Report on APEL and Messaging

Introduction

APEL is an accounting system which collects information about grid computing jobs and summarises this information into a useful form. It operates for the European Grid Initiative (EGI), with client software which is released as part of the European Middleware Initiative (EMI). It operates in conjunction with the EGI Accounting Portal to publish the collected data.

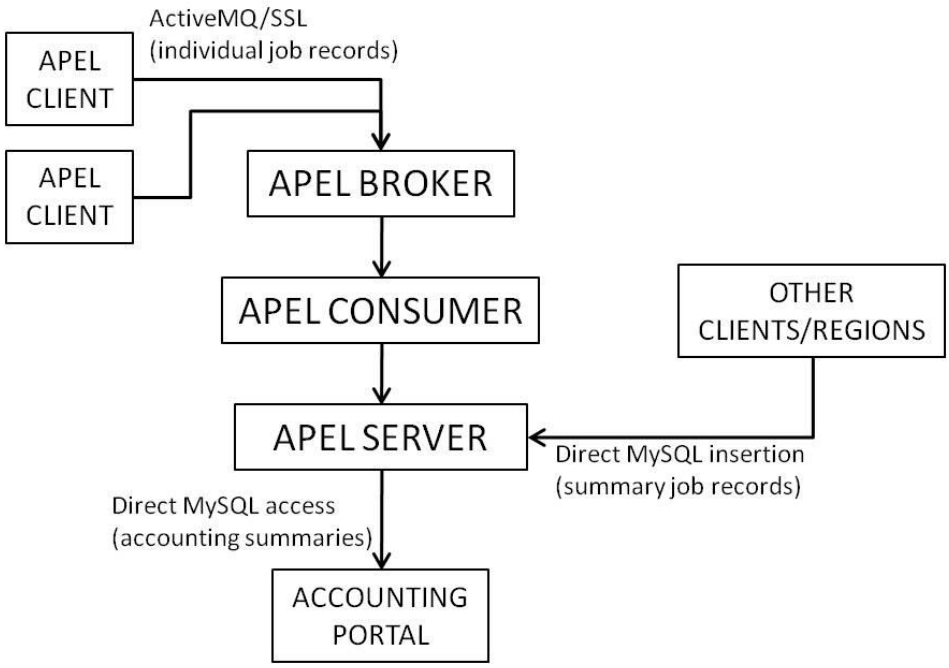
A key part of the APEL infrastructure is the transport of information from its client software, which runs on grid sites, to the central APEL repository hosted in the UK. Since 2009, we have used messaging to do this transport. This document describes our experience with messaging and how it has led to the design of the new APEL system.

How APEL works

APEL collects accounting data from sites participating in the EGI and WLCG infrastructures as well as from sites belonging to other Grid organisations collaborating with EGI, including OSG, NorduGrid and INFN. This enables international virtual organisations (VO) to gather all their accounting data in one place for easier visualisation and data mining. With the changes described here we expect the repository's functionality to be extended to include data from Unicore and Globus sites in EGI and to support other types of usage records than cpu.

The accounting data is gathered from all the different sensors into a central accounting database where it is processed to generate statistical summaries that are available through the EGI Accounting Portal.

Statistics are available for view in different detail by Users, VO Managers, Site Administrators and anonymous users according to well defined access rights.



Accounting Data Description

The APEL Accounting System distinguishes two different types of accounting records between the APEL client and the server. A 'job record' contains information about one job run on the grid. Clients (that is, grid sites) using the APEL software send details about every job to the APEL server. A 'sync record' contains information about the accounting information stored locally to the site. This is used by APEL to check that the information received by the server is the same as that stored at the site. The purpose of the sync records is only to ensure the integrity of the data transfer. The job record and sync record schemas are listed in Appendix A.

The job record schema is loosely based on that of the OGF Usage Record. The names of the fields are different, but there is a simple mapping to convert between the job record schema and a subset of the OGF UR.

