ebXML Transport, Routing & Packaging Reliable Messaging Specification

Working Draft 2-November-2000

This version:
ebXML Reliable Messaging Specification v0-084.doc
Latest version:
N/A
Previous version:
v0-080
Editor:
Jim Hughes <jfh@fs.fujitsu.com>
Authors:
Masayoshi Shimamura <shima@rp.open.cs.fujitsu.co.jp>
Contributors:
See Acknowledgements

Abstract

This document defines the structures and processes used to provide improved Reliable Messaging within the ebXML Transport, Routing and Packaging architecture.

This version proposes items that would be added to the basic Messaging Services specification during Phase 2 of the TRP activities.

Status of this Document

This document represents work in progress and no reliance should be made on its content.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in IETF RFC 2119.

Editor Note 1: This version of the Reliable Messaging specification:
– removes sections already included in MS spec v0.21d
– modifies the sequence number definition
– adds sliding window algorithm information
-- adds a proposal for network routing
– modifies non-normative material

Table of Contents

1 Introduction

1.1 Purpose and Scope

This specification defines the Reliable Messaging function used between ebXML Messaging Services. It responds to the requirements for Reliable Messaging found in section 4.2(1) of Reference [1]. Material from this draft document is identified for incorporation into the Messaging Services Specification.

The current Messaging Service Specification v0.21d already includes basic Reliable Messaging functions. This specification defines additional functions to be added to the Messaging Service Specification.

Where appropriate, MS-Editor notes are provided to show where items might appear in version 0.21d of the Messaging Services specification.

2 Sections to Add or Modify in MS Ver0.21d

The material in this section is suitable for inclusion in Phase 2 specifications.

2.1 XML Routing Header (replaces MS section 7.10)

Editor Note 2: Modifications consist of:
- Sequence Number begins with "0" instead of "1"
- Range of Sequence Numbers reduced to allow signed values
- allows for resetting Sequence Number
- removes implication of unilateral Sender reset
- adds a Sequence Number attribute to show a reset

One ***RoutingHeader*** element immediately follows the ***Header*** element. It is required in all ***ebXMLHeader*** documents. The ***RoutingHeader*** element is a composite element comprised of at least the following 4 required subordinate elements:

- ***SenderURI*** – the Sender's Messaging Service Handler URI.
- ***ReceiverURI*** – the Receiver's Messaging Service Handler URI.
- ***ErrorURI*** – URI designated by the Sender for reporting errors.
- ***Timestamp*** – timestamp of the ***RoutingHeader*** creation, in the same format used for ***Timestamp*** in the ***XML Header MessageData*** element.

When the ***RoutingHeader*** is used for a message sent with Reliable Messaging functions (***DeliverySemantics*** is set to "OnceAndOnlyOnce" in the ***XML Header ReliableMessagingInfo*** element), the Sender SHALL add one additional ***RoutingHeader*** element to the ***RoutingHeader***:

- ***SequenceNumber*** – Integer value that is incremented (e.g. 0, 1, 2, 3, 4...) for each Sender-prepared message sent to the Receiver. The Sequence Number consists of ASCII numerals in the range 0-99,999,999. In following cases, the Sequence Number takes the value "0":
    a) First message from the Sender to a particular Receiver
    b) First message after resetting Sequence Number information in the Sender

c) First message after wraparound (next value after 99,999,999)

The ***SequenceNumber*** element has a single attribute, ***Status***. This attribute is an enumeration, which shall have one of the following values:

- "Reset" – the Sequence Number is reset as shown in (a) or (b) above
- "Continue" – the Sequence Number continues sequentially (including (c) above)

When the Sequence Number is set to "0" because of (a) or (b) above, the ***Status*** attribute of all the messages contained in the current Sending Window (including the message with sequence number "0") is set to "Reset". The allows the Receiver to learn of the reset if messages are received out of order. For example, when the ***WindowSize*** is four, all the ***Status*** attributes of Sequence Number 0 to 3 take the value "Reset".

In all other cases, including (c) above, the ***Status*** attribute takes value "Continue".

The following fragment demonstrates the structure of the ***RoutingHeader*** element of the ***ebXMLHeader*** document when Reliable Messaging is used:

```
<RoutingHeader>
  <SenderURI>...</SenderURI>
  <ReceiverURI>...</ReceiverURI>
  <ErrorURI>...</ErrorURI>
  <Timestamp>...</Timestamp>
  <SequenceNumber Status="Reset">00000000</SequenceNumber>
</RoutingHeader>
```

2.2 Recovery Sequence for Transient Errors (addition to current MS section 7.12)

Editor Note 3: The following section is inserted as MS section 7.12.3, and the existing 7.12.3 is changed to 7.12.4. The changes proposed in this RM section 2.3 will alter this 7.12.x numbering, since a new section is inserted between 7.11 and 7.12.

