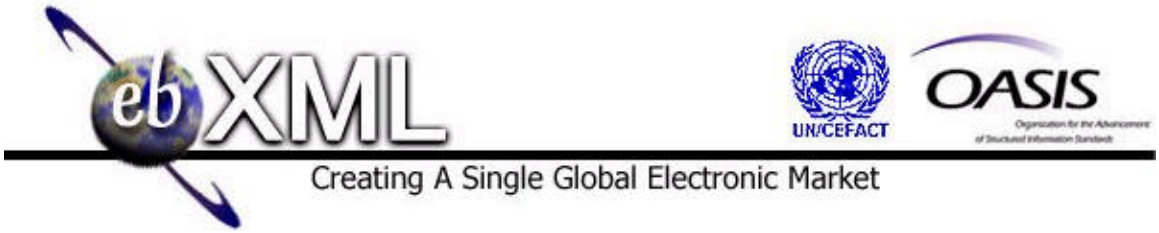


ebXML Transport, Routing & Packaging Modifications to Message Specification v0.8

Working Draft 14-November-2000

This version:
ebXML MS Modifications v0-088.doc
Latest version:
N/A
Previous version:
N/A
Editor:
TRP Team
Authors:
TRP Team
Contributors:
See Acknowledgements

Abstract

This document defines proposed modifications to the ebXML Message Services specification version 0.8. It is based on an original proposal from Fujitsu for Reliable Messaging technology, with added material from CommerceOne.

This version proposes items that would be added to the basic Message Services specification during Phase 2/3 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.

Table of Contents

1 Introduction

1.1 Purpose and Scope

This document defines material for incorporation into the Message Services Specification.

2 Sections to Add or Modify in MS Ver0.8

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 1: 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 Message Service Handler URI.
- ***ReceiverURI*** – the Receiver's Message 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". This 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 2: 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 here 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 Message 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 CPA 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.

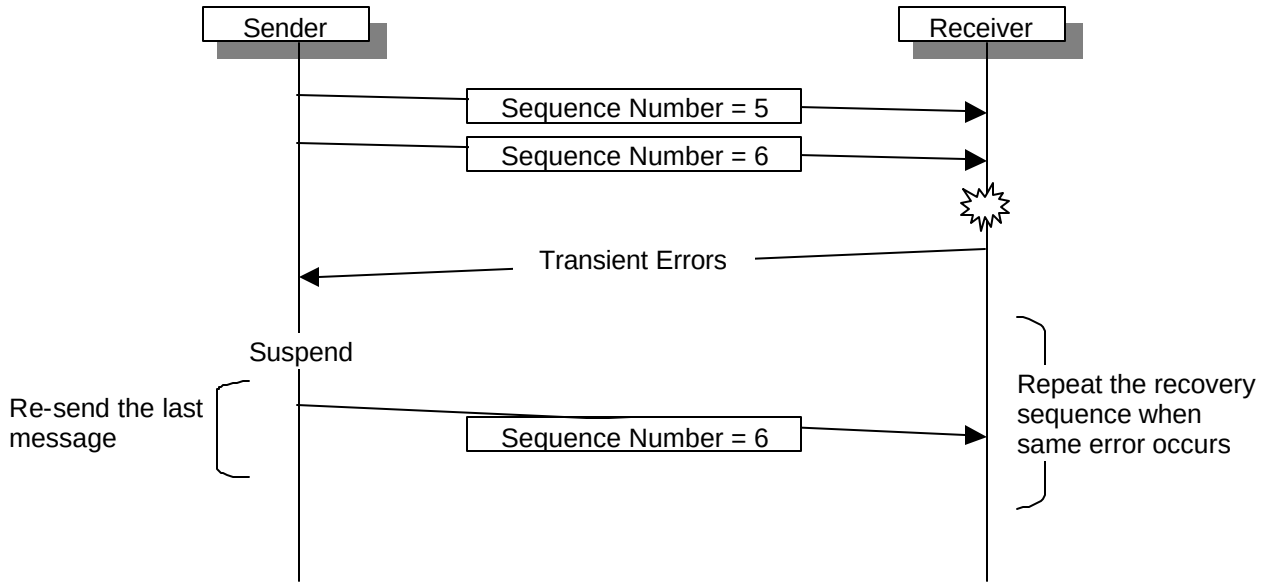


Figure 2-6: Recovery Sequence for Transient Errors

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

Editor Note 3: 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)

When sending messages using ***OnceAndOnlyOnce*** delivery semantics (Reliable Messaging), the Sender SHALL use a "Sliding Window" algorithm described below to guarantee message order and to prevent overflow of message buffers in the Receiver.