A third schema is used by APEL – the 'summary record'. This is used to store summaries of each month's data for different sites, VOs and optionally users. This summarized information is more useful and easier to work with. It is sent to the Accounting Portal, which visualizes the information and makes it available to end users.

Sites which do not use the APEL client software have a different publication method. Different accounting systems each have a dedicated table in a MySQL database in the APEL server with the schema for summary records. There are slight variations between these different dedicated tables. Another variant of this schema is used between the APEL server and the EGI Accounting Portal.

Transport of Accounting Data

Accounting Data is transported from the different clients, either sites or other infrastructures, into the Central Accounting Database.

The summarised data is transported from the Central Accounting Database to the Accounting Portal.

Transport Requirements

Data encryption: Data identifying an individual should not be sent across the wide area network in plain text.

Synchronisation checks: The client must be able to check that the accounting data sent has reached the server successfully.

Republishing of data: Clients must be able to resend updated accounting data.

Current transport system between APEL client and APEL server

The messaging system currently used in production is based on ActiveMQ via SSL.

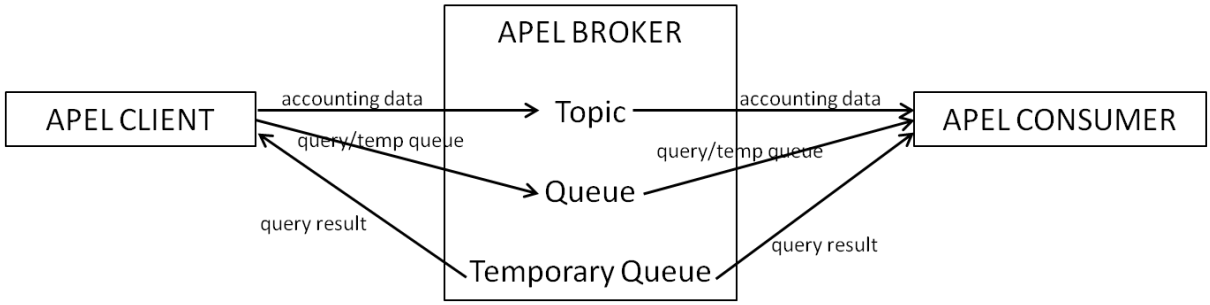


Figure 1

The current system makes use of three different services:

A publisher (APEL client)
A broker
A consumer

The APEL Client runs on the site's glite-APEL box, connecting to the APEL ActiveMQ broker service at RAL. The APEL client will act as a producer, publishing accounting records as ActiveMQ messages to a topic on the broker.

During this time, the APEL consumer service subscribes to the same topic. This consumer machine will receive the published accounting record messages and add them into a local MySQL database.

After publishing of records, the APEL client will check how many records have been accepted into the system by sending a message to a queue on the broker, together with the name of a temporary queue that will be used by the consumer to send the reply to the check. Check messages published to the queue are received by the consumer, processed and reply to on the defined temporary queue.

Authorisation/authentication

The APEL broker is configured to use certificate based authentication. This is a non-standard addition to the ActiveMQ configuration.

The broker will only accept a connection if the client certificate is trusted by one of the CAs in the trust store. The trust store contains the certificates by CAs belonging to the International Grid Trust Federation (IGTF).

A client will only be granted access to the broker if their specific certificate DN appears in a defined users configuration file. This file is updated several times a day by downloading the list of host DNs for all glite-APEL service endpoints in GOCDB.

Any client that wishes to publish accounting data to the APEL repository is therefore required to register its glite-APEL box in GOCDB, specifying the correct host DN.

Transport between external sites/regions and APEL server

As previously explained, external sites and regions have direct MySQL access (username/password) to dedicated tables in the APEL server.

The external sites are responsible for populating these tables with the correct accounting data for their sites. The summarising processes running on the APEL server will merge these data with the data received from the APEL clients to create the summary tables provided to the Accounting Portal.