When the Sender receives the error message "Transient Error", the appropriate recovery handler in the Sender executes a Messaging Service recovery sequence. The recovery sequence SHALL suspend sending of further messages to the Receiver for the period specified in the **MinRetrySecs** field in the error message. If the **MinRetrySecs** field does not exist in the error message, the **RetryInterval** specified in the TPA or elsewhere is used as the suspension time.

After the suspension, the Sender's recovery handler SHALL re-send the sent message to the Receiver. The format of the re-sent message is exactly the same as the original message. In the recovery sequence or after the recovery sequence,

- If the Sender receives the error message "Transient Errors" again, the recovery handler repeats the recovery sequence.
- If the Sender detects or receives another error, the recovery handler executes the appropriate recovery sequence for the error.
- If the Sender receives an Acknowledgment Message, the message transmission is completed.

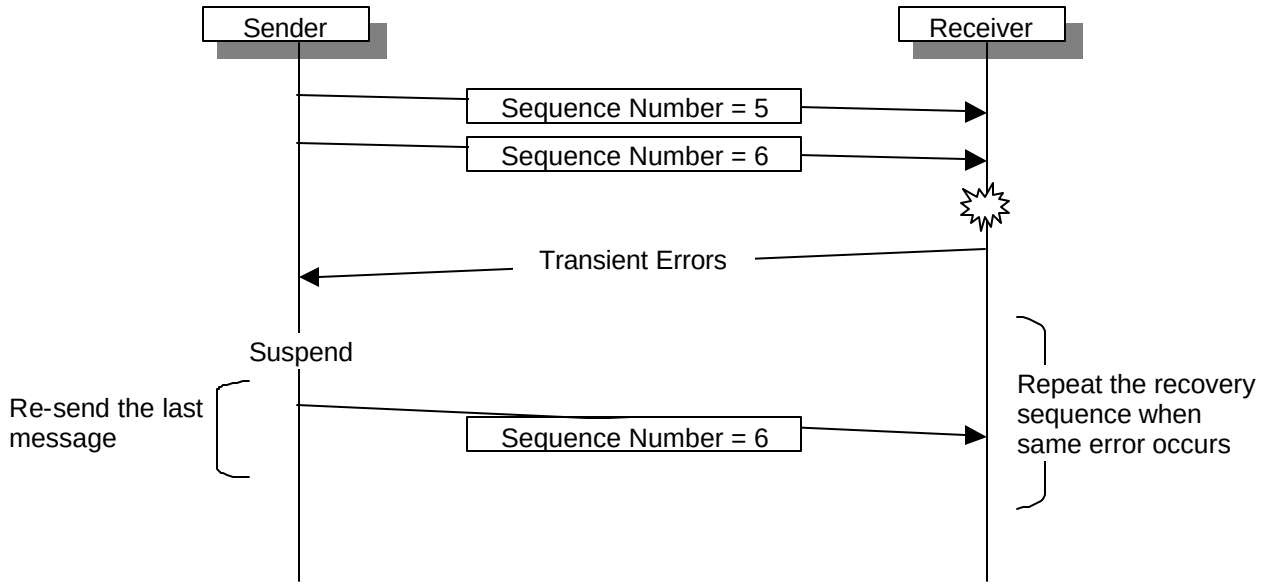


Figure 2-6: Recovery Sequence for Transient Errors

2.3 Reliable Messaging Sliding Window (new section, numbered MS section 7.12)

Editor Note 4: This section is inserted as a new section 7.12 and the existing 7.12-7.14 section numbers are incremented.

2.3.1 Sliding Window Overview (MS section 7.12.1)

In Reliable Messaging, the Sender and the Receiver SHALL use a "Sliding Window" algorithm described here to guarantee message order and to prevent overflow of message buffers in the Receiver.

In Sliding Window, the Sender manages sending of messages as following:

- The Sender has a ***Sending Window*** that demarks a sequence of messages, starting from the first unsent or unacknowledged message being sent.
- The Sending Window identifies a scope of sequential messages that the Sender MAY send at once without waiting for individual Acknowledgement Messages using Reliable Messaging. The Sender SHALL NOT send messages which do not belong to the Sending Window.
- The Sender may advance the Sending Window by a range of 1 to the size of the Sending Window only when the Sender receives Acknowledgement Messages.

