The Consultative Committee for Space Data Systems

Report Concerning Space Data System Standards

ASYNCHRONOUS MESSAGE SERVICE

INFORMATIONAL REPORT

CCSDS 735.0-G-1

GREEN BOOK

December 2012

The Consultative Committee for Space Data Systems

Report Concerning Space Data System Standards

ASYNCHRONOUS MESSAGE SERVICE

INFORMATIONAL REPORT

CCSDS 735.0-G-1

GREEN BOOK

December 2012

AUTHORITY

Issue:	Informational Report, Issue 1
Date:	December 2012
Location:	Washington, DC, USA

This document has been approved for publication by the Management Council of the Consultative Committee for Space Data Systems (CCSDS) and reflects the consensus of technical panel experts from CCSDS Member Agencies. The procedure for review and authorization of CCSDS Reports is detailed in *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-3).

This document is published and maintained by:

CCSDS Secretariat
Space Communications and Navigation Office, 7L70
Space Operations Mission Directorate
NASA Headquarters
Washington, DC 20546-0001, USA

FOREWORD

The CCSDS Asynchronous Message Service (AMS) specification is a Recommended Standard, CCSDS 735.1-B-1. This Informational Report provides additional perspectives on the motivation, design, implementation, and deployment of AMS in space flight missions.

Through the process of normal evolution, it is expected that expansion, deletion, or modification of this document may occur. This Report is therefore subject to CCSDS document management and change control procedures, which are defined in *Organization and Processes for the Consultative Committee for Space Data Systems* (CCSDS A02.1-Y-3). Current versions of CCSDS documents are maintained at the CCSDS Web site:

http://www.ccsds.org/

Questions relating to the contents or status of this document should be addressed to the CCSDS Secretariat at the address indicated on page i.

At time of publication, the active Member and Observer Agencies of the CCSDS were:

Member Agencies

- Agenzia Spaziale Italiana (ASI)/Italy.
- Canadian Space Agency (CSA)/Canada.
- Centre National d'Etudes Spatiales (CNES)/France.
- China National Space Administration (CNSA)/People's Republic of China.
- Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)/Germany.
- European Space Agency (ESA)/Europe.
- Federal Space Agency (FSA)/Russian Federation.
- Instituto Nacional de Pesquisas Espaciais (INPE)/Brazil.
- Japan Aerospace Exploration Agency (JAXA)/Japan.
- National Aeronautics and Space Administration (NASA)/USA.
- UK Space Agency/United Kingdom.

Observer Agencies

- Austrian Space Agency (ASA)/Austria.
- Belgian Federal Science Policy Office (BFSPO)/Belgium.
- Central Research Institute of Machine Building (TsNIIMash)/Russian Federation.
- China Satellite Launch and Tracking Control General, Beijing Institute of Tracking and Telecommunications Technology (CLTC/BITTT)/China.
- Chinese Academy of Sciences (CAS)/China.
- Chinese Academy of Space Technology (CAST)/China.
- Commonwealth Scientific and Industrial Research Organization (CSIRO)/Australia.
- CSIR Satellite Applications Centre (CSIR)/Republic of South Africa.
- Danish National Space Center (DNSC)/Denmark.
- Departamento de Ciência e Tecnologia Aeroespacial (DCTA)/Brazil.
- European Organization for the Exploitation of Meteorological Satellites (EUMETSAT)/Europe.
- European Telecommunications Satellite Organization (EUTELSAT)/Europe.
- Geo-Informatics and Space Technology Development Agency (GISTDA)/Thailand.
- Hellenic National Space Committee (HNSC)/Greece.
- Indian Space Research Organization (ISRO)/India.
- Institute of Space Research (IKI)/Russian Federation.
- KFKI Research Institute for Particle & Nuclear Physics (KFKI)/Hungary.
- Korea Aerospace Research Institute (KARI)/Korea.
- Ministry of Communications (MOC)/Israel.
- National Institute of Information and Communications Technology (NICT)/Japan.
- National Oceanic and Atmospheric Administration (NOAA)/USA.
- National Space Agency of the Republic of Kazakhstan (NSARK)/Kazakhstan.
- National Space Organization (NSPO)/Chinese Taipei.
- Naval Center for Space Technology (NCST)/USA.
- Scientific and Technological Research Council of Turkey (TUBITAK)/Turkey.
- Space and Upper Atmosphere Research Commission (SUPARCO)/Pakistan.
- Swedish Space Corporation (SSC)/Sweden.
- United States Geological Survey (USGS)/USA.

DOCUMENT CONTROL

Document	Title	Date	Status
CCSDS 735.0-G-1	Asynchronous Message Service, Informational Report, Issue 1	December 2012	Original issue

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

Figure Page

1 INTRODUCTION

1.1 PURPOSE AND SCOPE

This document describes the rationale, structure, and potential uses of the AMS protocols. It includes supporting materials for an AMS implementer, including implementation suggestions.

This document contains supporting information intended as a guide to developers, providing an overview of the protocols and their requirements to assist in the design and testing of an AMS implementation. In the event of any discrepancy between this guide and the AMS specification (CCSDS 735.1-B-1), the official specification always takes precedence.