Transport between server and Accounting Portal

The Accounting Portal has direct MySQL access (username/password) to the summaries database in the APEL server. Daily, the portal downloads the full set of summaries from the APEL server into their own database.

Some statistics

The APEL database currently accounts for more than 1,000M job records since 2004.

Around 230 sites publish daily to the APEL consumer using the APEL client. Another 106 sites insert summarised accounting data into the APEL server from other regions like OSG (US), INFN (Italy), IN2P3 (France) and SGAS (Scandinavia).

Table 1 shows statistics on the number of sites and records received by APEL since 2004.

Year	No of Records per Year	Avg Records per Month	Avg Records per Site per Month	No of Sites
2004	893320	74443	17179	52
2005	6038055	503171	34902	173
2006	19370172	1614181	69427	279
2007	47522651	3960221	158409	300
2008	95391552	7949296	289065	330
2009	196485795	16373816	505105	389
2010	403507843	33625654	998782	404
2011	360334215	45041777	973876	370

Table 1

Problems with the current system

1. Support for ActiveMQ/SSL: ActiveMQ via SSL is not widely used so finding support and help seems to be more challenging that it is for other protocols like STOMP.
2. Integration with EGI production messaging system: Despite extensive work by the EGI messaging team, it has not been possible to integrate the ActiveMQ/SSL interface with the current messaging infrastructure in production. APEL requires a dedicated and standalone broker service.
3. Several transport mechanisms: There are currently three transport mechanisms working in parallel in the APEL system.
4. The APEL client sends accounting data to the APEL server via ActiveMQ/SSL.
5. The non-APEL clients connect to a dedicated MySQL table in the APEL server where they insert their data directly.

6. The Accounting Portal has direct access to the APEL server summary database to download the data into their own servers.
7. Inadequate message format: The message format used by the APEL client is currently an SQL query. The obvious security implications of having direct SQL queries as messages are overcome by parsing the messages at the central APEL consumer.
    a. There is no version information on the message so updates to the format are difficult to introduce.
    b. At the same time, any changes in the server database schema require changes in the message format, which would require instant updates of all clients.
8. Inefficient encryption of data: In the current system the UserDN field for every record is encrypted using an asymmetric private-public key encryption together with a randomising function. This has proven to be a lengthy process at both client and server ends.
9. Mapping between data and sender lost: Even though there is an authentication/authorisation mechanism in place between client and server, there is no recorded link between the publishing site and the records sent. Once the records have been received by the APEL consumer, there is no mechanism to identify the site or host that sent the records.
10. Joined data transport and storage: The APEL consumer receives the records sent by the sites and inserts them directly into a MySQL database. At the same time, sites confirm that their publishing has been successful by sending a SELECT query into the consumer database. This join between transport and database has some important disadvantages. Any downtime in the MySQL database will therefore stop publishing of all sites. At the same time, the MySQL database needs to be constantly maintained and cleaned up, as any increase on query time will slow down publishing by all sites, potentially causing timeouts and client errors.

Design of the new APEL system

The problems found in the existing APEL system led to a few key design decisions.

- Using the STOMP protocol: this is simple, widely supported and recommended by the EGI message broker network. It supports both SSL and username and password authentication. This decision meant that we could use the EGI broker network
- Decoupling of the transport mechanism from other components. As well as simplification, it allows the same transport mechanism to be used throughout the system, and replacement of direct database access
- Encryption of entire messages: building encryption into the transport layer is much simpler than encrypting and decrypting one specific field. However, this doesn't necessarily speed up the process; the speed of any new design requires monitoring
- Use of IGTF certificates to encrypt and sign messages. These are widely available in grid computing; the signature of the messages also allows tracing of the sender of those messages
- A new message format reflecting the experiences of the OGF UR-WG and modifications proposed by the compute group of EMI (CAR).

As well as these, it was important to keep high data integrity between client and server. Previous APEL versions have had transport mechanisms which 'lost' records in transit. The only solution to this was to republish previous data until the server and the client databases matched. This unreliability was the cause of significant support work and higher load on the infrastructure.