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

- The Sender has a ***Sending Window*** that demarks a sequence of reliable messages, starting from the first unsent or unacknowledged reliable message being sent.
- The ***Sending Window*** identifies a scope of sequential messages that the Sender MAY send at once without waiting for individual MSH-level Acknowledgement Messages using Reliable Messaging. The Sender SHALL NOT send reliable messages that 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 corresponding to reliable messages within the ***Sending Window*** that were sent.
- With respect to a particular Receiver, the Sender may send unreliable messages (those not identified with ***OnceAndOnlyOnce*** semantics) interspersed among the reliable messages, and these messages will not affect the Sliding Window procedures.

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

In the Receiver, WindowSize determines the maximum number of messages that the Receiver may receive at one time before sending Acknowledgement Messages 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 individual message Sequence Numbers before passing these messages to a higher level.

The following figure shows an example of advancing the Sending Window in the Sender, based on MSH-level Acknowledgements received from the Receiver:

(a) The Sender has already sent reliable 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 a transmission error.

(b) Since the message with sequence number 6 does not reach the Receiver, the Sender only receives Acknowledgement Messages for the reliable 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. The Sender is free to initially send messages 8 and 9, and to initiate retry procedures for message 6, as described in the next section.

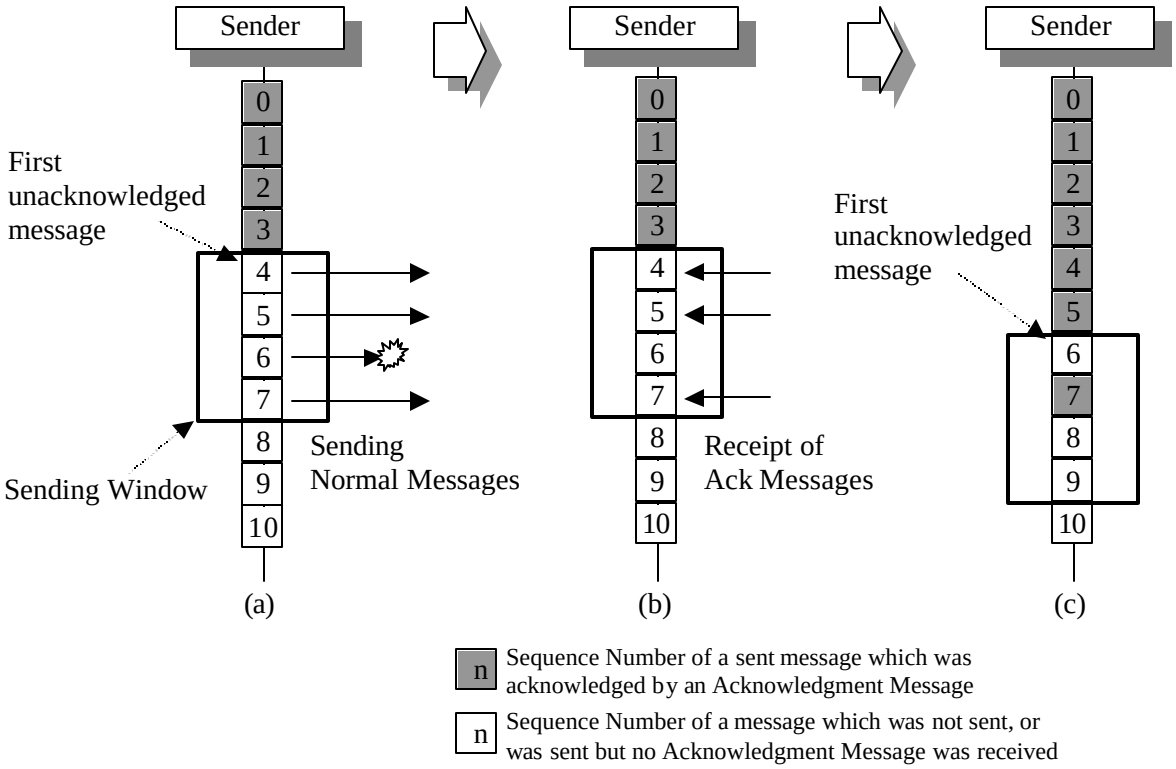


Figure 2-1 Sliding of Sending Window (window size = 4, sliding width = 2)

2.3.2 Sliding Window Recovery (MS section 7.12.2)

When an MSH transmission error occurs for a message which belongs to a Sending Window, the appropriate recovery handler in the Sender SHALL execute the appropriate Message 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 Message Service recovery sequence. The recovery sequence is same as recovery sequence for ***Timeout***.