1.2 DOCUMENT STRUCTURE

This Informational Report is structured as follows:

- Section 1 contains introductory information explaining the purpose, scope, document structure, and conventions and definitions, and provides a list of references.
- Section 2 contains an overview of the origins of AMS and the requirements AMS was proposed to satisfy.
- Section 3 contains an overview of the AMS system architecture.
- Section 4 contains a discussion of the AMS message exchange models.
- Section 5 contains a brief summary of the Meta-AMS (MAMS) protocol.
- Section 6 contains a brief summary of the Application AMS (AAMS) protocol.
- Section 7 contains a discussion of the Remote AMS (RAMS) protocol from a perspective that differs somewhat from that of the AMS specification.
- Section 8 contains a brief discussion of the AMS ‘transport service’ concept, with particular consideration of the use of message queues as a primary transport service.
- Section 9 contains an algorithm for estimating the worst-case elapsed time between failure of an AMS configuration server and the restoration of module registration activity.

1.3 CONVENTIONS AND DEFINITIONS

A full listing of conventions and definitions may be found in the AMS specification.

1.4 REFERENCES

The following documents are referenced in this Report. At the time of publication, the editions indicated were valid. All documents are subject to revision, and users of this Report are encouraged to investigate the possibility of applying the most recent editions of the documents indicated below. The CCSDS Secretariat maintains a register of currently valid CCSDS documents.

[1] *Asynchronous Message Service*. Recommendation for Space Data System Standards, CCSDS 735.1-B-1. Blue Book. Issue 1. Washington, D.C.: CCSDS, September 2011.

2 CONTEXT

2.1 ORIGINS

The CCSDS AMS Recommended Standard was developed in accordance with the findings of a CCSDS cross-area study that resulted in a concept paper that was delivered to CCSDS in June of 2005. The Abstract of that paper is reproduced below:

> *This concept paper proposes redirection of several current CCSDS standardization efforts on application message exchange, with the aim of eliminating duplication of effort. Specifically, it proposes:*
>
> - *That those key elements of the SOIS Message Transfer Service (MTS) service specification that are not already addressed by the SIS Asynchronous Message Service (AMS) service specification be added to the latter.*
> - *That development of MTS as a distinct product be discontinued.*
> - *That the AMS specification be further augmented, as necessary, to assure that it satisfies all requirements imposed on the MOIMS System Monitor and Control Protocol (SMCP) that pertain to general-purpose messaging.*
> - *That SMCP adopt AMS as its standard underlying end-to-end messaging service, and that the SMCP effort focus on the standardization of SMC application message syntax and semantics.*

2.2 PROPOSED REQUIREMENTS

2.2.1 GENERAL

An early step in the development of AMS was the identification of a set of proposed requirements that were uncovered during the cross-area study. These requirements are given below for informational purposes only. It should be noted that the final AMS design satisfies many, but not necessarily all of the proposed requirements, as requirements were found to be in conflict in some cases.

2.2.2 PROPOSED REQUIREMENTS FROM MOIMS

From the Mission Operations & Information Management Services (MOIMS) area, AMS shall:

a) enable a node to subscribe to a ‘scoped subset’ of all messages tagged with specified message ‘subject’ or message type (‘object ID’);

b) filter out messages that do not conform to a specified scope expression;

NOTE – This filtering can be done at the publisher if all message traffic is unicast. The more general approach, which applies to multicast message traffic as well, is to apply filtering at the subscriber upon reception.

c) enable a node to publish messages tagged with a specified subject;

d) assure that each subscriber receives every published message of the subject of the subscription, modulo any requested filtering;

e) enable a node to send a message privately to a single specified node, possibly with an implicit request for a responding private message;

f) enable a message to be staged at a rendezvous point and stored there until its intended receiver claims it;

g) enable functional partitioning of a message exchange continuum, so that:

– subject and node identifiers can be reused in multiple mission contexts without conflict;

– subject and node identifiers can be reused in multiple mission operational phases (test; simulation; Assembly, Test, and Launch Operations, etc.) without conflict;

h) limit service access (message transmission, message reception, and message content disclosure) to authorized users, and authenticate users before granting the access to which they have authorized access.

2.2.3 PROPOSED REQUIREMENTS FROM MTS OVERVIEW

From the MTS overview, AMS shall:

a) transfer information of arbitrary structure from one node to another node or to multiple nodes;

b) asynchronously notify application nodes of communication events, at the option of the application;

c) enable application to specify Quality of Service (QoS) within the range that is possible;

d) enable application to specify service delivery character (e.g., reliability parameters) within the range that is possible;

e) enable QoS and service delivery character specification at subscription granularity, where a subscription is in effect a 2-tuple (receiving node ID, subject ID);

f) provide applications with mapping between textual and numeric representations of node and subject IDs;

g) enable communication in a consistent manner via Local Area Network (LAN), Wide Area Network (WAN) (wire or R/F), or Internet;

h) enable communication in a consistent manner among nodes running on a single computer or multiple computers, at a single physical site (space vehicle or planetary surface location) or multiple sites;

i) utilize TCP/IP, SCPS-TP, data link layer protocols, pipes, shared memory, message queues as available;

j) utilize IP multicast as available;

k) operate in a consistent manner over a wide variety of computing platforms (operating systems and underlying hardware);

l) provide facilities for monitoring and controlling the message service;

m) minimize transmission and processing overhead;

n) minimize latency in message conveyance;

o) occupy as little memory as possible;

p) be thread-safe, i.e., usable by multiple threads of the same process concurrently.

