Operational Reliability for EDIINT AS2
<draft-duker-as2-reliability-12.txt>

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.html
The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Any questions, comments, and reports of defects or ambiguities in this specification may be sent to the mailing list for the EDIINT working group of the IETF, using the address <ietf-ediint@imc.org>. Requests to subscribe to the mailing list should be addressed to <ietf-ediint-request@imc.org>.
Abstract

The goal of this document is to define approaches to achieve a "once and only once" delivery of messages. The EDIINT AS2 protocol [AS2] is implemented by a number of software tools on a variety of platforms with varying capabilities and with varying network service quality. Although the AS2 protocol defines a unique "Message-ID", current implementations of AS2 do not provide a standard method to prevent the same message (re-transmitted by the initial sender) from reaching back-end business applications at the initial receiver. TCP underpinnings of HTTP over which AS2 operates generally provide a good quality of network connectivity, but experience indicates a need to be able to compensate for both transient server and socket exceptions, including "Connection refused" as well as "Server busy."

In addition, difficulties with server availability, stability, and loads can result in reduced operational reliability. This document describes some ways to compensate for exceptions and enhance the reliability of AS2 protocol operation. Implementation of these reliability features is indicated by presence of the "AS2- Reliability" value in the EDIINT-Features header.

Intended Status

The intent of this document is to be placed on the RFC track as an Informational RFC.

Feedback Instructions:
NOTE TO RFC EDITOR: This section should be removed by the RFC editor prior to publication.

If you want to provide feedback on this draft, follow these guidelines:

- Send feedback via e-mail to the ietf-ediint list for discussion, with "AS2 Reliability" in the Subject field. To enter or follow the discussion, you need to subscribe to ietf-ediint@imc.org.

- Be specific as to what section you are referring to, preferably quoting the portion that needs modification, after which you state your comments.

- If you are recommending some text to be replaced with your suggested text, again, quote the section to be replaced, and be clear on the section in question.
1. Introduction

AS2 Reliability has the goal of ensuring that the AS2 protocol succeeds in exchanging business data payloads exactly once, provided that the network routing and transport (IP and TCP) layers are fully functional. That is, the goals for reliability are, first, that errors associated with HTTP server operation and server initiated sub processes do not prevent delivering messages or their receipt responses (MDNs) at least once and, second, that retry or resending operations made to compensate for these errors do not result in the same message payloads being submitted for further processing more than once.

1.1 Key Word Conventions

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 [RFC2119].

1.2 Terminology and Scope Limitations

Initial Sender: The AS2 application ("sending implementation") which transmits the Message containing the business payload to the "initial receiver".

Initial Receiver: The AS2 application ("receiving implementation") which receives the Message containing the business payload. The initial receiver sends a MDN back to the initial sender.
Message: The business payload as embedded in a "wire" format and ready for transmission by a transfer protocol (including all MIME wrappings, headers, encodings, and security transformations).

Message Disposition Notification (MDN) - The Internet messaging format used to convey a receipt. This term is used interchangeably with receipt. See [RFC3798].

Message Identifier (Message-ID) - A globally unique identifier for a message. The sending implementation MUST guarantee that the Message-ID is unique for a given AS2-To and AS2-From pair. See [RFC5322].

Message Integrity Check (MIC) - The name given to the quantity computed over the body part with a message digest or hash function, in support of the digital signature service.

Payload: The business data exchanged between business applications.

Retry: When attempting to send a message using the POST method, the initial sender can encounter transient exceptions that result in a failure to obtain a HTTP status code or a transient HTTP error such as 503. "Retry" is the term used in this document to refer to an additional POST of the same message, with the same content (including the Message Integrity Check value) and with the same Message-ID value. A retry can occur after a few second delay or on a schedule. Retrying ceases when a message is sent (which is indicated by receiving a HTTP 200 range status code), or when a retry limit is exceeded. Concurrency is not allowed for retries for a given message.

