USB Power Delivery ENGINEERING CHANGE NOTICE

Title: Chunking Timing Issue

Applied to: USB Power Delivery Specification Revision 3.1 Version 1.3

Brief description of the functional changes proposed:
Add a mechanism to allow adaptive adjustment to tSenderResponse so that any length Extended Message responses can be accommodated without timing out. Increase the tSenderResponseTimer to 26ms to 32ms. Correct the text description of SenderResponseTimer to match the description provided by state diagram in Figure 6-65 Protocol layer Message reception.

Benefits as a result of the proposed changes:
Removes a potential cause of device incompatibility.

An assessment of the impact to the existing revision and systems that currently conform to the USB specification:
Removes a potential cause of device incompatibility.

An analysis of the hardware implications:

An analysis of the software implications:

An analysis of the compliance testing implications:

Increasing the tSenderResponseTimer to 26ms to 32ms.

The change will not affect existing products as they will not require the new timing, and will not affect current designs not yet released as product for the same reason. If all new designs make use of the new timing, then there will be no issues in the future. Any design that uses an actual tSenderResponseTimer of between 26 and 30ms will not require modification.

In compliance, only designs with tSenderResponseTimer between 24 and 26ms would need a waiver if they do not expect to request full length unchunked extended messages. There should be no reason not to grant such a waiver. The waiver would be a signal to the vendor to increase the timer for future designs.

Description of the Problem

Unchunked Non-Extended Message Situation

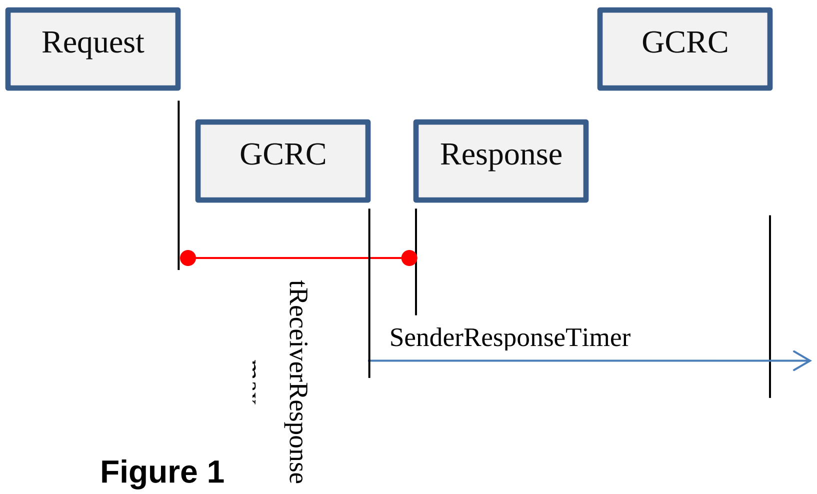


Figure 1

When a device sends a request message (start of AMS) it starts a timer (Sender Response Timer) and expects to have completely received a response message and have sent a GoodCRC within that time. The time (24 to 30ms) was chosen before Chunked Extended Messages were added.

The Policy Engine is responsible for maintaining this timer, and the Protocol Engine sends a notification, via the Chunking Layer, that the response message has arrived.

In the description that follows, some liberties have been taken for timers with the same value but different purposes, as they are officially named differently, and this causes problems in describing combinations of the timers. This was simply done for convenience here.

Problems arise in the situation where a request is sent and the response is an Extended Message with more than one chunk. Depending on the speed with which the messages are sent, it is possible that the Sender Response Timer will time out before the complete chunked message has been received and a Soft Reset will be issued.

So far this issue has been avoided because the only Extended Messages with more than one chunk has been Firmware Update and Security related and these were defined as not being AMSs and therefore not involving the SenderResponseTimer.

Now however we have EPR_Get_Sink_Caps and EPR_Get_Source_Caps which involve 2 chunk response messages plus a Chunk Request message, and **are** AMSs.