2.2.4 PROPOSED REQUIREMENTS FOR SPACE INTERNETWORKING

To facilitate space internetworking, AMS shall::

a) provide location transparency: applications shall not need to know one another's network locations in order to exchange data;

b) enable a single message exchange continuum to be automatically and dynamically split into two continua, and enable two message exchange continua to be dynamically and automatically merged into a single continuum;

c) be robust against software and network faults;

d) be tolerant of network disruption and delay;

e) minimize reliance on high data rates;

f) minimize reliance on connection-oriented protocols;

g) be non-proprietary;

h) integrate existing protocols and APIs where appropriate, in preference to new development;

i) scale to large numbers of communicating nodes;

j) provide simple API and management interfaces.

3 SYSTEM ARCHITECTURE

AMS provides for communication among user application *modules*. These application modules are simply distinct sequential flows of application control logic, whether called processes, tasks, or threads, that may all reside in a single system or may interoperate in a more complex system, e.g., among individual hardware components. Figure 3-1 shows the architectural elements of AMS on a single continuum.

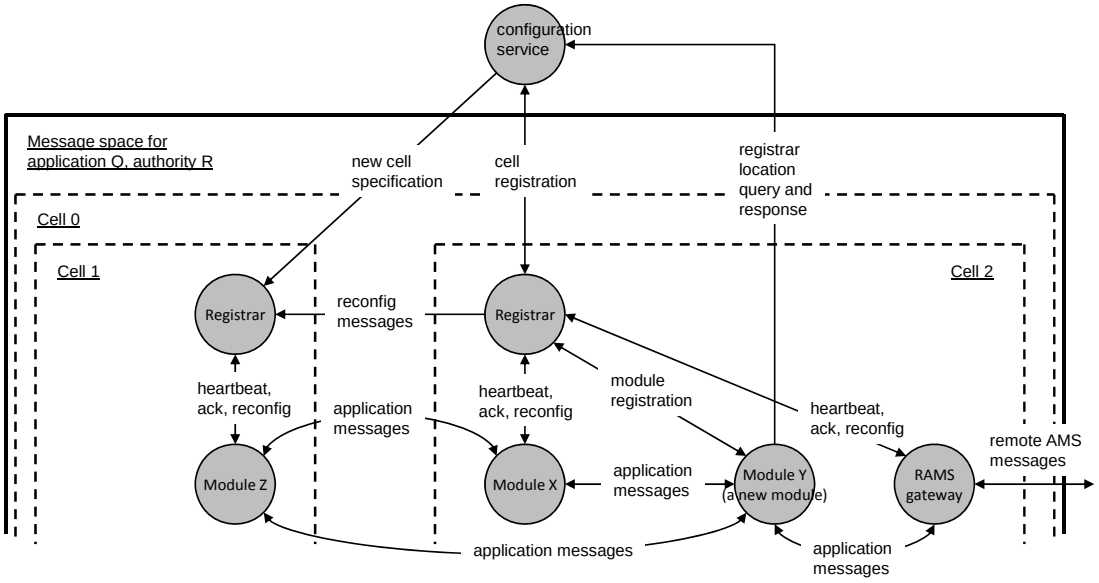


Figure 3-1: Architectural Elements of AMS (a Single Continuum)

Modules are managed by a registration server, or *registrar*, that monitors status and propagates configuration information. A module can only begin operation by announcing itself to a registrar. It learns the location of its registrar by querying a *configuration server*, which is responsible for monitoring the health of all registrars in some single *continuum*.

A continuum is a closed set of entities—modules, registrars, and a configuration server—that utilize AMS for purposes of communication among themselves. Although a continuum operates independently, it may be capable of interacting with a foreign continuum through the usage of the *Remote AMS (RAMS)* protocol. (For further discussion on RAMS, see section 7.) Multiple configuration servers may be defined for a single continuum for redundancy purposes, but only a single, highest-ranking configuration server instance will operate at any given time.

Every module is a participant in some instance of some AMS *application*, a data system implementation that utilizes the AMS protocol in its operation and is identified by some application name. Multiple instances of the same application can operate concurrently in the same AMS environment; they are distinguished by the names of the ‘authorities’ that are responsible for configuring and operating them.

An application instance is also termed a *venture*. A given venture may be entirely contained within a single continuum or may span multiple continua. The subset of a given venture that is contained within a given continuum is termed the *message space* for that venture in that continuum.

In addition to being topologically subdivided into message spaces, a venture may be organizationally subdivided into *units*. For example, a rover mission operations venture might comprise two messages spaces—one residing on the rover itself, the other in a ground operations center—and it might also comprise three units: a mobility system, a camera system, and a navigation system. Since the rover's mobility system encompasses both sensor and actuator nodes on the rover and a mission odometer in the ground operations center, the mobility system unit, like the venture itself, spans multiple continua.