The ***WindowSize*** parameter specifying size of the Sending Window may be determined in a number of ways, such as the TPA or some other method, and this same value is used by the Sender and Receiver.

In the Receiver, WindowSize determines the maximum number of messages which the Receiver may received at one time before sending an Acknowledgement Message to the Sender. The Receiver can calculate the required maximum buffer size using the WindowSize and other parameters (such as a maximum length of message), and can correct any invalid order of messages in the buffer by using Sequence Number before passing these messages to a higher level.

The following figure shows an example of advancing the Sending Window in the Sender, based on acknowledgements received:

(a) The Sender has already sent messages with sequence numbers 0 to 3 and has received Acknowledgement Messages for these sent messages. Thus, the message with sequence number 4 is the first unacknowledged message. Since the size of Sending Window is four, the Sending Window covers messages with sequence numbers 4 to 7. The Sender sends these messages in the Sending Window at one time without waiting for any Acknowledgement Messages. In the message transfer, the message with sequence number 6 is lost by an error.

(b) Since the message with sequence number 6 does not reach the Receiver, the Sender only receives Acknowledgement Messages for the sent messages with sequence numbers 4, 5 and 7.

(c) The message with sequence number 6 becomes the first unacknowledged or unsent message. The Sending Window advances by 2 to cover messages with sequence numbers 6 through 9.

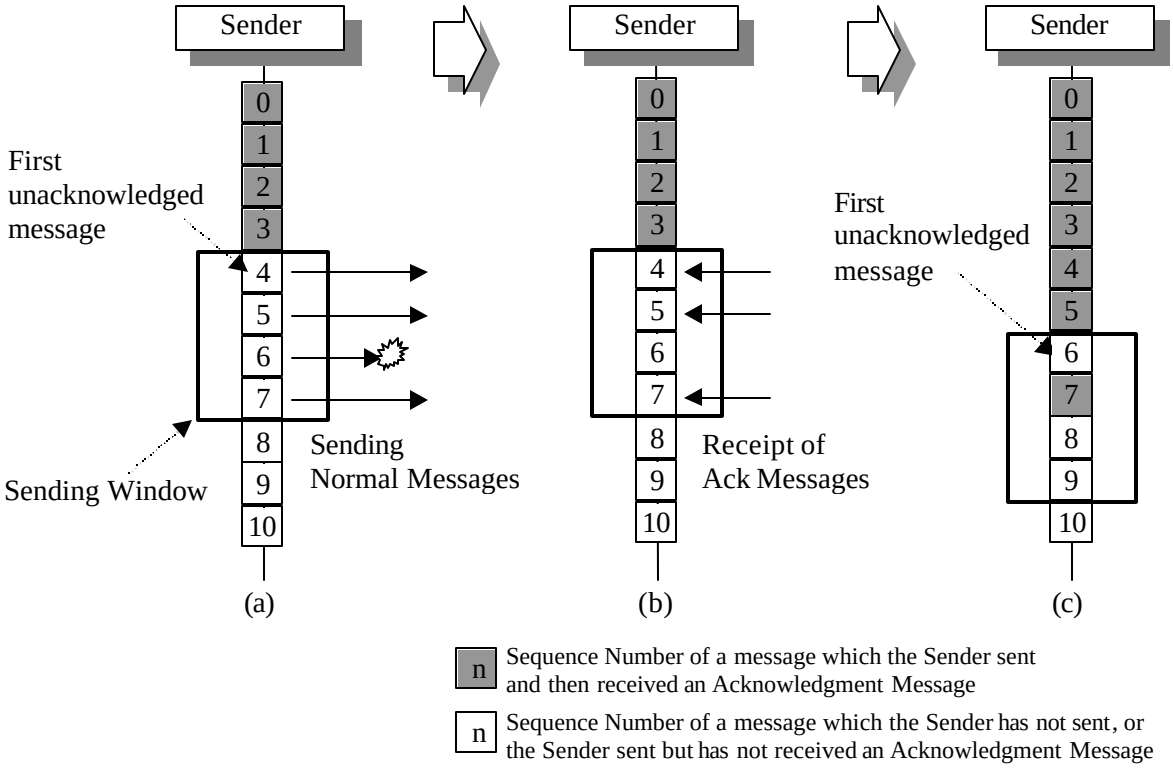


Figure 2-1 Sliding of Sending Window (window size = 4, sliding width = 2)

2.3.2 Sliding Window Recovery (MS section 7.12.2)

When an MSH error occurs in a message which belongs to a Sending Window, the appropriate recovery handler in the Sender SHALL execute the appropriate Messaging Service recovery sequence defined in the section "7.13 Reliable Messaging Recovery Procedures".