Chunked Situation

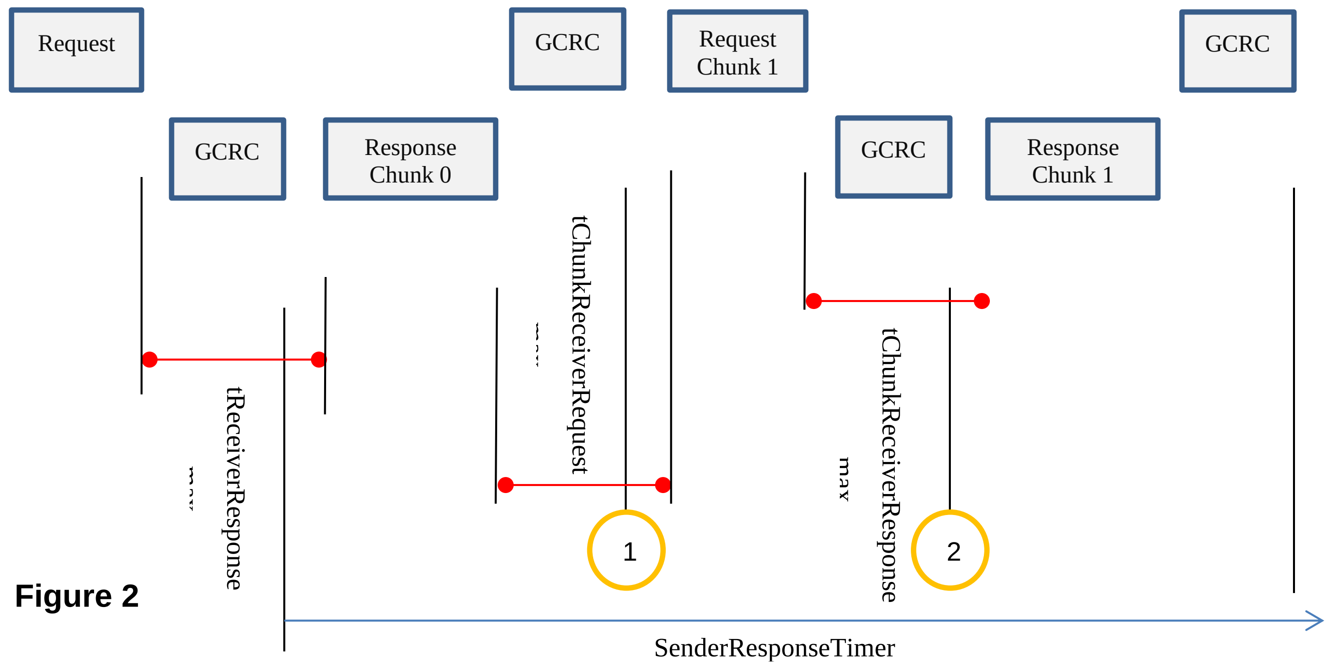


Figure 2

Taking as an example an EPR_Get_Sink_Caps request, the expected response is a two chunk EPR_Sink_Caps Extended Message. This involves the transfer of three message/GoodCRC pairs each of which is required to be completed within a time equivalent to the Sender Response Timer, so the total response time could be at least 3 x tReceiverResponse max or >45ms. If it takes this long then the Sender Response Timer (max 30ms) will time out and a Soft Reset will be issued.

Such a situation cannot be permitted to continue. We need to implement a solution that prevents timing out on Extended Messages with more than one chunk.

Ideally we need to set a timer that is larger when there are more chunks. The problem is that in the general case we may not know what length response to expect at the time we need to start the timer.

Proposed Solution

The proposed solution is to use a mechanism to extend tSenderResponse as chunks continue to be requested. This has the advantage of not changing the timeout value for any extended messages with only one chunk, and does not increase the time waited in the case of a breakdown in communication. It is a once and for all fix with very little extra logic.

This would involve the Chunking Engine sending a notification (SRT_Stop) at point 1 in Figure 2 to stop the SenderResponseTimer, and sending a notification (SRT_Start) at point 2 in Figure 2 to restart the SenderResponseTimer.