Description of the new system

Accounting Data Description

The new APEL Accounting System distinguishes three different types of accounting records between any client and the server: Individual Job Accounting Record; Summary Accounting Job Record; and Sync Record. These are described in more detail in Appendix C.

Another set of schemas is used for sending messages between the APEL server and the EGI Accounting Portal.

ActiveMQ/STOMP Transport

The messaging for the new APEL system is done with ActiveMQ via STOMP.

Transport of records is done by the Secure Stomp Messenger (SSM) which is a tool developed at RAL and written in Python.

SSM (Secure Stomp Messenger)

The SSM python package is designed to send files using the STOMP protocol. It is a general-purpose messaging system, designed for APEL but usable by any system that needs to use ActiveMQ/Stomp for peer-to-peer communication.

Each file is encrypted using the certificate of the consumer to which it will be sent, and signed using the host machine's certificate. On receipt, the file is decrypted and placed on the receiving machine's filesystem with a second file containing the sender's certificate DN.

SSM Operational Overview

An SSM object can be created as a producer, a consumer or both.

Producers send messages to a specified topic on an ActiveMQ server. Producers may only transmit an encrypted message against the X509 certificate of *one* consumer on a single topic. Consumers listen to a nominated topic on an ActiveMQ server. Consumers may only listen to a single topic, but may accept messages sent by multiple producers.

The SSM is associated with a MessageDatabase object, which implements a standard API (the default Message database object supplied with SSM is a simple file-system based message store). A producer SSM queries the MessageDatabase object for outgoing messages, if there are any to send then the SSM attempts to send the message to the topic. For encryption, the producer needs the consumer's X509 certificate: this is obtained during a certificate request/response cycle at least once in the lifetime of the SSM object. The producer will use a local certificate to sign the message, and the consumer's certificate to encrypt it. The producer sends the message via Stomp to the specified topic, and then waits for an

acknowledgement from the consumer. After an acknowledgement the producer removes the outgoing message from the MessageDatabase, and prepares to send the next message. If there is no acknowledgement, then the producer leaves the message in the MessageDatabase.

A consumer SSM listens to a single topic for messages sent by producers. The messages are decrypted using the consumer's X509 certificate, and then the signature of the producer is verified and checked against an Access Control List of valid producer DNs. If successful, the message is sent to the inbox of the MessageDatabase. A consumer SSM acknowledges received messages by sending an Ack message to the producer, via a topic nominated by the producer (contained in the header of the message). The consumer SSM tracks the MD5 checksums of received messages, so that it can discard messages that have been received twice in succession.

There is a re-acknowledgement mechanism between the producer and consumer; the producer will send the MD5 sum of the last message that the consumer acknowledged, in the header of the next message it sends. This tells the consumer that the ack has been received, so that the producer will therefore not attempt to resend that message - the consumer can then forget the MD5 sum of that particular message.

This means that if we put the messages in the appropriate place on the filesystem and the SSM is running, it will deliver the messages to a second SSM via the broker, and place the same messages in the output directory along with a second file containing the DN of the sending certificate.

The first part of a session involves the consumer sending its public key to a publisher. Every message is encrypted before it's sent, using the consumer's public key, and decrypted on receipt by the SSM. The publisher waits for acknowledgment of each message before it attempts to send the next. Detailed diagrams of the message sequences are in Appendix D.

Evaluation of new APEL system

Resolution of problems with the current system

The new design resolves most of the identified problems with the existing APEL system.

- The transport mechanism is supported by the EGI broker network, and this means that authentication and broker administration are handled externally
- The transport mechanism is independent of each use case, so can be used in all parts of the APEL system which require messaging
- No external connections to an APEL database are necessary
- The messaging system is decoupled from server processes and can remain in operation independently
- It is easy to identify the source of any message
- New message formats are well defined
- Both client and server are absolutely sure when a message has been delivered.

Performance and reliability testing

Need to fill in this section.