When the Sender has sent all the messages which belong to a Sending Window, but the Sender still has not received an Acknowledgement Message for the first message in the Sending Window, the appropriate recovery handler in the Sender MAY execute a Messaging Service recovery sequence. The recovery sequence is same as recovery sequence for ***Timeout***.

2.3.3 Sliding Window Parameters (MS section 7.12.3)

In Sliding Window, the messaging service uses the following Messaging Service parameter.

This information may be determined in a number of ways, such as the TPA or some other method.

Argument	*Outline Description*
WindowSize	Size of Sending Window. • Integer value specifying a number of messages. • Identifies number of sequential messages which the Sender MAY send at once without waiting for Acknowledgement Messages using Reliable Messaging. • Identifies the maximum number of messages which the Receiver could receive before sending an Acknowledgement Message to the Sender.

2.4 Reliable Routing (new section, added to MS section 7)

Editor Note 5: Probably insert this section as 7.13, before the Error Reporting section.

2.4.1 Store and Forward Semantics

Reliable Routing consists of a series of individual simple Reliable Messaging transmissions which are each between a Sender and a Receiver. These *Store and Forward* semantics consist of the following sequence:

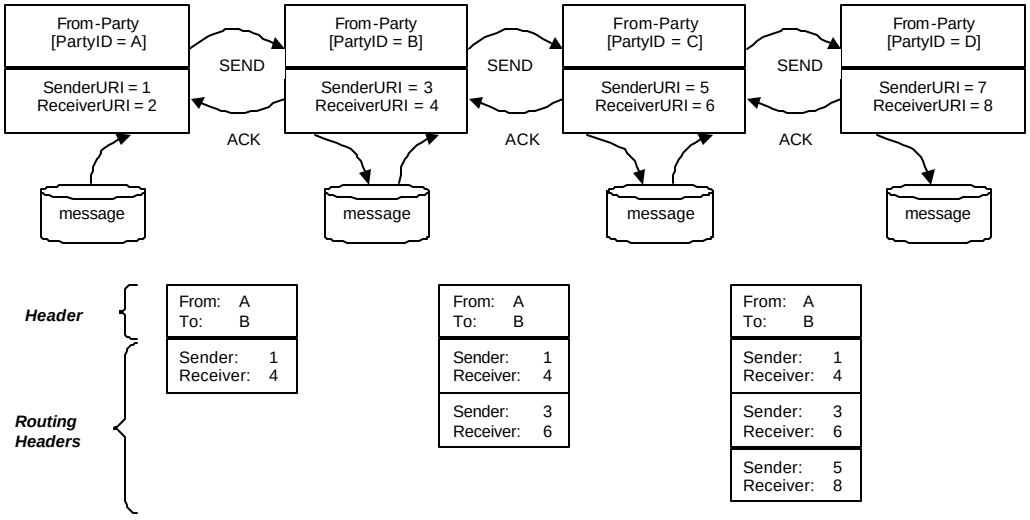
(1) Sender A transfers a message to Receiver B using Reliable Messaging.

(2) After completion of the reliable messaging transmission between Sender A and Receiver B, Sender B transfers the received message to Receiver C using Reliable Messaging.

(3) After completion of the reliable messaging transmission between Sender B and Receiver C, Sender C transfers the received message to Receiver D using Reliable Messaging.

(4) [Repeat until end of routing]

205 **Figure 2-2 Reliable Routing**



206

207

208 ### 2.4.2 Routing Information

209 The first Sender (From-Party's Sender) specifies the From/To elements in the Header, and the
210 SenderURI/ReceiverURI elements in the Routing Header for first message transferred to the
211 Router.

212 When the message is forwarded between Routers, the Router adds a new ***RoutingHeader*** to the
213 end of the ebXML Header Document, and this new Routing Header contains new
214 SenderURI/ReceiverURI elements for message forwarding to the next Router.

215 ### 2.4.3 Error Handling in Routing

216 When message forwarding to a subsequent Router is not available or fails at a particular Router,
217 and if that Router received the message from previous Sender, the Router's Receiver returns an
218 ErrorMessage (Transient Error) to the Sender instead of an Acknowledgement Message. By this
219 rule, the Sending Messaging Service Handler will not receive a Messaging Service
220 acknowledgement of successful transmission until the Receiving Party's Message Service
221 Handler has actually received and stored the message.

