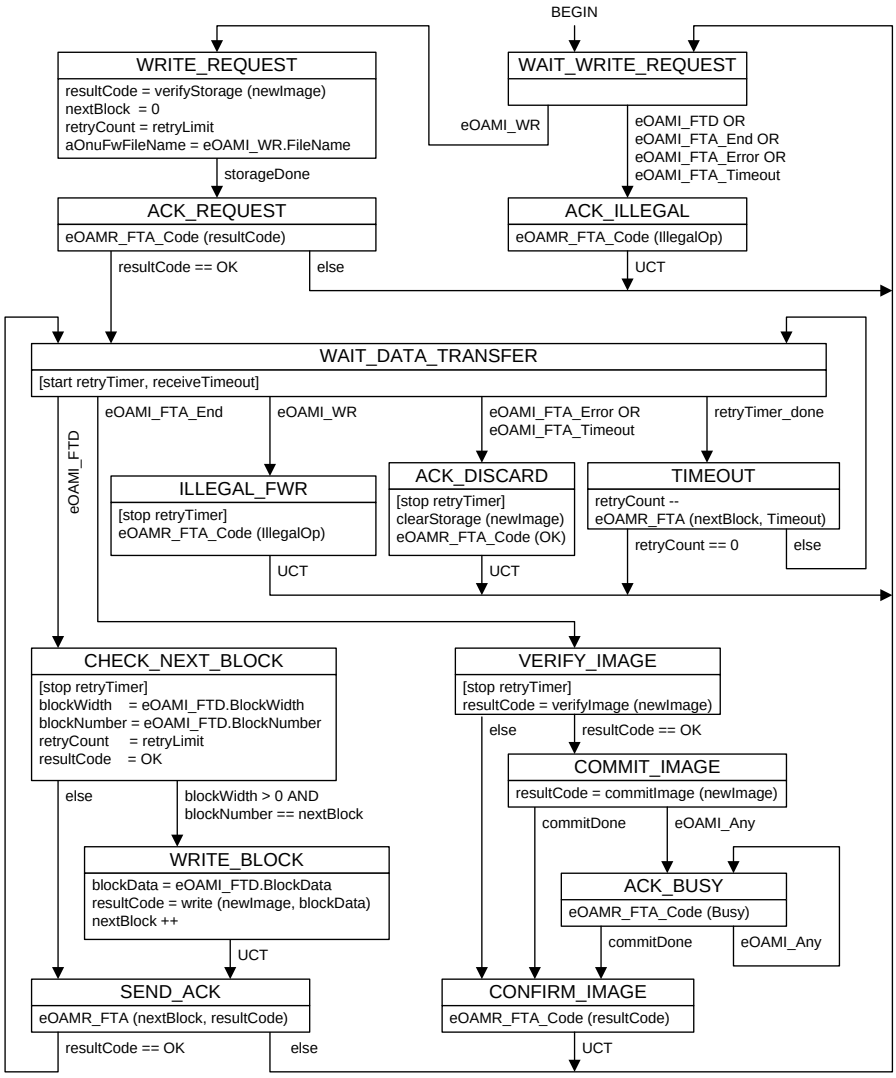

Figure 12-3—ONU software image download and committing process state diagram

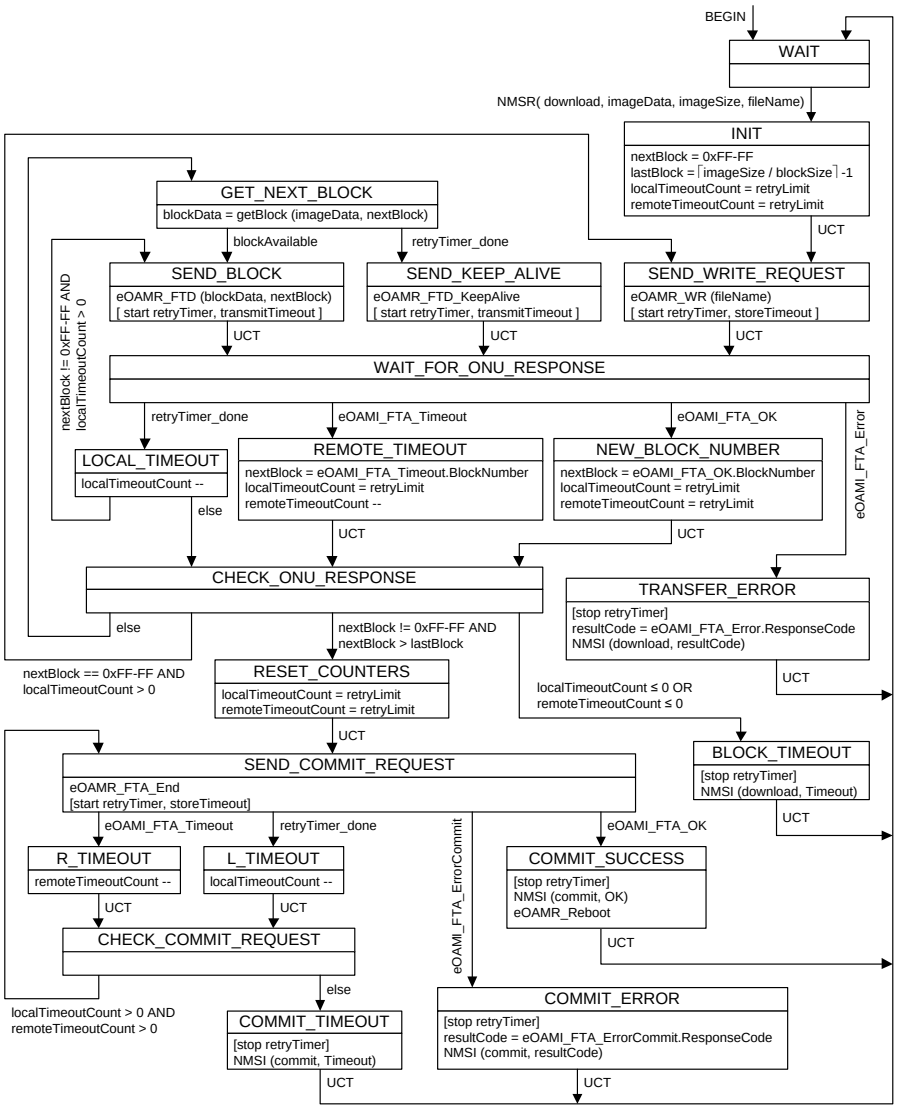

Figure 12-4—OLT software image download process state diagram