2.3.3 Sliding Window Parameter (MS section 7.12.3)

In Sliding Window, both the Sender and Receiver Message Service Handlers use a common Message Service parameter.

This information may be determined in a number of ways, such as the CPA 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 ebXML Simple Routing (new section, added to MS section 7)

Editor Note 4: Probably insert this section as 7.13, before the Error Reporting section.

2.4.1 Simple Routing Overview

In the most simple case of ebXML Messaging, point-to-point messages are sent from the Message Service Handler associated with Party A (the *Sending_MSH*) to the Message Service Handler associated with Party B (the *Receiving_MSH*). In this case, there is exactly one Routing Header in the message that identifies the Message Service Handlers used in the transmission.

More complex routing networks can be constructed as sequences of pairwise MSH links, where the intermediate MSH nodes have higher level logic (above the MSH level) to receive messages, process them and pass the (possibly modified) message to the next MSH. In such a network, the intermediate nodes might radically transform the message being passed, possibly as the result of some workflow or other rules-based definitions.

However, a simple network might be constructed of intermediate Message Service Handlers that perform very limited functions on a message being transmitted through the network. The characteristics of such a network path are:

a) The initial node of the path consists of the *From_Party* and its associated *Sending_MSH*.

b) Intermediate nodes consist of two Message Service Handlers that receive and retransmit the message, and only a minimal set of logic is used to process the message.

c) The distinguishing characteristic of this model is that these intermediate nodes may only modify the message by adding a new Routing Header element. Especially, the *From_Party* and *To_Party* fields in the Header, and the Payload, are not modified at these intermediate nodes.

d) The terminal node of the path consists of the *Receiving_MSH* and its associated *To_Party.*

e) Messages may be sent with or without ***OnceAndOnlyOnce*** (Reliable Messaging) semantics.

f) Messages sent with Reliable Messaging may use one of three possible methods for the return of MSH-level Acknowledgement Messages, which are further described in later subsections:

   i. At each link of the network, the Receiver returns a normal MSH-level Acknowledgement Message to the Sender, completely asynchronous and with no relation to later transmissions in subsequent links.

   ii. In addition to (i) above, when the message is finally received at the end of the network, the terminal Receiver sends a Delivery Notification Message to the first Sender, using Reliable Messaging.

   iii. At each link of the network, the Hub defers sending a MSH-level Acknowledgement Message to the previous Sender until the next Receiver in the network has returned an Acknowledgement Message.

The following subsections further describe this model of *ebXML Simple Routing*.

2.4.2 Simple Routing – *Best Effort* Semantics

When used for messages without ***OnceAndOnlyOnce*** semantics, ebXML Simple Routing consists of a series of individual point-to-point Message Service transmissions that are each between a Sender and a Receiver. These semantics consist of the following sequence as illustrated in Figure 2-2:

(1) The Sending MSH for Party A prepares the message with one Routing Header and transfers the message to Hub 1.

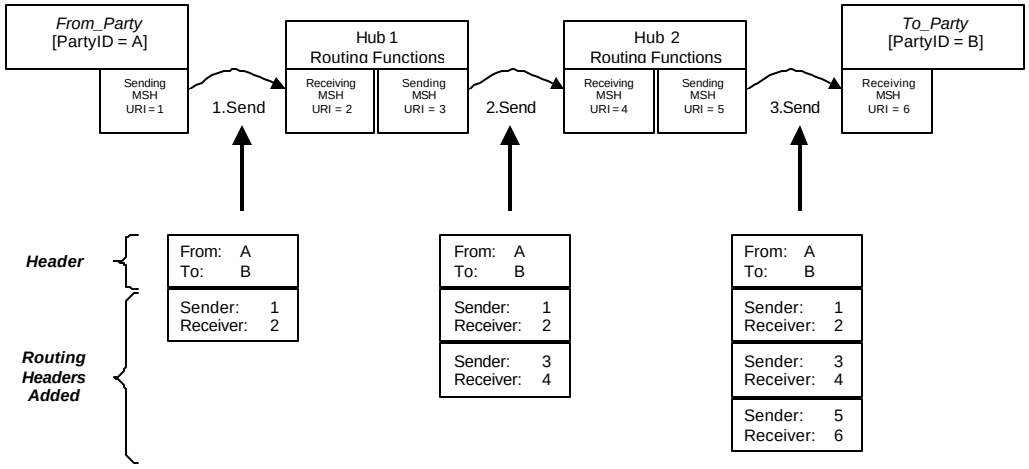
(2) The Hub receives the message, appends a new Routing Header for the next link and sends the message to the next Hub.