Possible limitations

- Although STOMP is widely used, the fact that messages are signed and encrypted and the requirement for preliminary messages means that it is not straightforward to develop new clients to interact with the SSM.
- Decoupling of the message system from any other processes means that immediate validation of the contents of a message is not possible. A separate mechanism is required to inform clients of rejected messages.
- It is now possible to connect to the EGI network of brokers using STOMP and SSL. In this case, direct encryption of the messages using the server's certificate is unnecessary.

Conclusions

The new design has simplified the APEL server considerably. Many lessons have been learnt from past APEL versions, and significant improvements have been made.

The SSM is a key component of the new design, and has simplified data transfer for all parts of the APEL system. It transfers information quickly, securely and reliably from many clients to a central server, using the infrastructure available in the EGI project. Having a single mechanism reduces workload for APEL managers. For APEL purposes, it has fulfilled its requirements well.

The main disadvantage of the SSM is the level of customisation on top of the transport protocol. This means that is more difficult to understand and develop against than a program which uses STOMP as simply as possible. This complication may deter potential users from adopting this program.

Appendix A: Existing data schemas between APEL clients and server

Individual Job Record

A complete grid job accounting record describes the resources consumed by a single executing job.

Key	Value	Description	Mandatory
RecordIdentity	String	Unique identifier for the job	Yes
ExecutingSite	String	GOCDB official sitename	
LocalJobID	String	Batch System Job ID	
LCGJobID	String	WMS Job ID	
LocalUserID	String	Local username	
LCGUserID	String	User's X509 DN	
LCGUserVO	String	User's VO	
ElapsedTime	String	Wallclock time duration (ISO 8601 format)	
BaseCpuTime	String	CPU time duration (ISO 8601 format)	

ElapsedTimeSeconds	int	Wallclock time for the job (seconds)	
BaseCpuTimeSeconds	int	CPU time for the job (seconds)	
StartTime	String	Start time of the job (local time)	
StopTime	String	Stop time of the job (local time)	
StartTimeUTC	String	Start time of the job (UTC time)	
StopTimeUTC	String	Stop time of the job (UTC time)	
StartTimeEpoch	int	Start time of the job (epoch)	
StopTimeEpoch	int		
ExecutingCE	String	Head node where the job was submitted	
MemoryReal	int	Real memory consumed by job (kbytes)	
MemoryVirtual	int	Virtual memory consumed by job (kbytes)	
SpecInt2000	int	SI2K benchmark	
SpecFloat2000	int	SF2K benchmark	
EventDate	Date	Stop date of the job YYYY-MM-DD	
EventTime	Time	Stop time of the job HH:MM:SS	
MeasurementDate	Date	Timestamp entry was created YYYY-MM-DD	
MeasurementTime	Time	Timestamp entry was created HH:MM:SS	

Table 1

Sync Record

Key	Value	Description	Mandatory
RecordIdentity	String	Unique identifier for the job	Yes
ExecutingSite	String	GOCDB official sitename	
NJobs	int	Number of jobs	
Ndays	int	Number of days	
RecordStart	Date	Date of earliest job in month YYYY-MM-DD	
RecordEnd	Date	Date of latest job in month YYYY-MM-DD	
MeasurementDate	Date	Timestamp entry was created YYYY-MM-DD	
MeasurementTime	Time	Timestamp entry was created HH:MM:SS	