222 ## 2.5 Other Minor Changes to MS v0.21d

223 MS Section 7.12.2

224 Line 820-821: Change "the last sent message" to "the sent message"

225 Line 824-825: Change "the final message" to "the message"

226 Line 825: Insert period after "attempts"

227 Figure 7-3: Change "Re-send the last message" to "Re-send the sent message"

228 MS Section E.4

229 Line 1706: Change "the last message" to "the sent message"

230 Figure E.4: Change "Re-send the last message" to "Re-send the sent message"

3 Modifications to ebXML Glossary

The following items should be placed in the approved ebXML Glossary.

3.1 Reliable Messaging Terms

3.1.1 Once And Only Once

A message delivery semantic that means:

- Message delivery is guaranteed under most circumstances, and the Sending Party will be notified if there is no delivery.
- A message will always reach the Receiving Party no more than once.
- If a message does not reach the Receiving Party, the Sending Party does not need to execute retry procedures (retry is automatically executed by the messaging service).

3.1.2 At Most Once

A message delivery semantic that means:

- Message delivery is not guaranteed
- A message will always reach the Receiving Party no more than once.
- If a message is not delivered, the Sending Party can detect the incident, and if the Sending Party wants to guarantee message delivery, the Sending Party must execute retry procedures

3.1.3 Best Effort

A message delivery semantic that means:

- Message delivery is not guaranteed
- A message will always reach the Receiving Party no more than once.
- If a message does not reach the Receiving Party, the Sending Party can not detect the incident

4 Phase 2/Phase 3 Activities

Material in this section will be discussed in future TRP meetings as a likely base for further additions to the Messaging Service specification.

4.1 Transfer of Large Documents within Messages

When the Sender wishes to send a large Document, the Sender may refer to the following Messaging Service parameters. This information may be determined in a number of ways, such as the TPA or some other method.

Table 4-1 Messaging Service Parameters used in transfer of large documents

Argument	*Outline Description*

MaxSize	Maximum size of a payload. • Integer value specifying the number of bytes. • The Sender SHALL NOT send an ebXML message which contains a payload larger than the MaxSize
CompressEncoding	Encoding to compress the Payload. • A string specifying the encoding. • The Sender MAY compress the Payload using the encoding. The Receiver SHALL uncompress the encoded Payload before passing it to the application.

When the Sender needs to send a message which would have a payload larger than **MaxSize**, the Sender can use a compression encoding specified by **CompressEncoding** to compress the Payload.

After the compression, if the resulting payload is still larger than the MaxSize, the Sender can split the Document into parts and can send these parts separately as individual Payload Documents using individual Normal Messages.

When a Normal Message carries part of a split Document as its Payload, the **MessageData** element in the **Header** element has **SplitId** and **SplitNumber** elements.

- **SplitId** – a unique identifier conforming to [RFC2392] for the Document which was split. All the Normal Messages which carry a portion of same Document have same SplitId.
- **SplitNumber** – Integer value that is incremented (e.g. 1, 2, 3, 4...) for each Sender prepared message to carry part of same Document. The SplitNumber starts from "1" and is incremented in order of the part. The SplitNumber is unique only within a specific SplitId. The Split Number has a single attribute, **Total**.
  - "Total" – Integer value which indicate total number of the parts comprising the split Document.

The following fragment demonstrates the structure of the **SplitId** element and **SplitNumber** element of the **MessageData** element:

```
<MessageData>
    <MessageId>UUID-A</MessageId>
    ...
    <SplitId>UUID-B</SplitId>
    <SplitNumber Total="5">3</SplitNumber>
</MessageData>
```

5 Non-Normative Material

The majority of the material in this section should remain in a non-normative Implementer's Guide.

5.1 Basic Concepts

To achieve reliable messaging between Parties, this specification defines a process that enables the Parties' ebXML Messaging Services to communicate with each other using "Once and Only Once" semantics, coupled with a timeout to determine lost messages.

For the purposes of this document, the term "*Sender*" means the Sending Party's Messaging Service that sends the message on the underlying message transport, and "*Receiver*" means the Messaging Service used by the Receiving Party. The term "*From-Party*" means the party that originally prepared the message and provided the message to its Messaging Service, and the term "*To-Party*" means the party that was identified by the From-Party as the final recipient of the message.

For example, a simple message transmission using two Message Service Handlers and one transport is shown in Figure 2-5-1.