Resend: The AS2 protocol normally requests a (signed or unsigned) MDN response in the HTTP response message body. When a MDN is not received in a timely manner, the initial sender may choose to resend the original message. Because the message has already been sent, but has presumably not been processed according to expectation, the same message, with the same content and the same Message-ID value is sent again. This operation is referred to as a resend of the message. This document will suggest guidelines to prevent AS2 software implementations that receive duplicate messages from distributing that message to back-end business applications, as well as guidelines on resend intervals and resend counts for various modes of AS2 operation. Resending ends when the MDN is received or the resend count is reached.

Resubmit: Accidents happen, and possibly the remote system will need to get a new copy (a "resubmit") of a message that was previously exchanged. In addition, neither Resending nor Retrying continue forever, but the data may still need to be exchanged at a later time,
so a message may need to be resubmitted. When data that failed to be exchanged or was exchanged but lost is resubmitted in a new message (with a new Message-ID value, and possibly with new timestamps, and boundary delimiters), it is called resubmission. Resubmission is normally a manual compensation and is not discussed in this document further.

Duplicate: Duplicate copies of the same message are messages between the same AS2-To and AS2-From organizations which have the same Message-ID. This document will recommend ways to respond to duplication in messages as indicated by messages being received with the same Message-ID values. These duplicates arise in some cases when retries and/or resends are allowed, and success indicators (such as HTTP "200 OK" or MDNs) are not received by the initial sender.

2. AS2 Modes of Operation

There are many user selectable options within the AS2 protocol, but there are two main modes of operation commonly used that differ in how the MDN is returned to the initial sender.

Transferring data via AS2 involves two organizations that are identified using AS2-To and AS2-From header values. The initial sender places its identifying value in the AS2-From header and POSTS an AS2 formatted message to the organization whose value is found in the AS2-To header. The initial sender can indicate that it wants to receive its MDN on a different connection by including a header, Receipt-Delivery-Option, with an http, https, or (rarely) mailto URL as its value. Use of the mailto URL is not further considered in this document.

When the MDN is requested to be returned on a distinct TCP connection using the included URL, the AS2 operation mode is called "asynchronous." An asynchronous MDN is returned when the initial receiver has the information and resources available to do so, and the formatted MDN is POSTED to the delivery URL with the initial sender identifier as the AS2-To value and the original recipient as the AS2-From value.

The AS2 protocol does not specify how asynchronous MDN delivery is scheduled; it is left to the receiving implementation to determine how MDNs will be returned. The protocol in effect uses the "200" level status code to determine that the initial message has been sent. The initial sender then enters into a state of "waiting" or "expecting" a MDN to be received.

While it is expected that a MDN be returned in a timely fashion, there has not been an agreed upon deadline, and receiving implementations have had flexibility in scheduling return. It is therefore possible that an indefinite waiting period occur when a MDN...
is "lost" (for whatever reason). Most implementations do eventually time out this wait for a lost MDN. Implementations also try to recover from this protocol failure by resending the original message. Under certain circumstances, therefore, duplicate messages can arrive at the recipient.

When no Receipt-Delivery-Option is included in the original message, and a MDN is requested (whether signed or unsigned), the AS2 operation mode is said to be "synchronous." This means that the protocol requires that the MDN arrive back on the same TCP connection, and in the MIME body of the HTTP reply message. When using the synchronous mode, there can be a timeout by either side waiting for the HTTP reply, and that timeout usually aborts the protocol by closing the connection. In such a case, the message has not been successfully sent, so the payload from the message should not be distributed to a back end business application, and the message can only be retried (or perhaps resubmitted, an option not discussed here). Therefore resend compensation will not be discussed for the synchronous mode, but instead retry compensation will be the main topic.

3. AS2 Reliability Concepts

Introducing timeouts for various wait states does not in itself promote the goal of delivering the message. Instead it simply cleans up the protocol state machine so that it can be restarted. Delivery thus requires that the message be sent again either in a retry or a resend operation.