(3) [Step 2 might be repeated for additional Hubs.]

(4) The final Hub receives the message, appends a new Routing Header for the final link and sends the message to the Receiving MSH associated with the *To_Party*.

Figure 2-1 illustrates Simple Routing with Best Effort semantics, and no reliable messaging Acknowledgement messages are sent. Hubs are not required to persist the message as it transits the Hub. The sequence of transmissions is: *1.Send*, followed by *2.Send*, followed by *3.Send*.

Figure 2-2 Simple Routing – *Best Effort* Semantics



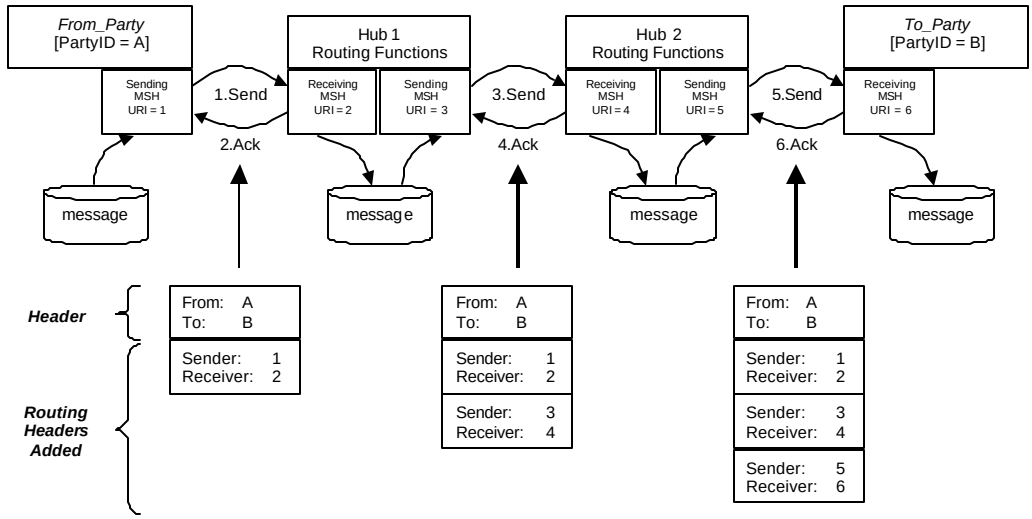
2.4.3 Simple Routing – *Once and Only Once* Semantics, Asynchronous ACK

Figure 2-3 illustrates Simple Routing with *Once And Only Once* semantics, where each Hub SHALL persist the message and SHALL return Message Service Handler Acknowledgement messages at each link, asynchronously with respect to Acknowledgement Messages received for later links in the network.

This method of Simple Routing is indicated by a parameter in the CPA or other suitable document.

Messages are persisted at each Hub until their receipt is acknowledged from the next Hub (which means that the message has been persisted at the subsequent Hub). The sequence of transmissions is: *1.Send*, followed by {*2.Ack* and *3.Send*}, followed by {*4.Ack* and *5.Send*}, followed by *6.Ack*. Within any Hub, the sending of an Ack to the previous link and sending of the retransmitted message to the next link are asynchronous.

Figure 2-3 Simple Routing – *Once And Only Once* Semantics, Asynch ACK



2.4.4 Simple Routing – *Once and Only Once* Semantics, Asynch ACK with Notification

Figure 2-4 illustrates Simple Routing with *Once And Only Once* semantics, where each Hub SHALL persist the message and SHALL return Message Service Handler Acknowledgement messages at each link, asynchronously with respect to Acknowledgement Messages received for later links in the network. In addition, in this method, the final Receiving MSH SHALL send a Delivery Notification Message, formatted as a Normal Message with ***OnceAndOnlyOnce*** semantics, to the first Sending MSH.