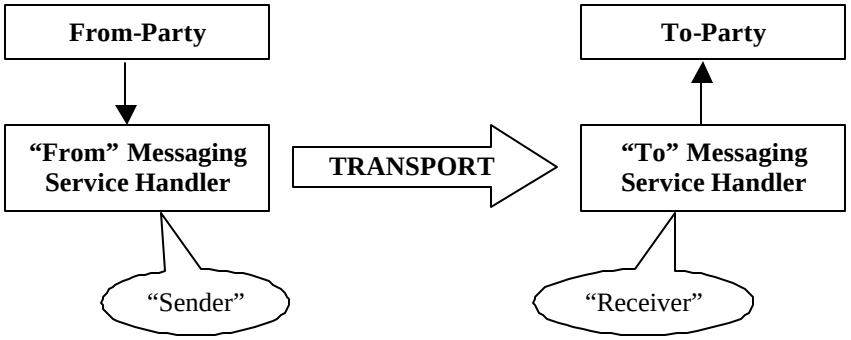


Figure 2-5-1: Simple Message Transmission

Reliable Messaging consists of the following basic concepts:

1) Messages are sent and received through Messaging Service Handlers (MSH), which function on behalf of their respective Parties (and Business Processes). With respect to a particular underlying transport, each MSH can be identified as a "Sender" or a "Receiver".

2) A message is identified by its **MessageId** field, which is contained in the Message Header's **MessageData** element created by the Sender.

3) When the From-Party requests Reliable Messaging semantics for the message, the Sender sets the **DeliverySemantics** field in the **ReliableMessagingInfo** element of the Message Header to "OnceAndOnlyOnce".

4) Reliable Messaging processing requires no changes to the Message Header during transmission, once the Message Header is prepared.

5) Reliable Messaging uses a "Routing Header" contained in the Message Envelope.

6) A Reliable message indicated by setting the **DeliverySemantics** field to **OnceAndOnlyOnce.**

7) For each reliable message, the Sender generates a **Sequence Number** that is unique to the MSH Sender-Receiver pair. For subsequent reliable messages, the Sender increments the Sequence Number placed in that message. The **Sequence Number** is contained in the Routing Header Data Element.

8) A Messaging Service level Acknowledgement is sent from the Receiver to the Sender for every received message with a message type of Normal after persisting the message.

9) Within a reliable message transmission, the Receiver must determine whether a received message is a duplicate message. Two possible approaches are through using the **MessageId** and/or the Sender-Receiver unique **Sequence Number.** If the received message is a duplicate, the Receiver discards the message after sending the acknowledgement. If the message is not a duplicate, the Receiver stores the message in its persistent storage, sends an acknowledgement and delivers the message to a higher processing level.

10) Because every message received with Reliable Messaging semantics will cause the sending of a related Acknowledgement Message, the Sender must be prepared to discard duplicate Acknowledgement Messages if multiple copies of the original message are sent.

11) To detect loss of a reliable message, the Sender sets a timeout, retry interval and number of retries for that message. If the transmitted reliable message is lost due to system or communication failure, the Sender will re-send this message using these parameters before reporting failure to the From-Party. These values might be specified in the Trading Partner Agreement (TPA) or some other fashion.

5.2 Detection of Repeated Messages by the Receiver

Detection of repeated messages in the Receiver using **Message Identifiers** and/or **Sequence Numbers** is implementation dependent.

Comparison of Message Identifiers could be used to detect duplicated messages. Another effective detection logic can be suggested which uses **Sequence Numbers**, which are unique to a particular Sender-Receiver pair.

The Receiver receives the reliable message and then compares the received reliable message's **Sequence Number** with the immediately previous reliable message's **Sequence Number**.

5.2.1 Duplication Check Window

In Reliable Messaging with Sliding Window, the Receiver can use Sequence Numbers and a **Duplication Check Window** to detect for duplication of messages. The Receiver manages the Duplication Check Window as following:

- The Receiver has a Duplication Check Window that terminates at the last received message. The Duplication Check Window has a size that is specified by the WindowSize parameter.
- The Duplication Check Window identifies a scope of sequential messages that the Receiver shall use when checking whether a received message is a duplicate or not.
- The Receiver can advance the Duplication Check Window up to the size of the Duplication Check Window when the Receiver receives new messages.

The following diagram shows how the Duplication Check Window is advance by the Receiver.