The subset of a unit that is contained within a given continuum is termed the *cell* of that unit that is in that continuum. A cell comprises a registrar and its associated modules.

A module is categorized by the *role* it is designated to perform in the overall system. Message exchange among modules may be filtered based on any combination of unit, role, continuum, and the associated subject of the specific message.

Messages are transmitted between entities in AMS over a transport service. The AMS protocols may operate on virtually any available communications method. Multiple transport protocols may be used to facilitate messaging between diverse sets of modules, or to optimize delivery of specific message types. Transport services are described in more detail in section 8, Transport Services.

4 MESSAGE EXCHANGE MODELS

4.1 GENERAL

AMS was primarily designed for communications between and within spacecraft systems; however, it may also be applicable to a variety of other systems requiring communications between discrete software applications or hardware units.

AMS supports a variety of message exchange models, including send/receive, publish/subscribe, synchronous query, and announcements. A complete AMS implementation is capable of communication using any combination(s) of these models. Implementers alternatively may implement compliant systems focusing on a subset of these available capabilities to meet mission-specific goals.

To complement the definitions available in the specification, the following sections provide an overview of the message exchange models available in the AMS protocol.

4.2 DIRECT EXCHANGE (SEND/RECEIVE)

In this model, messages are sent to an explicitly designated recipient, addressed by module ID and domain. A message may only be sent or received if corresponding invitations are in place, as established through an *invitation* primitive. An *invitation* is sent 'to notify all modules in the message space of the module's willingness to accept private messages on the indicated subject in the domain of the service request'. This means to send a message, the recipient must *invite* messages on the indicated subject, from a *continuum*, *unit*¸ and *role* that includes the sender. (The domain of the invitation may be 'all continua' and/or 'all units' and/or 'all roles'.)

4.3 SYNCHRONOUS QUERY

Messages may be privately transmitted to a specified module with the sender's activities optionally suspended until a reply is received. This model uses the *query* and *reply* primitives and, as in the previous section, requires *invitations* to be in place for the specified subject and domain for both query and reply.

4.4 PUBLISH AND SUBSCRIBE MODEL

In this scenario, 'messages may be published anonymously to a time-varying community of self-selected subscribers.' A message is published on a specified subject to all modules that have a current subscription on the indicated subject for a domain that includes the sender.

Subscriptions may be asserted or cancelled at any time. Subscriptions are issued using the *subscribe* primitive to a specified subject and domain in the same manner as an invitation. Subscriptions and invitations are, however, distinct, and neither may be used as criteria for accepting messages outside their respective exchange models.

4.5 ANNOUNCEMENTS

Announcements complement the direct exchange model by allowing a single message to be transmitted privately to all valid recipients. An *announce* request is directed towards a specific *subject* and *domain*. The message is copied to all modules in that domain that currently invite messages on that subject from a domain that includes the sender.

In contrast, the *publish* request transmits messages to all modules with a valid subscription on a domain matching the sender. For *announce*, the message is privately transmitted to all modules in the addressed *domain* that have an active *invitation* on this subject from a *domain* that includes the sender: the announcement domain declared by the sender constrains transmission of the message to a set of potential receivers, and the invitation domain declared by each potential receiver constrains acceptance and reception of the message at that receiver.

5 META-AMS (CONFIGURATION) MESSAGE FLOW

5.1 GENERAL

This section provides an overview of the general flow of selected configuration messages (Meta-AMS Protocol Data Units [MPDUs]) and related methods. Horizontal dashed lines in the accompanying figures represent independent, potentially concurrent message flows in diagrams illustrating several related examples.

5.2 REGISTRAR INITIALIZATION

Upon commencing operations, the registrar shall locate the configuration server and send an *announce_registrar* MPDU to the configuration server. The configuration server (referenced in figures as 'CS') will verify that this registrar belongs to a known cell in which no registrar is currently active and will respond accordingly.

A *rejection* will be transmitted with an appropriate explanation if verification fails; otherwise a *registrar_noted* will be returned. A *cell_spec* indicating the updated status of the registrar will be sent by the configuration server to all other registrars in the message space. If no other cells are active, a single *cell_spec* will be returned to the initializing registrar with a unit number set to that registrar's own unit number to indicate this condition. Otherwise, one *cell_spec* is returned for every other cell in the message space. This exchange is illustrated in figure 5-1.