The Delivery Notification Message is formatted as follows:

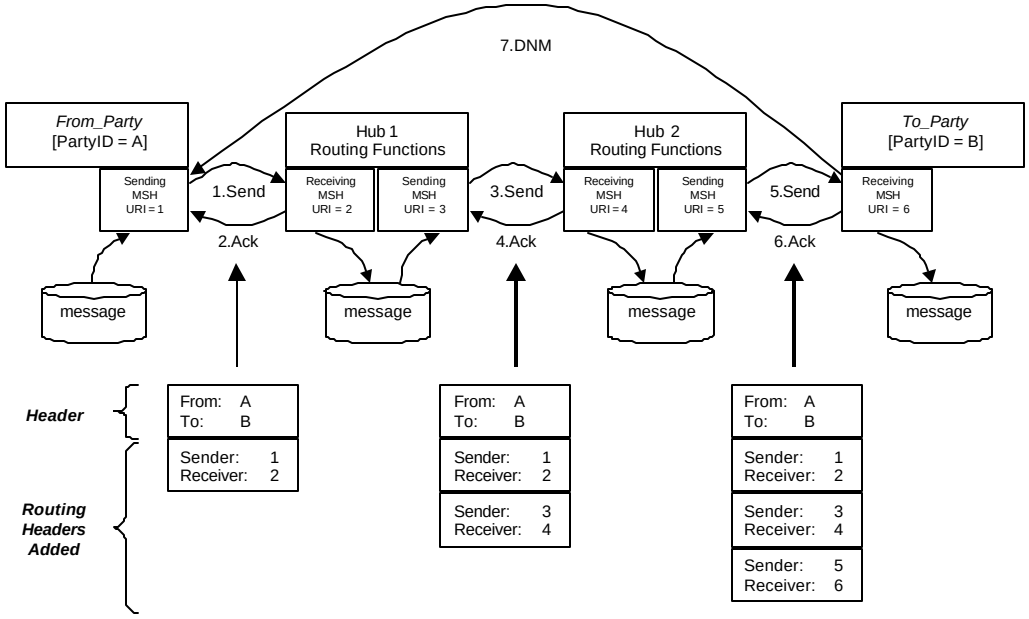
- ***MessageType*** = "Normal"
- There is no Payload and no business level response information
- ***From*** = URI of the final Receiving MSH in the sequence
- ***To*** = URI of the initial Sending MSH in the sequence
- ***TPAId*** and ***ConversationID*** = as shown in the message Header
- ***ServiceInterface***, ***Action*** and ***RefToMessageId*** are empty
- ***DeliverySemantics*** = OnceAndOnlyOnce
- Simple Routing Semantics may be any of the three reliable methods shown in this section

This method of Simple Routing is indicated by a parameter in the CPA or other suitable document.

Messages are persisted at each Hub until their receipt is acknowledged from the next Hub (which means that the message has been persisted at the subsequent Hub). The sequence of transmissions is: *1.Send*, followed by {*2.Ack* and *3.Send*}, followed by {*4.Ack* and *5.Send*}, followed by *6.Ack* and then *7.DNM*. Within any Hub, the sending of an Ack to the previous link and sending of the retransmitted message to the next link are asynchronous.

Editor Note 5:
- Because the returning DNM is sent with reliable message semantics, the following picture might not be clear: this DNM must traverse all the nodes back to the From_Party, and one of the three methods for confirming delivery must be used.
- Additional information is needed in the DNM Header or Payload so that the Sender can distinguish a DNM from a Normal Message that must be passed to a higher-level process. Possibly the fact that the To_Party UID is exactly the Recipient UID in the last Routing Header might be sufficient…?

299 **Figure 2-4 Simple Routing – *Once And Only Once* Semantics, Asynch ACK + Notification**