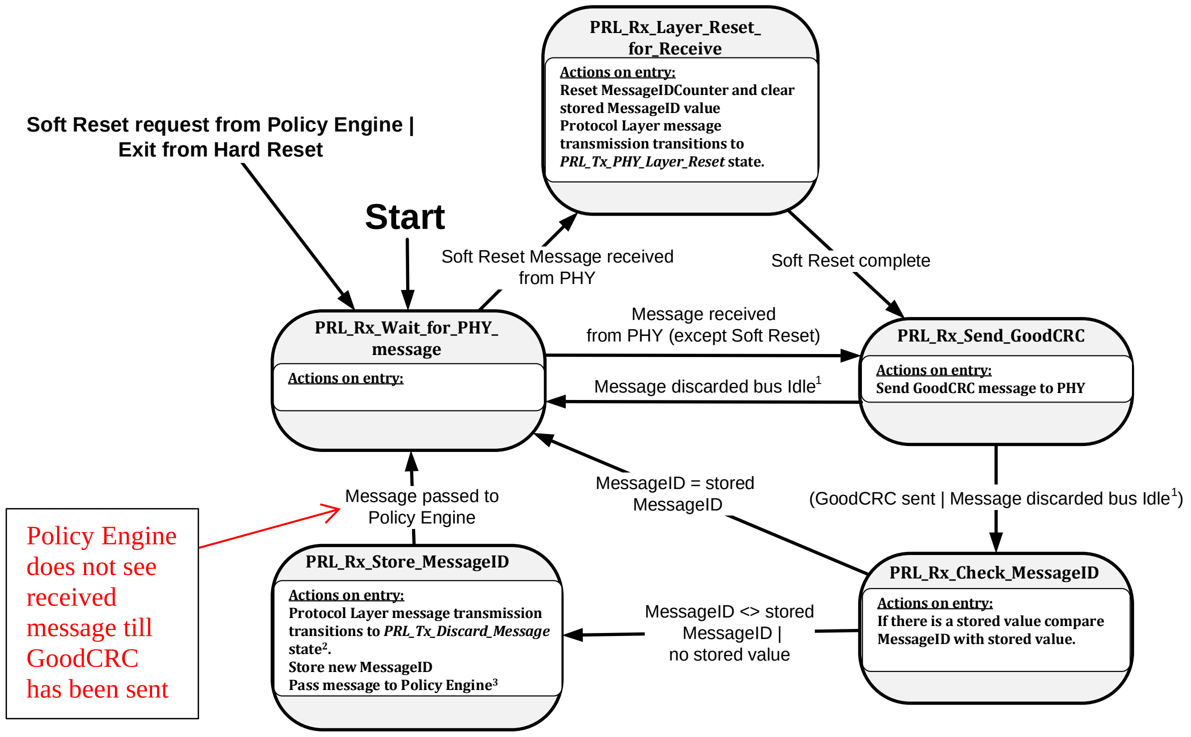
To enable the change to be done with the minimum of complication it is proposed to separate the SenderResponseTimer out from the Policy Engine, and describe it within its own state machine. This state machine can then be stopped and started from the Policy Engine and also the Chunking Layer.

Finally it was noticed that the SenderResponseTimer text description has never matched the state diagram implementation as Figure 6-65 shows that the received message is not passed to the Policy Engine until the GoodCRC has been completely sent. This should not effect implementations, that are assumed to follow the state diagram. So it is proposed to bring the text description into line with the state diagrams.

6.12.2.3 Protocol Layer Message Reception

Figure 6-65 shows the state behavior for the Protocol Layer when receiving a Message.

Figure 6-65 Protocol layer Message reception


PRL_Rx_Layer_Reset_for_Receive
Actions on entry:
Reset MessageIDCounter and clear stored MessageID value
Protocol Layer message transmission transitions to PRL_Tx_PHY_Layer_Reset state.
Soft Reset request from Policy Engine | Exit from Hard Reset
Start
Soft Reset Message received from PHY
Soft Reset complete
PRL_Rx_Wait_for_PHY_message
Actions on entry:
Message received from PHY (except Soft Reset)
Message discarded bus Idle¹
PRL_Rx_Send_GoodCRC
Actions on entry:
Send GoodCRC message to PHY
MessageID = stored MessageID
Message passed to Policy Engine
(GoodCRC sent | Message discarded bus Idle¹)
Policy Engine does not see received message till GoodCRC has been sent
PRL_Rx_Store_MessageID
Actions on entry:
Protocol Layer message transmission transitions to PRL_Tx_Discard_Message state².
Store new MessageID
Pass message to Policy Engine³
MessageID <> stored MessageID | no stored value
PRL_Rx_Check_MessageID
Actions on entry:
If there is a stored value compare MessageID with stored value.