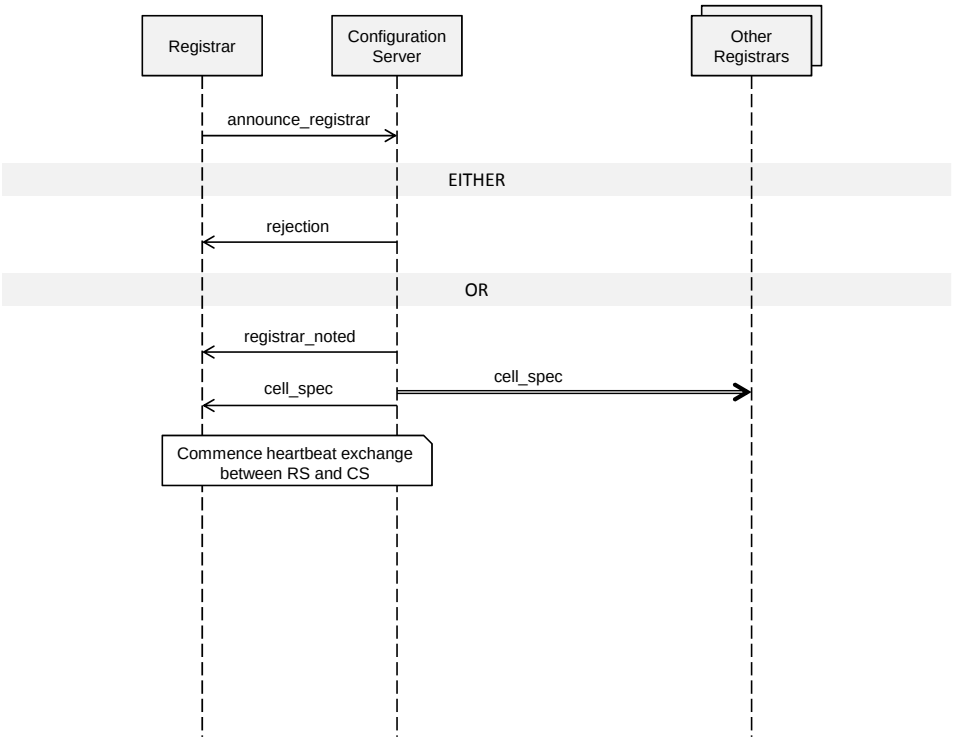


Figure 5-1: Registrar Initialization

5.3 MODULE REGISTRATION

Module startup and registration can be viewed as a three-part process, which is depicted in figure 5-2. First, the configuration server is located using the 'configuration server interrogation' process (not illustrated) as described in the AMS specification. If the configuration server is not available, registration cannot proceed.

Next, the module sends a *registrar_query* to the configuration server to determine the location of the module's registrar. If a registrar is not available for the module's indicated cell, a *registrar_unknown* message is returned (not illustrated), resulting in the generation of a ***Fault.indication*** in the module. Otherwise a *cell_spec* MPDU is returned, noting the location of the registrar.

The module may now attempt to register with its Registration Server (RS). A *module_registration* MPDU is transmitted and the registrar will reply with either a *you_are_in* MPDU if successful, or a *rejection* message (not depicted) indicating the reason for refusing registration.

If the registration is accepted, the implementer may choose between two valid methods for the subsequent registration configuration:

– The primary method is for the RS to transmit an *I_am_starting* message, containing the new module's configuration state, to all other modules in the cell and to all other registrars in the message space, to be forwarded to their respective modules. Each module will respond with an *I_am_here* message, containing its identifying information and a listing of all active subscriptions and invitations.

– Alternatively, the registrar may be tasked to monitor and record the status of all subscription, invitation, and related information for modules in its message space. The RS may choose in this case to construct, from its own information, one *I_am_here* message for each registered module and transmit all of those messages to the new module. The registrar will then transmit a *module_has_started* message to all other modules in place of the procedures previously discussed. In either case, the behavior of the new module remains the same.