300

301 ### 2.4.5 Simple Routing – *Once and Only Once* Semantics, Synchronous ACK

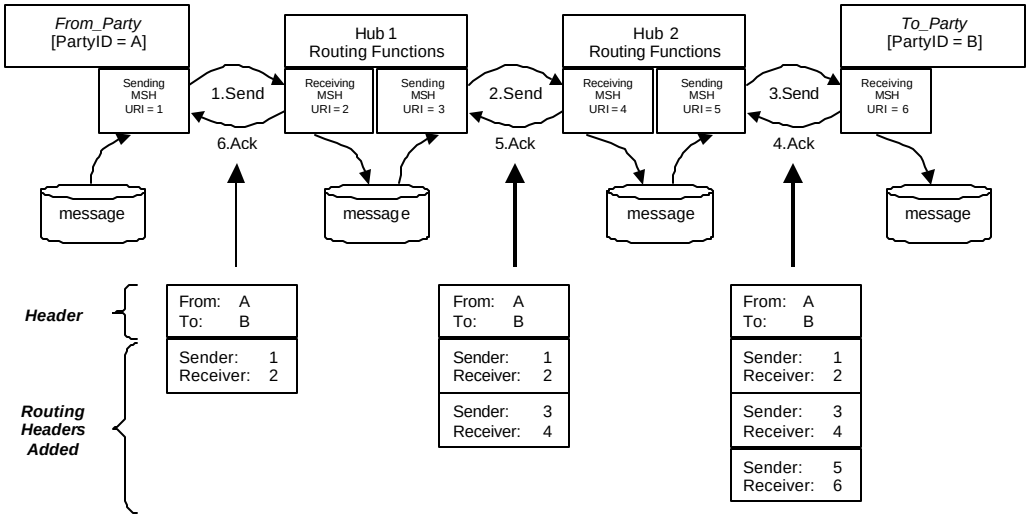
302 Figure 2-5 illustrates Simple Routing with *Once And Only Once* semantics, where each Hub
303 SHALL persist the message and SHALL return Message Service Handler Acknowledgement
304 messages at each link, synchronously with respect to Acknowledgement Messages received from
305 later links in the network.

306 This method of Simple Routing is indicated by a parameter in the CPA or other suitable document.

307 Messages are persisted at each Hub until their receipt is acknowledged from the next Hub (which
308 means that the message has been persisted at the subsequent Hub). The sequence of
309 transmissions is: *1.Send*, *2.Send*, *3.Send*, *4.Ack*, 5.*Ack* and *6.Ack*. Within any Hub, the sending
310 of an MSH-level Acknowledgement Message to the previous link SHALL not occur until an
311 Acknowledgement Message is received from the next Node. Thus, the initial Sending MSH will
312 not receive an Acknowledgement Message from the first Hub until the message has been
313 received and stored at the terminal Receiving MSH.

314

315 **Figure 2-5 Simple Routing – *Once And Only Once* Semantics, Synch ACK**



316

317 ### 2.4.6 Error Handling in Simple Routing – *Once and Only Once* Semantics

318 When message routing to a subsequent Hub is not available or fails at a particular Hub, the Hub's
319 Receiving MSH returns an ErrorMessage (Transient Error) to the Sending MSH instead of an
320 Acknowledgement Message. [David Burdett will write.]

321 *Editor Note 6: What are the new error messages needed for Simple Routing, for all four*
322 *cases?*

323 ## 2.5 Implicit Acknowledgements [David Burdett]

324 [Need to add – Might be a complete rewrite of Acknowledgements, to cover both Explicit and
325 Implicit Acks.]

326 ## 2.6 Modifications to Appendix E, Communication Protocol Interfaces
327 ## [David Burdett]

328 Cover relation to specific protocols (synchronous protocols, especially). Need to add FTP
329 references.

330 ## 2.7 Modifications to Appendix F, Requirements

331 Need to insert the Message Service Requirements Phases chart.

332 ## 2.8 Other Minor Changes to MS v0.8

333 MS Section 7.12.2

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

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

Line 825: Insert period after “attempts”

Figure 7-3: Change “Re-send the last message” to “Re-send the sent message”

MS Section E.4

Line 1706: Change “the last message” to “the sent message”

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 Message 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

Editor Note 7: This section will probably be deleted… Material in this section will be discussed in future TRP meetings as a likely base for further additions to the Message 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 Message Service parameters. This information may be determined in a number of ways, such as the CPA or some other method.

Table 4-1 Message 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 that was split. All the Normal Messages that 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 Message 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 Message Service that sends the message on the underlying message transport, and "*Receiver*" means the Message Service used by the Receiving Party. The term "*From-Party*" means the party that originally prepared the message and provided the message to its Message 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.

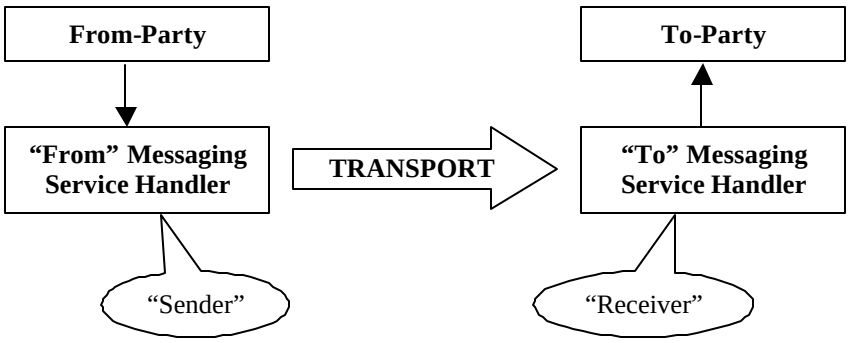


Figure 2-5-1: Simple Message Transmission

Reliable Messaging consists of the following basic concepts:

1) Messages are sent and received through Message 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 Message 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.