Table 2

Appendix B: Current APEL message format

Accounting Records

Each ActiveMQ message contains one or more records with the following format:

```
REPLACE INTO LcgRecords (RecordIdentity, ExecutingSite, LocalJobID, LCGJobID,
LocalUserID, LCGUserID, LCGUserVO, ElapsedTime, BaseCpuTime,
ElapsedTimeSeconds, BaseCpuTimeSeconds, StartTime, StopTime, StartTimeUTC,
```

```
StopTimeUTC, StartTimeEpoch, StopTimeEpoch, ExecutingCE, MemoryReal,
MemoryVirtual, SpecInt2000, SpecFloat2000, EventDate, EventTime,
MeasurementDate, MeasurementTime) VALUES ('recordIdentity', 'executingSite',
'localJobID', 'globalJobID', 'localUserID', 'globalUserName', 'localUserVO',
'elapsedTime', cpuTime', elapsedTimeSeconds, cpuTimeSeconds, 'startTime',
'stopTime', 'startTimeUTC', 'stopTimeUTC', startTimeEpoch, stopTimeEpoch,
'executingCE', memoryReal, memoryVirtual, specInt2000, specFloat2000,
'eventDate', 'eventTime', 'currentTimeStampDate', 'currentTimeStampTime')
```

Check queries

A check message contains a select query with the following format:

```
SELECT COUNT(*) FROM LcgRecords WHERE ((MeasurementDate = 'measurementDate'
AND MeasurementTime >= 'measurementTime') OR MeasurementDate >
'measurementDate') AND ExecutingSite = 'sitename'
```

Sync Records

A sync message contains one or more records with the following format:

```
REPLACE INTO LcgRecordsSync_v2 (RecordIdentity, ExecutingSite, Njobs, Ndays,
RecordStart, RecordEnd, MeasurementDate, MeasurementTime) VALUES ('recordID',
'executingSite', nJobs, nDays, 'recordStart', 'recordEnd',
'currentTimeStampDate', 'currentTimeStampTime')
```

Appendix C: New Data Schemas

Individual Job Accounting Record

A complete grid job accounting record describes the resources consumed by a single executing job.

Key	Value	Description	Mandatory
Site	String	GOCDB sitename	Yes
SubmitHost	String	Head node where the job was submitted	Yes
LocalJobID	String	Batch System Job ID	Yes
LocalUserID	String	Local username	
GlobalUserName	String	User's X509 DN	
VO	String	User's VO	
GridGroup	String	User’s VO Group	
Role	String	User’s VO Role	
WallDuration	int	Wallclock time for the job (seconds)	Yes
CpuDuration	int	CPU time for the job (seconds)	Yes
Processors	int	Number of processors	
NodeCount	int	Number of nodes	
StartTime	int	Start time of the job (epoch time)	Yes

EndTime	int	Stop time of the job (epoch time)	Yes
MemoryReal	int	Memory consumed by job (kbytes)	
MemoryVirtual	int	Virtual memory consumed by job (kbytes)	
ScalingFactorUnit	String	HepSpec06 | SpecInt2000 | custom	Yes
ScalingFactor	double	Value of either HepSpec, SpecInt or custom	Yes

Table 3

Summary Accounting Job Record

A summary job record describes the resources used by a collection of jobs. The collection is based on the VOMS information (VO, group and role), Site and User per month.

Key	Value	Description	Mandatory
Site	String	GOCDB sitename	Yes
Month	int	Month of summary	Yes
Year	int	Year of summary	Yes
GlobalUserName	String	User's X509 DN	
VO	String	User's VO	
Group	String	User's VOMS group	
Role	String	User's VOMS role	
EarliestEndTime	int	End time of the first job in the month (epoch time)	
LatestEndTime	int	End time of the last job in the month (epoch time)	
WallDuration	int	Sum of wall clock times for all jobs in the month	Yes
CpuDuration	int	Sum of CPU time for all jobs in the month	Yes
NormalisedWallDuration	int	Sum of normalised wall clock time for all jobs	Yes
NormalisedCpuDuration	int	Sum of normalised CPU times for all jobs	Yes
NumberOfJobs	int	Total number of jobs	Yes

Table 4

Sync Record

The Sync schema is an internal record used by APEL which defines the number of job records per month that a site contains in its local database.

The Sync record is used to compare the local site data with the data published successfully to the APEL server and publish this result as a Nagios test to the EGI Nagios infrastructure.