Unchunked Extended Message Situation

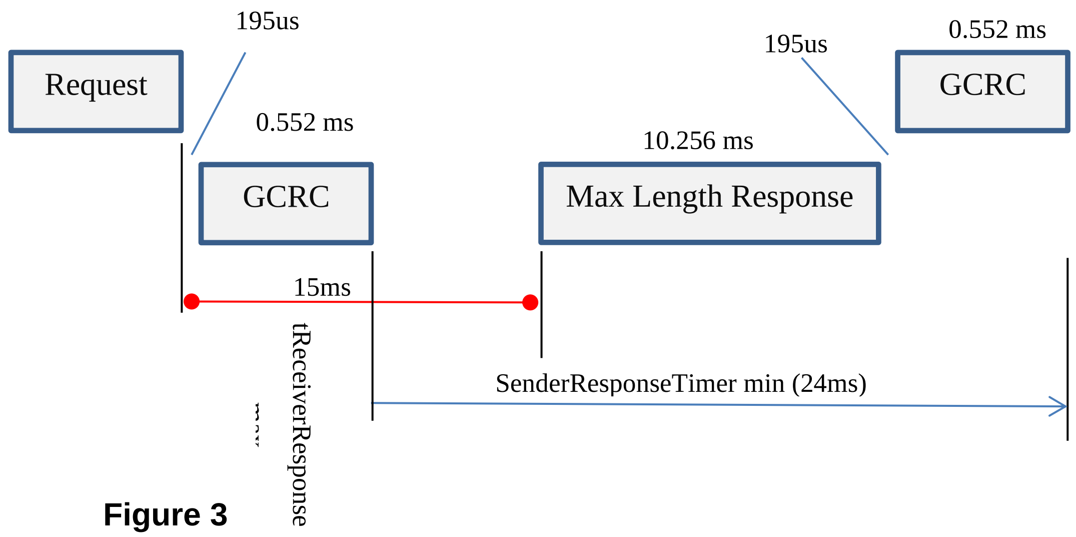


Figure 3

Duration of Longest Unchunked Extended Message

• Preamble	64 bits
• Start of Packet	20 bits
• Message Header	20 bits
• Extended Message Header	20 bits
• Data bits	260*10 = 2600 bits
• CRC	40 bits
• End of Packet	5 bits

Total bits = 2769

Duration of bit at 270kb/s = 3.704e-6

Duration of message = 10.256ms

Duration of Longest GoodCRC Message

• Preamble	64 bits
• Start of Packet	20 bits
• Message Header	20 bits
• CRC	40 bits
• End of Packet	5 bits

Total bits = 149

Duration of bit at 270kb/s = 3.704e-6

Duration of message = 0.552 ms

15-.552-.195 = 14.253ms

.552+10.256+.195 = 11.003ms

= 25.265ms

24ms

So, longest time for message to arrive is 25.265ms and SenderResponseTimer can expire in 24ms.

The simplest fix for this is to increase the minimum tSenderResponseTimer to 26ms, and either leave the maximum tSenderResponseTimer at 30ms or increase it to 32ms.

The change will not affect existing products as they will not require the new timing, and will not affect current designs not yet released as product for the same reason. If all new designs make use of the new timing then there will be no issues in the future. Any design that uses a tSenderResponseTimer of between 26 and 30ms will not require modification.

In compliance only designs with tSenderResponseTimer between 24 and 26ms would need a waiver if they do not expect to request full length unchunked extended messages. A waiver would be a signal to the vendor to increase the timer for future designs.

Actual Change Requested

(a). Section 6.6.2 “SenderResponseTimer”, Page 227

From Text:

The ***SenderResponseTimer*** ***Shall*** be used by the sender’s Policy Engine to ensure that a Message requesting a response (e.g. ***Get_Source_Cap*** Message) is responded to within a bounded time of ***tSenderResponse***. Failure to receive the expected response is detected when the ***SenderResponseTimer*** expires.

The Policy Engine’s response when the ***SenderResponseTimer*** expires ***Shall*** be dependent on the Message sent (see Section 8.3).

The ***SenderResponseTimer*** ***Shall*** be started from the time the last bit of the ***GoodCRC*** Message ***EOP*** (i.e. the ***GoodCRC*** Message corresponding to the Message requesting a response) has been received by the Physical Layer. The ***SenderResponseTimer*** ***Shall*** be stopped when the last bit of the expected response Message ***EOP*** has been received by the Physical Layer.

The receiver of a Message requiring a response ***Shall*** respond within ***tReceiverResponse*** in order to ensure that the sender’s ***SenderResponseTimer*** does not expire.

The ***tReceiverResponse*** time ***Shall*** be measured from the time the last bit of the Message ***EOP*** has been received by the Physical Layer until the first bit of the response Message Preamble has been transmitted by the Physical Layer.

To Text:

The ***SenderResponseTimer*** ***Shall*** be used by the sender’s Policy Engine to ensure that a Message requesting a response (e.g. ***Get_Source_Cap*** Message) is responded to within a bounded time of ***tSenderResponse***. Failure to receive the expected response is detected when the ***SenderResponseTimer*** expires.

For Extended Messages received as Chunks, the SenderResponseTimer will also be started and stopped by the Chunking Rx State Machine. See Section 8.3.3.1.1 for more details of the SenderResponseTimer operation.

The Policy Engine’s response when the ***SenderResponseTimer*** expires ***Shall*** be dependent on the Message sent (see Section 8.3).

The ***SenderResponseTimer*** ***Shall*** be started from the time the last bit of the ***GoodCRC*** Message ***EOP*** (i.e. the ***GoodCRC*** Message corresponding to the Message requesting a response) has been received by the Physical Layer. The ***SenderResponseTimer*** ***Shall*** be stopped when the last bit of the expected response Message ***EOP*** has been received by the Physical Layer.

The receiver of a Message requiring a response ***Shall*** respond within ***tReceiverResponse*** in order to ensure that the sender’s ***SenderResponseTimer*** does not expire.

The ***tReceiverResponse*** time ***Shall*** be measured from the time the last bit of the the ***GoodCRC*** Message ***EOP***, corresponding to the expected response Message, has been received by the Physical Layer until the first bit of the response Message Preamble has been transmitted by the Physical Layer.