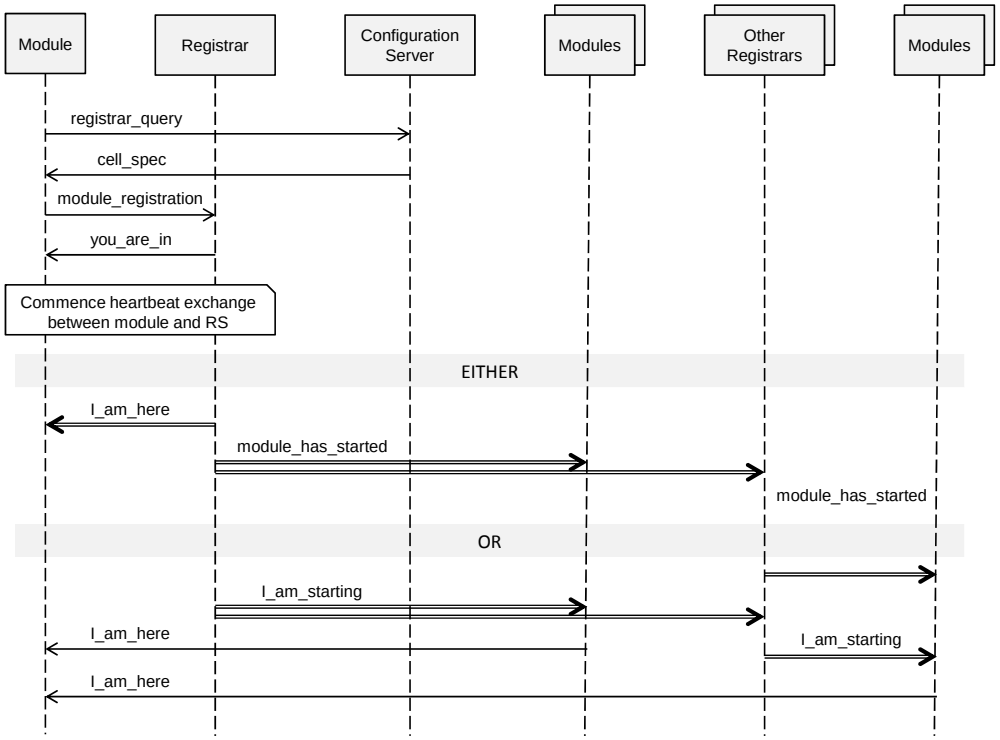


Figure 5-2: Module Registration

5.4 HEARTBEATS

Heartbeats are exchanged periodically between the configuration server and each registrar, and between the registrar and each associated module. This process is described in detail in the AMS specification. The following figures provide sample illustrations of heartbeat exchanges between a module, its registrar, and its configuration server.

Figure 5-3 illustrates nominal heartbeat exchange among all entity types. The horizontal dashed line indicates that each heartbeat transmission is an independent event, based on internal conditions or timers. An implementation may choose to transmit heartbeats either in response to a required heartbeat receipt, or directly based on an internal timer.

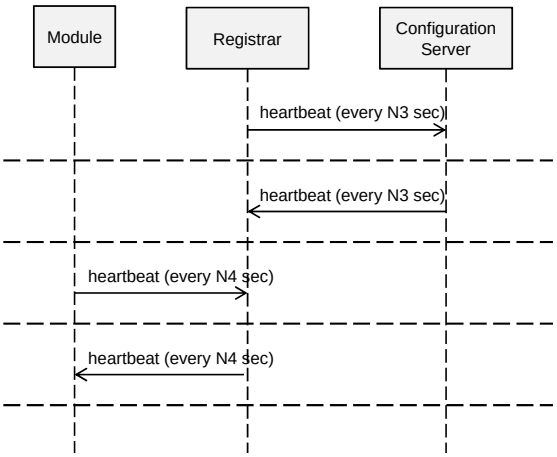


Figure 5-3: Heartbeat Exchanges

Figure 5-4 illustrates actions taken by the registrar in the event of an imputed module failure.

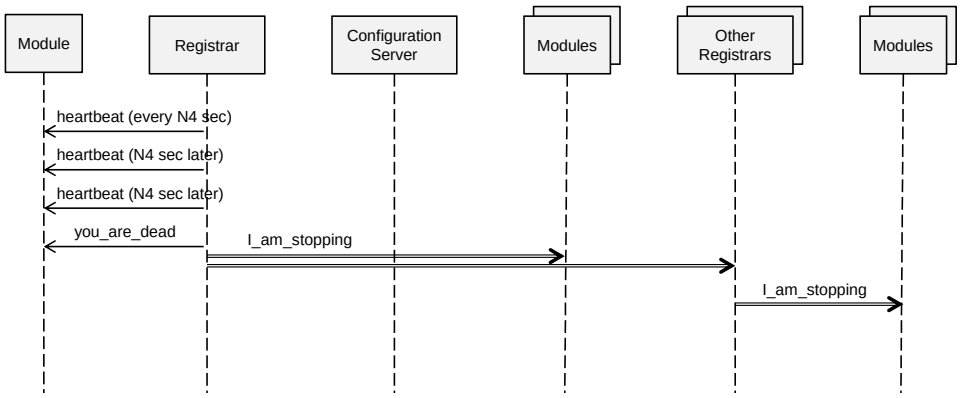


Figure 5-4: Imputed Module Failure

A module may also impute the failure of its registrar in a reciprocal manner. In this event, the module will query the configuration server for the location of the restarted registrar and reconnect as depicted below in figure 5-5.

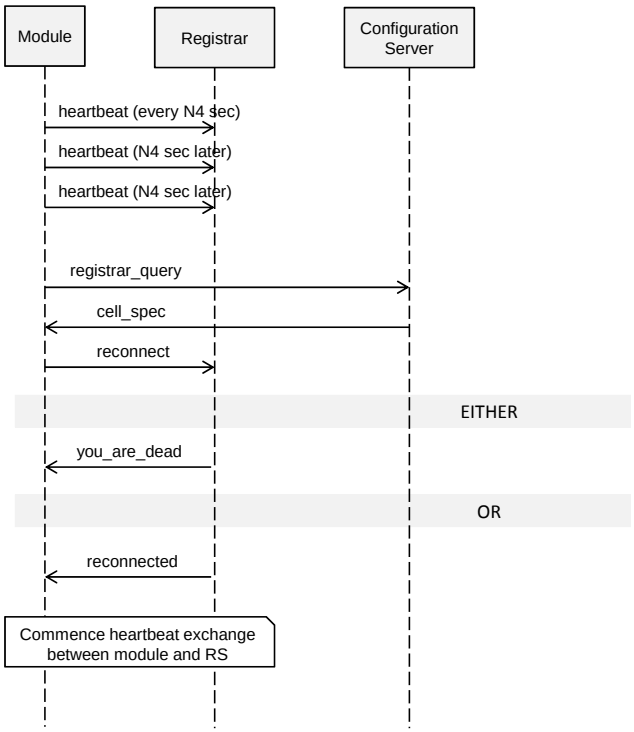


Figure 5-5: Imputed Registrar Failure

5.5 ASSERTION OF SUBSCRIPTIONS AND INVITATIONS

Modules may assert invitations to privately transmitted messages, or subscriptions to publically broadcast messages. The mechanisms involved in subscriptions and invitations mirror each other and provide the same capabilities for different message exchange models.