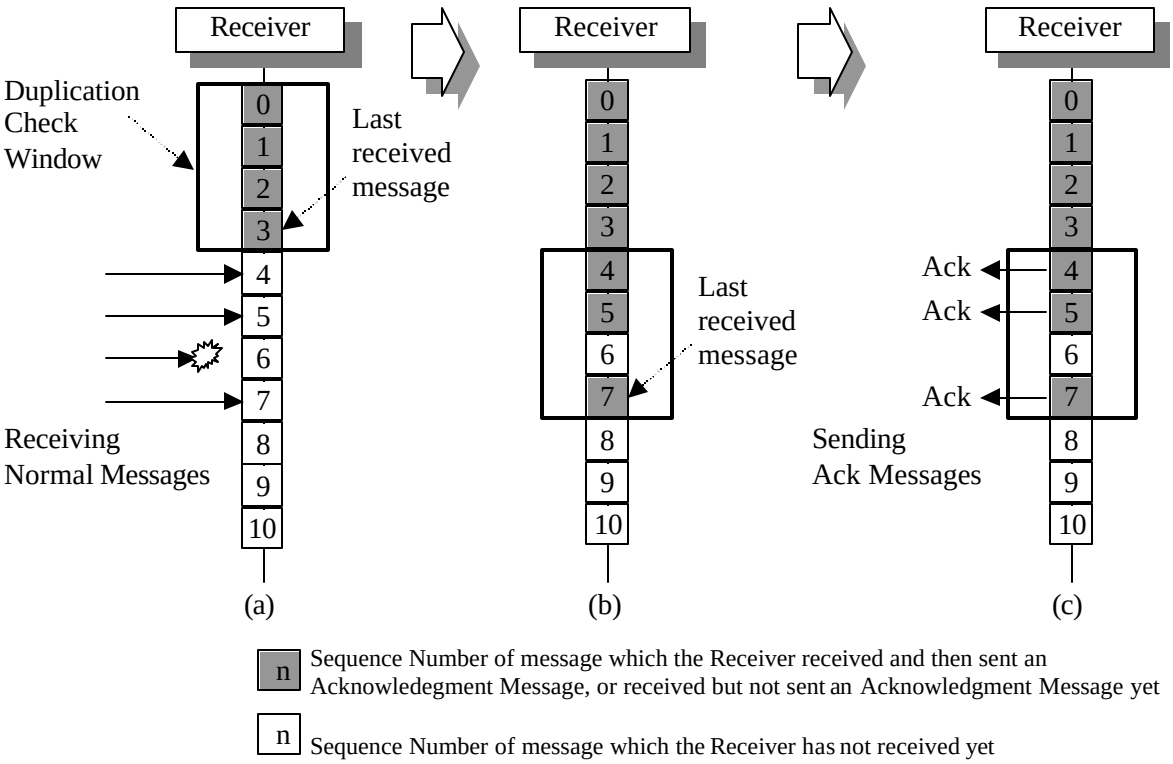


Figure 5-2 Sliding of Duplication Check Window (window size = 4, sliding = 4)

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.