It is important to have precise understanding of what "sending the same message again" means. When exchanging business data, there are both payload transaction identifiers and message identifiers. In AS2, the message identifier is the value of the Message-ID header, and the procedures described below will assume that each message has a unique Message-ID value in the message headers. Implementations MUST NOT change any content of the message when retrying or resending. This requirement allows implementations to use Message-ID values to detect duplicate messages, and avoid sending their payloads to the internal business applications that process the business data. [Note: duplicate payloads could still be sent, but they would have to be sent in different messages. Implementations MAY provide duplicate detection for payloads as well. Implementations will need to be informed about the specific business data (such as the interchange control numbers of the ISA [or UNB] header of ANSI X12 [or EDIFACT] payloads, or the InstanceIdentifier in the DocumentIdentification block of a Standard Business Document Header (SBDH) used in some XML messages) in order to offer a service for duplicate payload detection.]
4. Basic Initial Sender Operation

Sending implementations MUST be able to configure how much time is allowed before closing an unresponsive HTTP connection prior to receiving the HTTP status code reply.

Sending implementations MUST retain an exact copy of every message (including the Message-ID value) which is attempted to be sent or is sent. Repackaging a payload will not necessarily produce the same message, because MIME boundary delimiter values, timestamps, and other dynamic data used in assembling messages may not be the same. It is implementation dependent how this copy is retained.

Several parameters relating to the number and schedule for retries and resends need to be described. Implementations MUST allow the configurability of these parameters but are allowed to use an implementation dependent "back off" algorithm for lengthening intervals.

5. Initial Sender Operation for Retry Situations

Relevant conditions:

Connection refused:
Closed connection prior to HTTP reply received.
Transient exception (such as 503) in HTTP reply codes

Behavior and Capabilities:

- Maximum number of retries.

Sending implementations MUST be capable of configuring either a maximum number of retries, and/or a total elapsed time for the retry duration. Sending implementations MUST stop retrying when a successful send occurs, when reaching the total retry number, or when reaching the total elapsed time for retry duration. The count of retries SHALL begin with the first retry counting as the first one. So, if five retries are allowed, a total of 6 attempts can be made to send the message using the retry operation, provided retry does not attain success or otherwise stop.

- Minimum initial interval of retries.

Sending implementations MUST be capable of configuring a minimum retry interval. The minimum interval pertains to the first retry, and (depending on an implementation dependent algorithm) remaining ones. The function governing lengthening intervals between retries MUST increase monotonically (stay the same or increase). The minimum retry
interval begins after the failure to send is determined.

- Maximum retry duration

Sending implementations MUST be capable of configuring either a period within which all retries of the same message are attempted and/or a maximum number of retries. After this period expires, a payload would have to be resubmitted to be exchanged. This interval begins after the initial attempt to deliver is known not to have succeeded. (Success is marked by the reception of a 200 level HTTP status code.) A retry in progress when the maximum retry interval is reached does not have to be stopped. The retry process may exceed the maximum retry duration before the maximum number of retries is reached.

Implementations are permitted to restrict the range of values for the configurability of the above maximum and minimum values. Implementations should engage some kind of "back off" algorithm to avoid exacerbating resource use on heavily loaded servers. (High workloads are often behind the "connection refused" or "server busy" error conditions.) Implementations are also allowed to alter ranges of configurability for one range of value based upon a user selection of some other maximum or minimum value, but no requirements are made on implementations as to how these restrictions are defined.

- Diagnostic logging

Sending implementations SHOULD keep a record of the condition that caused a failure in sending a message as this log may help identify a cause of and a solution to a sending failure. For example, if the time involved in all retries of sending a message has approximately the same value and the error is reported as an unexpected close in a connection, then a review of the values governing closing connections on both sides, followed by their adjustment can be useful. Of course, other factors may be involved-- ranging from network congestion to unpredictably large payloads to be exchanged-- that may also need further tweaking.

6. Initial Sender Operation for Resend Situations

Because successful delivery of the message in the synchronous MDN mode implies that the initial sender must receive a response which contains both a HTTP 200 level status code and a MDN in the body of the response, the resend operation is not defined for the synchronous MDN mode of operation.

Relevant conditions:

MDN not received after a resend interval has expired.
Behavior and Capabilities:

Sending implementations MUST note when successful sending has occurred (when using asynchronous MDNs), so that resending may conform with the Resend configuration parameters.

- **Maximum number of Resends**
  
  Sending implementations MUST be capable of configuring either a total number of resends, and/or a total elapsed time for the resend duration. Sending implementations MUST stop resending when a MDN is received, when reaching the total number of resends, or when exceeding the total allowed time for resends.