Figure 5-6 illustrates the subscription assertion and cancellation message flows. The procedure for invitations is virtually identical, utilizing instead the analogous *invite* and *disinvite* message types.

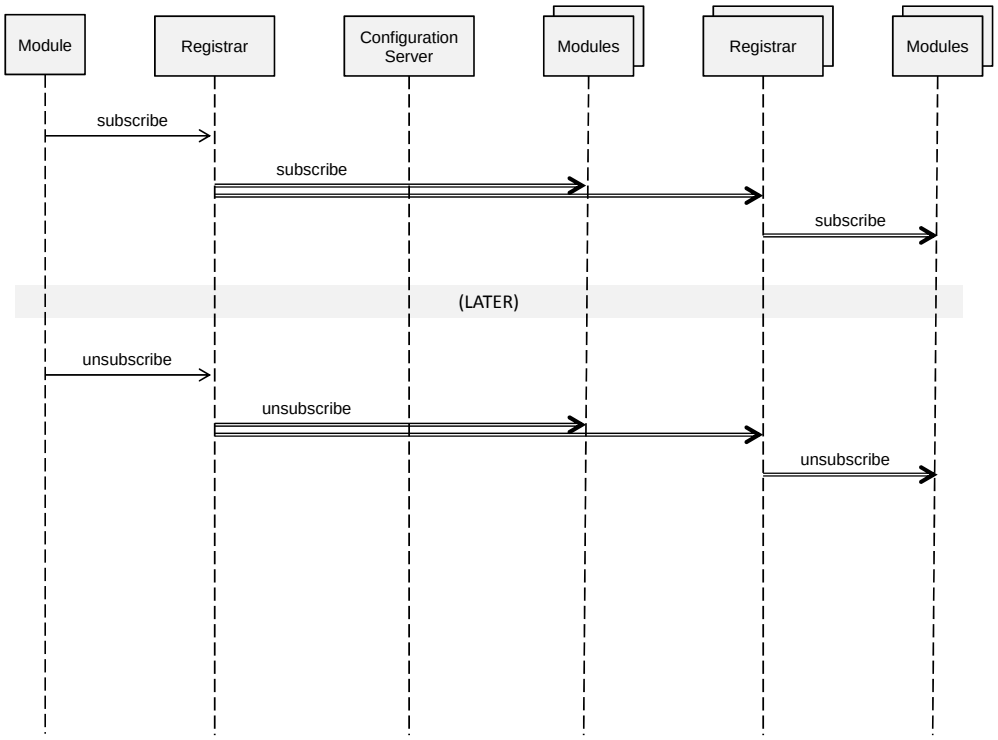


Figure 5-6: Subscription Assertion and Cancellation

6 APPLICATION AMS MESSAGE FLOW

6.1 OVERVIEW

This section illustrates the process for sending user AAMS messages between local modules (within a single continuum).

6.2 SEND PRIVATE MESSAGE

The purpose of the Send Private Message use case is to show how two AMS modules interact for the purpose of unidirectional point-to-point messaging between two onboard applications. This is an example of how a mission timeline application may send a telecommand to a payload controller. Figure 6-1 shows the scenario for unidirectional point-to-point asynchronous messaging.