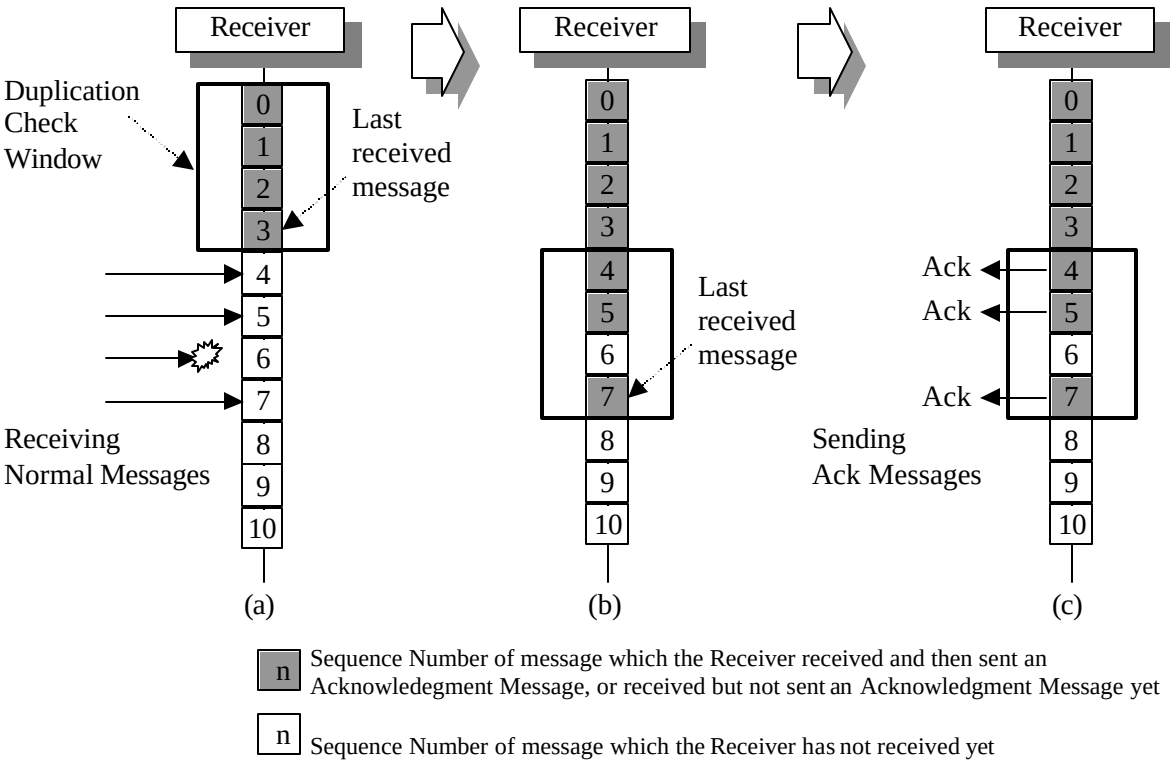


Figure 5-2 Sliding of Duplication Check Window (window size = 4, sliding = 2)

5.2.2 New Message Window

In Reliable Messaging with Sliding Window, the Receiver can use Sequence Numbers and **New Message Window** to detect new messages (not duplicate messages). The Receiver manages the New Message Window as following:

- The Receiver has a New Message Window that starts from the first unreceived or unacknowledged message. The New Message Window size is specified by WindowSize parameter.
- The New Message Window identifies a scope of sequential messages that the Receiver does not need to check whether the received message is a duplicate or not.
- The Receiver can advance the New Message Window up to the size of the Duplication Check Window specified by the WindowSize parameter only when the Receiver returns Acknowledgement Messages to the Sender

The following diagram illustrates advancing the New Message Window.