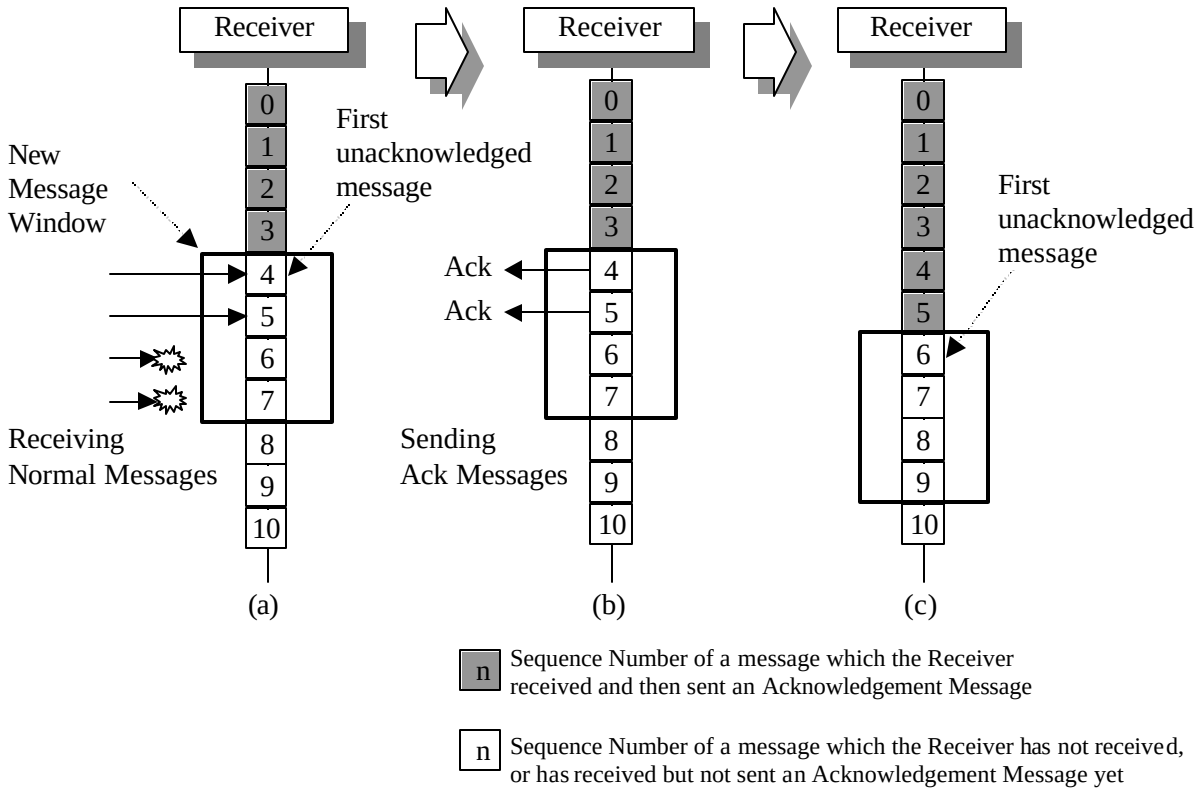


Figure 5-3 Sliding of New Message Window (window size = 4, sliding width = 2)

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 the Duplication Check Window, but it belongs to the 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)

With this process, the following Figure 5-4 shows the required actions when messages (on the left) arrive, given a particular Receiver state (on the right). The Receiver State consists of certain received and acknowledged messages (the shaded messages), unreceived or unacknowledged messages (the unshaded messages) and the two Windows. The arriving messages carry sequence numbers that are compared to the Receiver State.

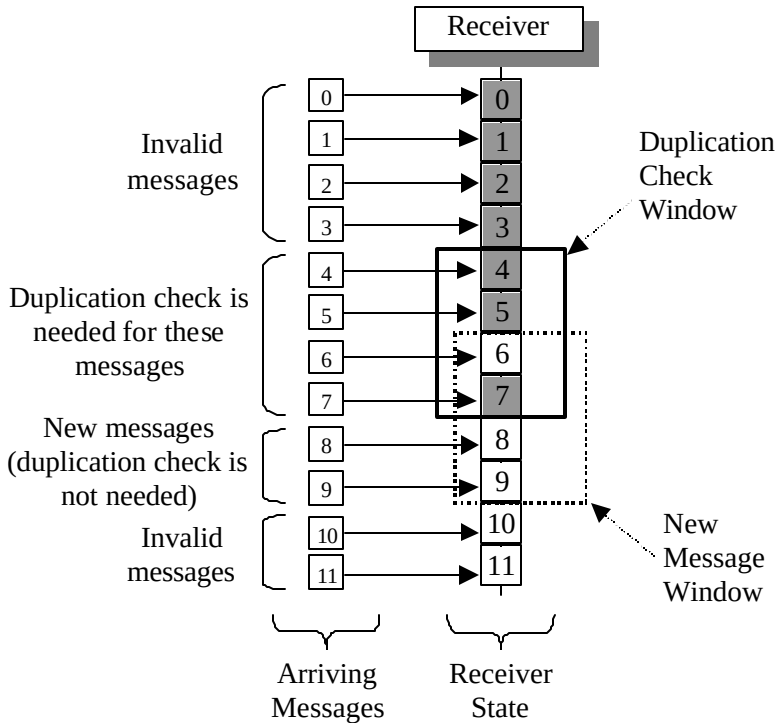


Figure 5-4 Checking arriving messages, given a particular Receiver State

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

…And the TRP team.