(b). Table 6-68 “Time Values”, Page 240

From Text:

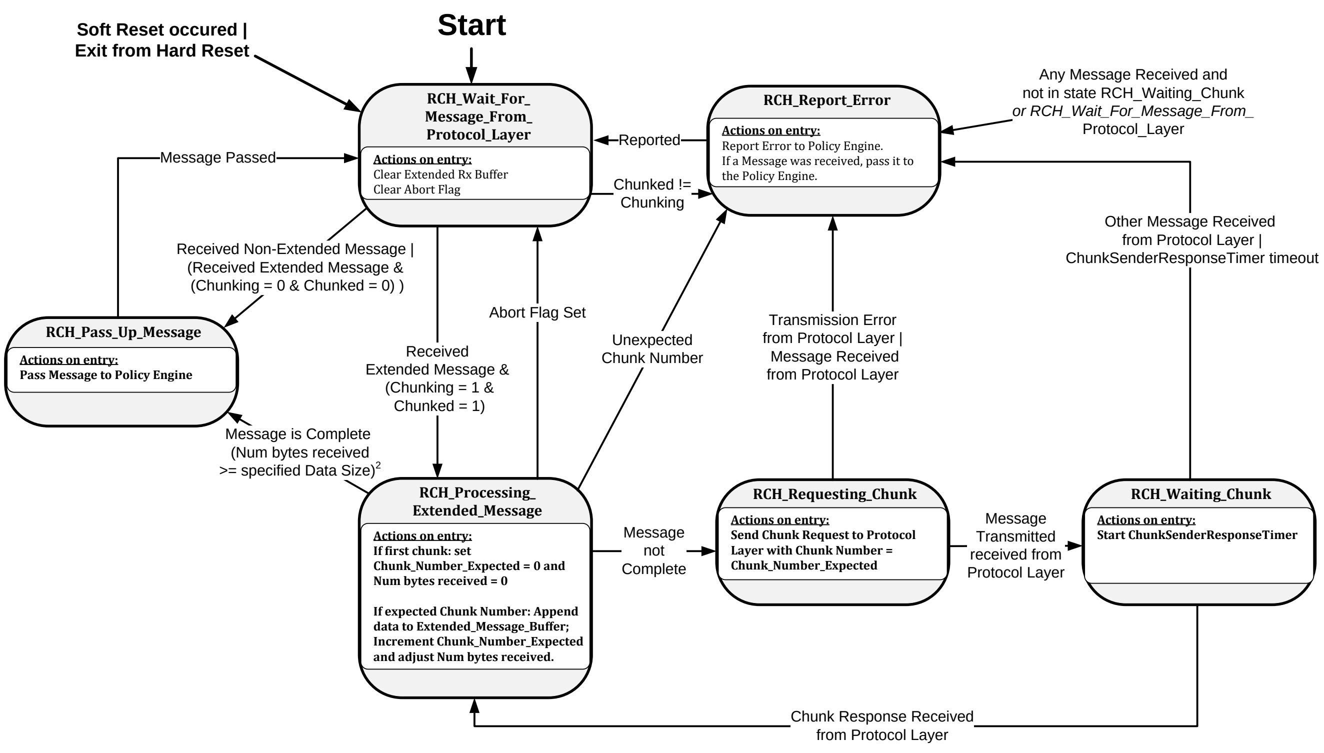
tSenderResponse	**24**	**27**	**30**	**ms**	**Section 6.6.2**

To Text:

tSenderResponse	26	29	32	ms	Section 6.6.2

(c). Figure 6-59 “Chunked Rx State Diagram”, Page 252

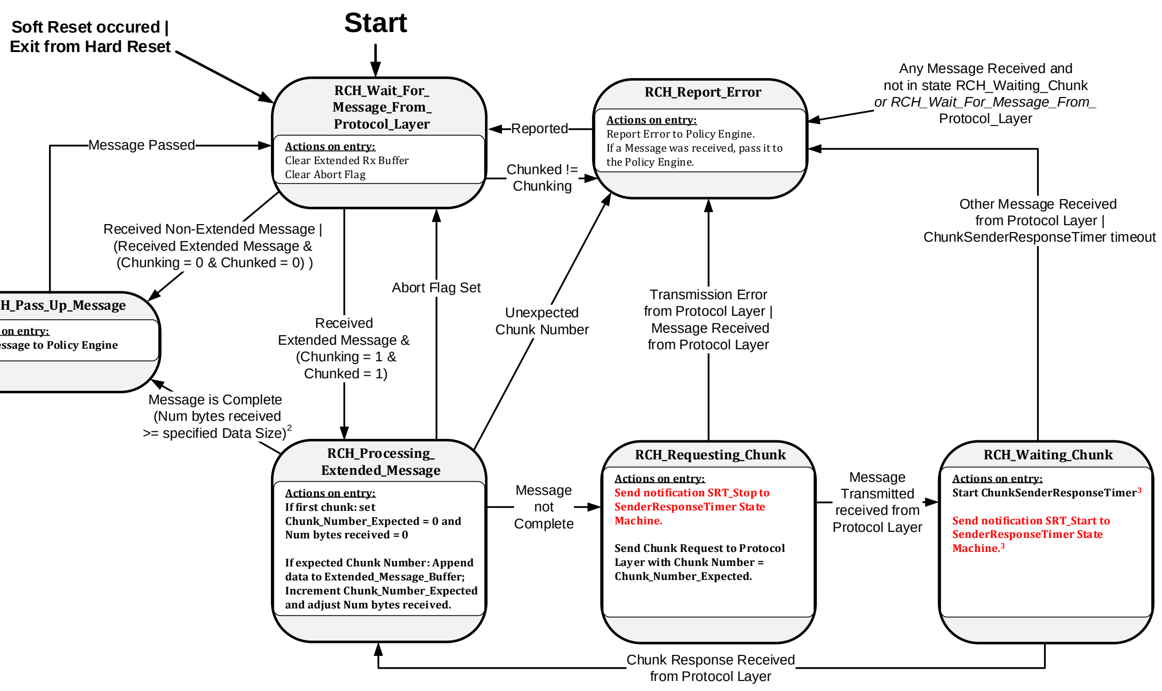
From Text:


Start
Soft Reset occured | Exit from Hard Reset
RCH_Wait_For_ Message_From_ Protocol_Layer
Actions on entry:
Clear Extended Rx Buffer
Clear Abort Flag
RCH_Report_Error
Actions on entry:
Report Error to Policy Engine.
If a Message was received, pass it to the Policy Engine.
Reported
Chunked != Chunking
Any Message Received and not in state RCH_Waiting_Chunk or RCH_Wait_For_Message_From_ Protocol_Layer
Message Passed
Received Non-Extended Message | (Received Extended Message & (Chunking = 0 & Chunked = 0))
Other Message Received from Protocol Layer | ChunkSenderResponseTimer timeout
RCH_Pass_Up_Message
Actions on entry:
Pass Message to Policy Engine
Abort Flag Set
Received Extended Message & (Chunking = 1 & Chunked = 1)
Unexpected Chunk Number
Transmission Error from Protocol Layer | Message Received from Protocol Layer
Message is Complete (Num bytes received >= specified Data Size)2
RCH_Processing_ Extended_Message
Actions on entry:
If first chunk: set Chunk_Number_Expected = 0 and Num bytes received = 0
If expected Chunk Number: Append data to Extended_Message_Buffer; Increment Chunk_Number_Expected and adjust Num bytes received.
Message not Complete
RCH_Requesting_Chunk
Actions on entry:
Send Chunk Request to Protocol Layer with Chunk Number = Chunk_Number_Expected
Message Transmitted received from Protocol Layer
RCH_Waiting_Chunk
Actions on entry:
Start ChunkSenderResponseTimer
Chunk Response Received from Protocol Layer

[1] Chunking is an internal state that is set to 1 if the ‘Unchunked Extended Messages Supported’ bit in either Source Capabilities or Request is 0. It defaults to 1 and is set after the first exchange of Source Capabilities and Request. It is also set to 1 for ***SOP’*** or ***SOP’’*** communication.

[2] Additional bytes received over specified ***Data Size*** will be as a result of padding in the last chunk.

To Text:


Start
Soft Reset occured | Exit from Hard Reset
RCH_Wait_For_ Message_From_ Protocol_Layer
Actions on entry:
Clear Extended Rx Buffer
Clear Abort Flag
RCH_Report_Error
Actions on entry:
Report Error to Policy Engine.
If a Message was received, pass it to the Policy Engine.
Any Message Received and not in state RCH_Waiting_Chunk or RCH_Wait_For_Message_From_ Protocol_Layer
Reported
Chunked != Chunking
Message Passed
Received Non-Extended Message | (Received Extended Message & (Chunking = 0 & Chunked = 0))
Other Message Received from Protocol Layer | ChunkSenderResponseTimer timeout
RCH_Pass_Up_Message
Actions on entry:
Pass Message to Policy Engine
Abort Flag Set
Received Extended Message & (Chunking = 1 & Chunked = 1)
Unexpected Chunk Number
Transmission Error from Protocol Layer | Message Received from Protocol Layer
Message is Complete (Num bytes received >= specified Data Size)2
RCH_Processing_ Extended_Message
Actions on entry:
If first chunk: set Chunk_Number_Expected = 0 and Num bytes received = 0
If expected Chunk Number: Append data to Extended_Message_Buffer; Increment Chunk_Number_Expected and adjust Num bytes received.
Message not Complete
RCH_Requesting_Chunk
Actions on entry:
Send notification SRT_Stop to SenderResponseTimer State Machine.
Send Chunk Request to Protocol Layer with Chunk Number = Chunk_Number_Expected.
Message Transmitted received from Protocol Layer
RCH_Waiting_Chunk
Actions on entry:
Start ChunkSenderResponseTimer3
Send notification SRT_Start to SenderResponseTimer State Machine.3
Chunk Response Received from Protocol Layer