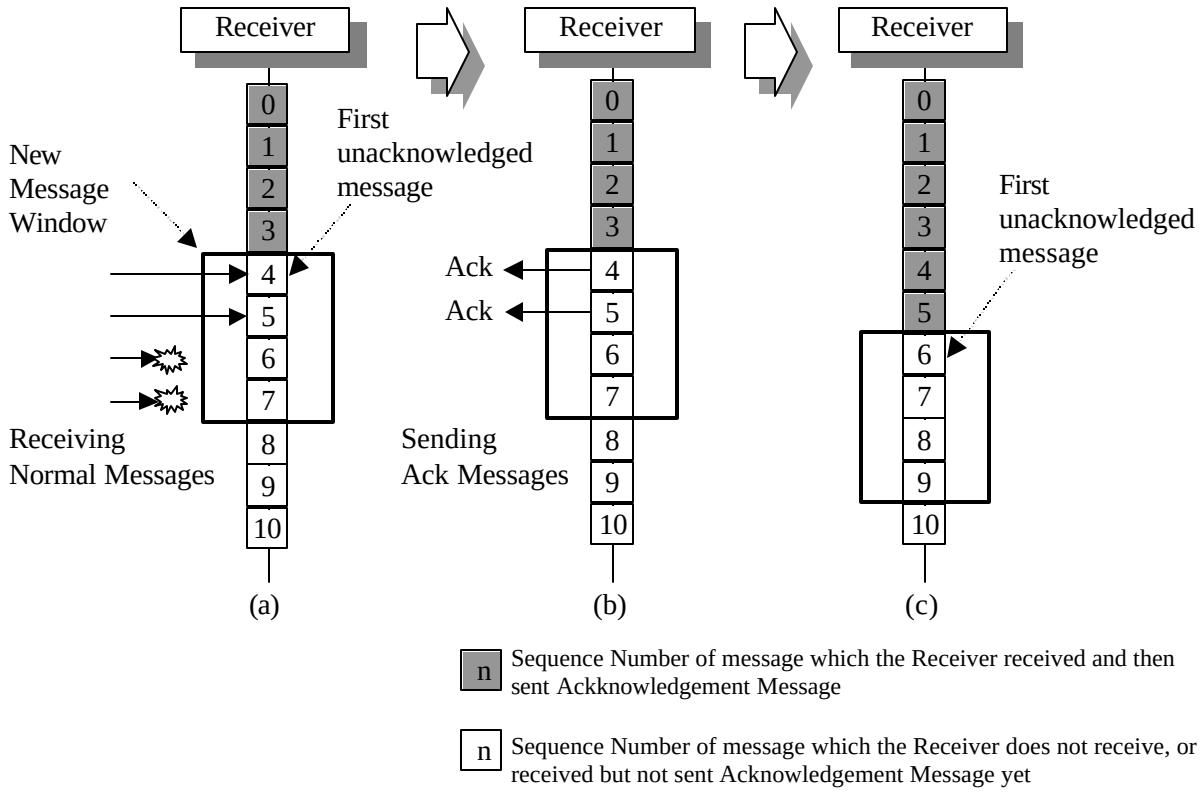


Figure 5-3 Sliding of New Message Window (window size = 4, sliding width = 2)

5.2.3 Process for checking Received Messages

The Receiver can detect duplication of messages and invalid messages using the Duplication Check Window and the New Message Window as following:

1) If the received message belongs to the Duplication Check Window, duplication check shall be executed:
   - If it is a duplicate, the Receiver throws it away and returns an Acknowledgement Message.
   - If it is not a duplicate, the Receiver memorizes its sequence number, stores the message and returns an Acknowledgement Message
2) If the received message does not belong to Duplication Check Window and belongs to New Message Window, it is a new message
   - The Receiver memorizes its sequence number, stores the message and returns Acknowledgement Message
3) Any other case, the message is invalid.
   - The Receiver throws it away and returns Error Message (ebXML Message Error)

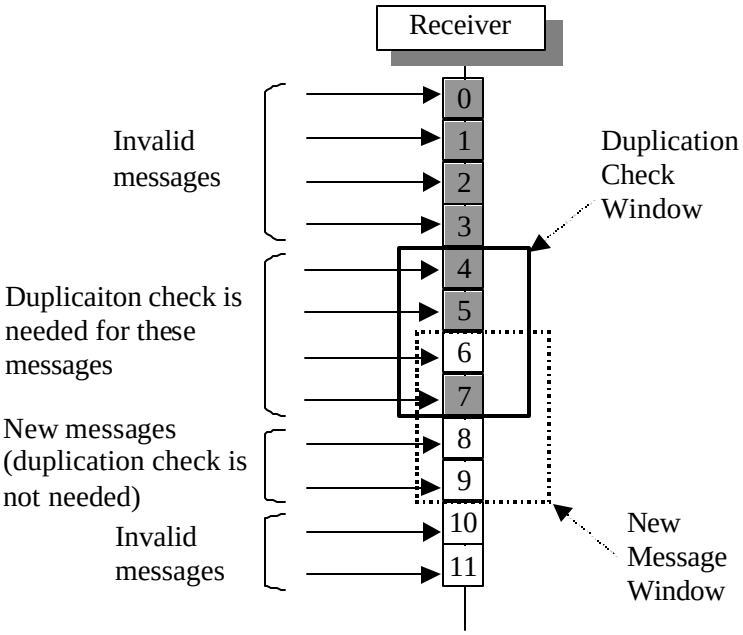


Figure 5-4 Process for checking received messages

6 Acknowledgements

The author wishes to acknowledge the members of the ebXML TR&P who commented on Fujitsu's proposal in the face-to-face meetings and in e-mail.

7 Author's Address

Masayoshi Shimamura
Fujitsu Limited
Shinyokohama Nikko Bldg., 15-16, Shinyokohama 2-chome
Kohoku-ku, Yokohama 222-0033, Japan
Telephone: +81-45-476-4590
E-mail: shima@rp.open.cs.fujitsu.co.jp