- **Minimum interval between Resends**
  
  Sending implementations MUST be capable of configuring an interval of time separating resends. Implementations MUST ensure for a given message that retry and resend operations are not interwoven. For example, during a resend attempt, retries could occur. In this situation, the sending implementation MUST ensure that another resend does not start while retries are still occurring.

- **Maximum resend duration**
  
  Sending implementations MAY configure a total duration for resend operations and MUST NOT start additional resend attempts when that duration is exceeded. This interval begins when the first successful send operation occurs. (Success for the sender is determined by its reception of a 200 level HTTP status code.)

7. Initial Receiver (Server) Operation

Behavior and Capabilities:

Receiving implementations MUST return an appropriate MDN (when a MDN is requested) even when a message is detected as a duplicate.

Duplicate elimination is based on Message-ID values. Receiving implementations MUST retain Message-ID values for the pairs of organizations exchanging data, beginning with the successful receipt of the message. Successful receipt may possibly occur from the receiving implementation’s point of view even if the initial sender does not see the HTTP reply status code, thereby causing the initial sender to initiate a retry.
Receiving implementations SHOULD retain Message-IDs until the initial sender has exhausted all retry and resend durations. Since the receiving implementation may not know these durations, the receiving implementation MUST retain each Message-ID for a minimum of five days unless users explicitly agree to configure the time period for a different time period.

Receiving implementations MUST be configurable so that backend business applications are not sent the contents (payload) from the same message more than once. It is recommended that implementers make this the default. (However different messages, as determined by their Message-ID values, may still send the same payload contents.)

Receiving implementations MAY be configurable so that backend business applications do not receive the same payload more than once (for mutually agreed upon business data types). This functionality is, however, not specified in this document.

8. Additional Reliability Considerations with Synchronous MDNs

There can be combinations of server and client behavior that, even when the network is fully functional, still interfere with reliable AS2 data exchange.

When clients, their operating systems, or intermediate HTTP relay agents choose to close TCP connections before the server has had time to complete the processing needed to create the reply, repetition of the client HTTP request need not lead to a successful outcome, no matter how often the retry operation is repeated.

Timeouts while waiting for a HTTP response may themselves create errors. The intent of these timeouts is to avoid waste of resources tied up in possibly indefinite delays ("hangs") in HTTP response. However, with short timeout periods and for very large files, the security processing required to be able to form the MDN may, especially under very heavy loads, lead to a particularly bad outcome. The initial sender may attempt to repeatedly retry its HTTP POST creating additional load with no better outcome (timeout before MDN reply is received).

In order to avoid this timeout situation, receiving implementations MUST support the HTTP 102 (Processing) status code [HTTP-Codes]. The 102 (Processing) status code is an interim response used to inform the client that the server has accepted the complete request, but has not yet completed it. This status code SHOULD only be sent when the server has a reasonable expectation that the request will take significant time to complete. As guidance, if a method is taking longer than 20 seconds (a reasonable, but arbitrary value) to process, the server MUST return a 102 (Processing) response. The server MUST also send a final status code indicating success (200 range) or otherwise after the request has been completed, as required by the HTTP protocol.
Receiving implementations MUST NOT delay the sending of a MDN in order to allow the message payload to be processed by a back end application. The HTTP 102 status code is intended to keep a session active while the receiving implementation is processing the message to create a MDN.

9. Security Considerations

None

10. IANA Considerations

None

11. Acknowledgements

The authors wish to extend gratitude to the following individuals,
- John Koehring of Axway for his comments on early drafts.
- Yury Bogucharov of Microsoft for his help clarifying retry and resend approaches (clarified in version 4 of this draft).
The authors also wish to thank all of the participants in the Drummond Group AS2 Reliability Interoperability testing for their feedback and helpful comments.

Normative References


Informative References


Authors’ Addresses

John Duker
Procter & Gamble
2 Procter & Gamble Plaza
Cincinnati, OH 45202 USA
Email: john.duker.ecom@gmail.com

Dale Moberg
Axway Inc.
8388 E. Hartford Drive, Suite 100
Scottsdale, AZ 85255 USA
Email: dmoberg@us.axway.com

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.iert.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

All IETF Documents and the information contained therein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION THEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.