[1] Chunking is an internal state that is set to 1 if the ‘Unchunked Extended Messages Supported’ bit in either Source Capabilities or Request is 0. It defaults to 1 and is set after the first exchange of Source Capabilities and Request. It is also set to 1 for ***SOP’*** or ***SOP’’*** communication.

[2] Additional bytes received over specified ***Data Size*** will be as a result of padding in the last chunk.

[3.] This state is responsible for starting two timers of similar length. The implementor should mitigate against more than one of these timers resulting in recovery action.

(d). Section 6.12.2.1.2.4 “RCH_Requesting_Chunk State”, Page 253

From Text:

On entry to the ***RCH_Requesting_Chunk*** state the Chunked Rx state machine ***Shall***:

- Send Chunk Request to Protocol Layer with ***Chunk Number*** = Chunk_Number_Expected.

The Chunked Rx State Machine ***Shall*** transition to the ***RCH_Waiting_Chunk*** state when:

- Message Transmitted is received from the Protocol Layer.

The Chunked Rx State Machine ***Shall*** transition to the ***RCH_Report_Error*** state when:

- Transmission Error is received from the Protocol Layer, or
- A Message is received from the Protocol Layer.

To Text:

On entry to the ***RCH_Requesting_Chunk*** state the Chunked Rx state machine ***Shall***:

- Send notification SRT_Stop to SenderResponseTimer state machine (see Section 8.3.3.1.1)
- Send Chunk Request to Protocol Layer with ***Chunk Number*** = Chunk_Number_Expected.

The Chunked Rx State Machine ***Shall*** transition to the ***RCH_Waiting_Chunk*** state when:

- Message Transmitted is received from the Protocol Layer.

The Chunked Rx State Machine ***Shall*** transition to the ***RCH_Report_Error*** state when:

- Transmission Error is received from the Protocol Layer, or
- A Message is received from the Protocol Layer.

(e). Section 6.12.2.1.2.5 "RCH_Waiting_Chunk State", Page 253

From Text:

On entry to the ***RCH_Waiting_Chunk*** state the Chunked Rx state machine ***Shall***:

- Start the ***ChunkSenderResponseTimer.***

The Chunked Rx State Machine ***Shall*** transition to the ***RCH_Processing_Extended_Message*** state when:

- A Chunk is received from the Protocol Layer.

The Chunked Rx State Machine ***Shall*** transition to the ***RCH_Report_Error*** state when:

- A Message, other than a Chunk, is received from the Protocol Layer, or
- The ***ChunkSenderResponseTimer*** expires.

To Text:

On entry to the ***RCH_Waiting_Chunk*** state the Chunked Rx state machine ***Shall***:

- Start the ***ChunkSenderResponseTimer.***
- Send notification SRT_Start to SenderResponseTimer state machine (see Section 8.3.3.1.1)

The Chunked Rx State Machine ***Shall*** transition to the ***RCH_Processing_Extended_Message*** state when:

- A Chunk is received from the Protocol Layer.

The Chunked Rx State Machine ***Shall*** transition to the ***RCH_Report_Error*** state when:

- A Message, other than a Chunk, is received from the Protocol Layer, or
- The ***ChunkSenderResponseTimer*** expires.

(e). Section 8.3.3.1 "8.3.3.1 Introduction to state diagrams used in Chapter 8", Page 567

From Text:

Timers are included in many of the states. Timers are initialized (set to their starting condition) and run (timer is counting) in the particular state it is referenced. As soon as the state is exited then the timer is no longer active. Where the timers continue to run outside of the state (such as the ***NoResponseTimer***), this is called out in the text. Timeouts of the timers are listed as conditions on state transitions.

To Text:

Timers are included in many of the states. Timers are initialized (set to their starting condition) and run (timer is counting) in the particular state it is referenced. As soon as the state is exited then the timer is no longer active. Where the timers continue to run outside of the state (such as the ***NoResponseTimer***), this is called out in the text. Timeouts of the timers are listed as conditions on state transitions.