Key	Value	Description	Mandatory

Site	String	GOCDB sitename	Yes
NumberOfJobs	int	Total number of jobs for that month	Yes
Month	int	Month	Yes
Year	int	Year	Yes

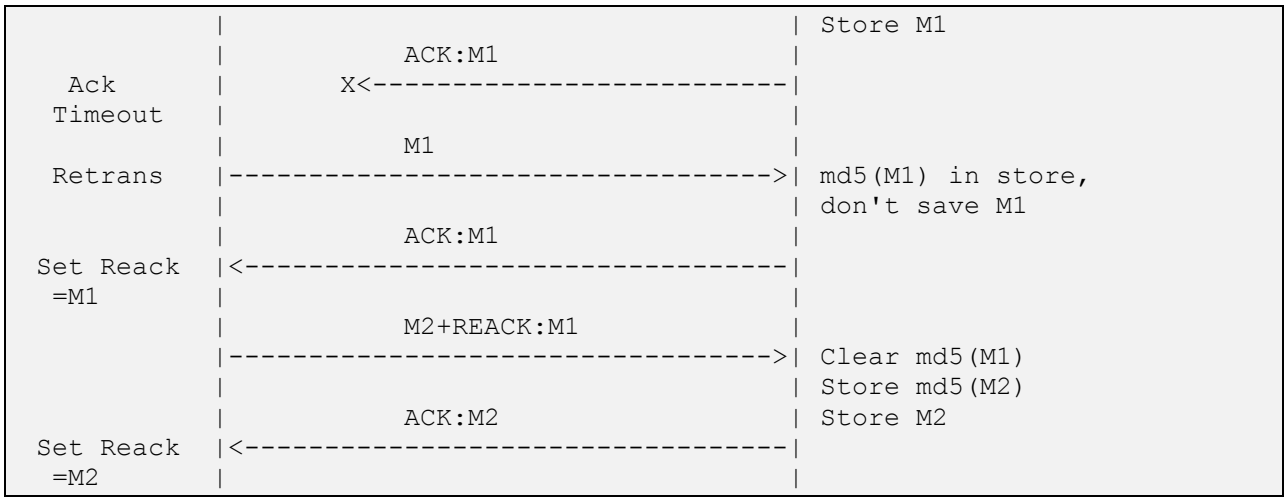
Appendix D: SSM Message Sequence Diagrams

Normal Operation

```
- NB ALL messages from a producer have a header 'reply-to' set to the
  topic/queue that the Consumer is to send an acknowledgement on.

    Producer                                        Consumer
            |           CERTREQ                    |
            |------------------------------------->|
            |                                      |
            |           CERTRESP                   |
            |<-------------------------------------|


- Certificate request has no body, just another header: 'ssm-msg-type:
certreq'
- Certificate response sent to queue/topic specified by 'reply-to' in CERTREQ
- CERTRESP body is the producer's certificate in PEM format (body is not
encrypted)

                           ...

            |           M1                         |
            |------------------------------------->| Store md5(M1)
            |                                      | Store M1
            |           ACK:md5(M1)                |
 Set Reack  |<-------------------------------------|
  =M1       |                                      |

- M1 headers: 'ssm-msg-type: msg, ssm-msg-id: <md5>' (md5 of message body)
- Ack headers: 'ssm-msg-type: ack, ssm-msg-id: <md5>' (md5 of acked message
M1)


            |           M2+REACK:M1                |
            |------------------------------------->| Clear md5(M1)
            |                                      | Store md5(M2)
            |           ACK:md5(M2)                | Store M2
 Set Reack  |<-------------------------------------|
  =M2       |                                      |

- M2 reack header: 'ssm-reack: <md5>' (md5 of M1)
```

Ack message lost

```
    Producer                                        Consumer
            |           M1                         |
            |------------------------------------->| Store md5(M1)
```


ACK:M1
Ack Timeout
M1
Retrans
md5(M1) in store, don't save M1
ACK:M1
Set Reack =M1
M2+REACK:M1
Clear md5(M1)
Store md5(M2)
ACK:M2
Store M2
Set Reack =M2