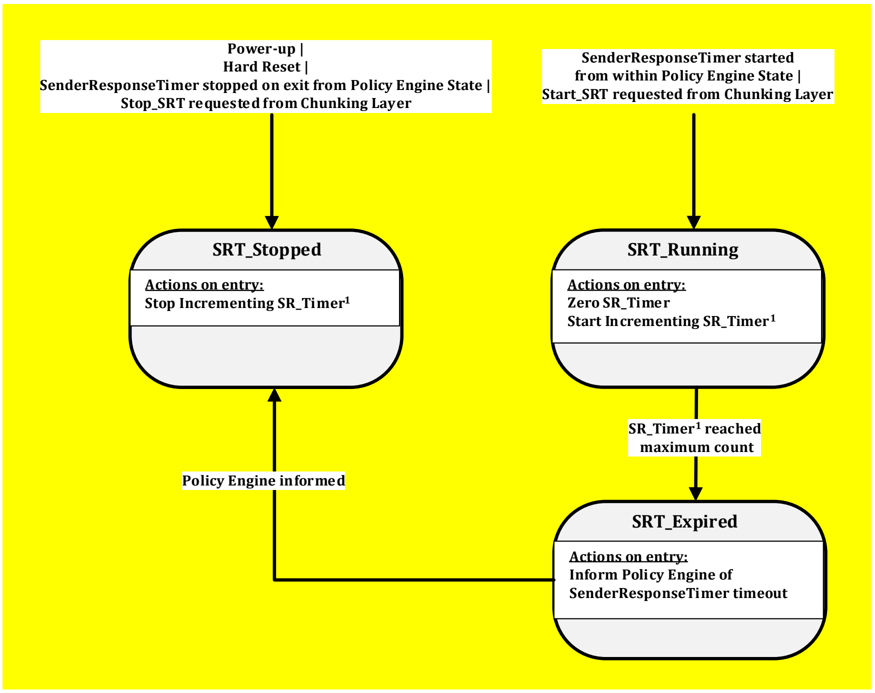
The SenderResponseTimer is a special case, as it may be stopped and started from outside the states in which it is used. To allow this to be done without over-complicating the state diagrams, the SenderResponseTimer is described with its own state diagram (Figure 8-8xx). The control of this Timer is shared between the Policy Engine and the Chunking Layer.

(d). Section 8.3.3.1.1 "SenderResponseTimer State Diagram", Page 253

New Text:

Figure 8-8xx below shows the state diagram for the Policy Engine in a Source or a Sink Port. The following sections describe operation in each of the states.

Figure 8-8xx SenderResponseTimer State Diagram


Power-up |
Hard Reset |
SenderResponseTimer stopped on exit from Policy Engine State |
Stop_SRT requested from Chunking Layer
SenderResponseTimer started from within Policy Engine State |
Start_SRT requested from Chunking Layer
SRT_Stopped
Actions on entry:
Stop Incrementing SR_Timer¹
SRT_Running
Actions on entry:
Zero SR_Timer
Start Incrementing SR_Timer¹
SR_Timer¹ reached maximum count
Policy Engine informed
SRT_Expired
Actions on entry:
Inform Policy Engine of SenderResponseTimer timeout

1. The SR_Timer is regarded as the mechanism within the SenderResponseTimer state machine that implements the SenderResponseTimer.

8.3.3.1.1.1 SRT_Stopped State

The ***SRT_Stopped*** State ***Shall*** be the starting state for the SenderResponseTimer either on power up or after a Hard Reset. On entry to this state the Policy Engine ***Shall*** stop incrementing the SR_Timer.

The Policy Engine ***Shall*** transition to the ***SRT_Running*** state:

- When the SenderResponseTimer is started from within a Policy Engine state, or
- When a Start_SRT is requested from the Chunking Layer.

8.3.3.1.1.2 SRT_Running State

On entry to the ***SRT_Running*** state the SenderResponseTimer state machine ***Shall***:

- set the SR_Timer to zero
- start running SR_Timer.

The SenderResponseTimer state machine ***Shall*** transition to the ***SRT_Expired*** state:

- When the SR_Timer reaches its maximum count

The SenderResponseTimer state machine ***Shall*** transition to the ***SRT_Stopped*** state:

- When the SenderResponseTimer is stopped by exiting a Policy Engine state, or
- When a Stop_SRT is requested from the Chunking Layer

8.3.3.1.1.3 SRT_Expired State

On entry to the ***SRT_Running*** state the SenderResponseTimer state machine ***Shall*** Inform Policy Engine of SenderResponseTimer timeout

The Policy Engine ***Shall*** then transition to the ***SRT_Stopped*** state:

- When the policy Engine has been informed.

(d). Section 8.3.3.28 "Policy Engine States", Page 665

New Text:

State name	Reference
Error! Reference source not found.	
SRT_Stopped	Section **Error! Reference source not found.**
SRT_Running	Section **Error! Reference source not found.**
SRT_Expired	Section **Error! Reference source not found.**