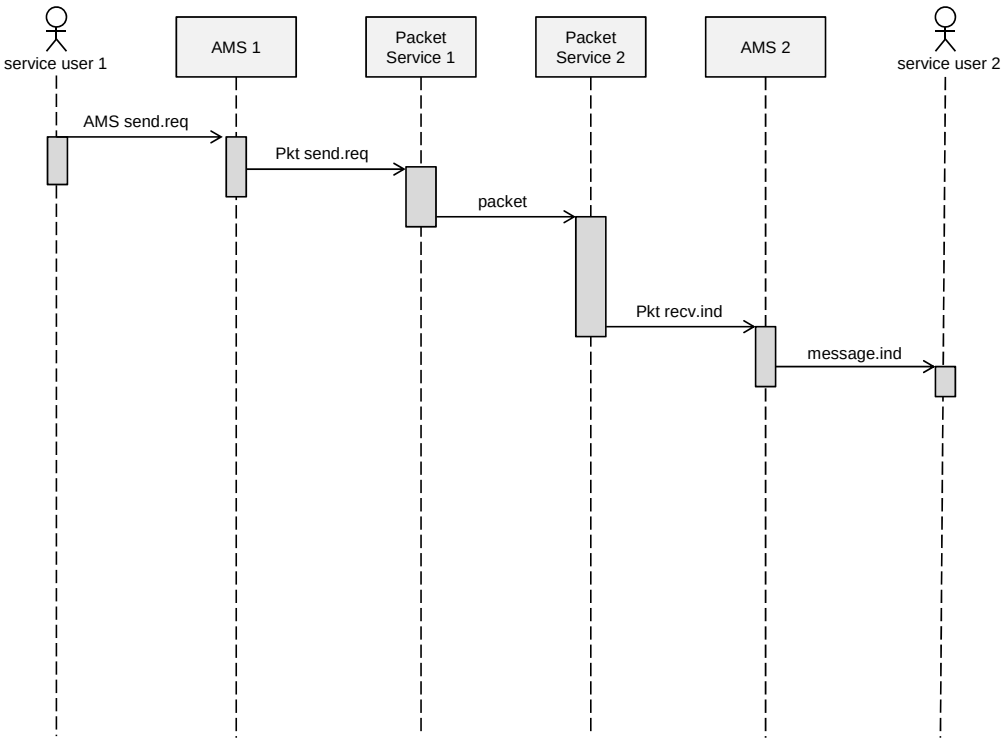


Figure 6-1: Send Message from Onboard Application to Onboard Application

In this figure, service user 1 issues a **Send.request** to send a message to a single specified destination, service user 2. A Protocol Data Unit (PDU) is built to contain the new message. The PDU is then sent directly to the destination via a **Packet_Send.request** of the Packet Service. Control is returned to service user 1 as soon as the message has been sent.

Upon reception of the PDU, the Packet Service will, with a P**acket_Receive.indication**, notify AMS, which in turn will notify service user 2 with a **Message.indication**.

6.3 QUERY MODULE WITH PRIVATE MESSAGE

The purpose of the Query Module with Private Message use case is to show how two AMS modules interact for the purpose of bidirectional point-to-point query/reply messaging. This illustrates how, for example, a Fault Detection Isolation and Recovery (FDIR) application may query the status of a Payload Controller application. Figure 6-2 shows the scenario for bidirectional point-to-point synchronous query/reply messaging.

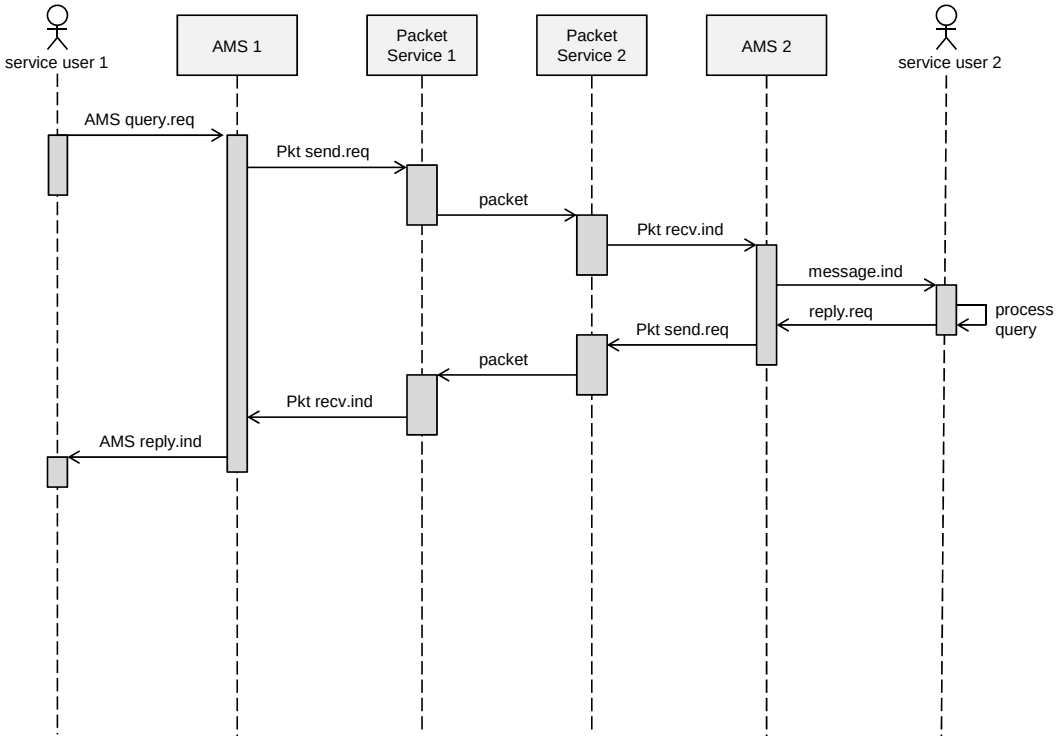


Figure 6-2: Query from Onboard Application to Onboard Application

Service user 1 issues an AMS **Query.request** to send a message to a single specified destination AMS, service user 2. The query will be identified by a unique context identifier that service user 1 provides. A PDU is built to contain the new message. The PDU is then sent directly to the destination via a **Packet_Send.request** of the Packet Service. Service user 1's AMS module will be blocked awaiting a reply from service user 2.

Upon reception of the PDU, the Packet Service will notify AMS with a **Packet_Receive.indication**, which in turn will notify service user 2 with a **Message.indication**.

At this point service user 2 will process the query. Once it is processed, service user 2 will issue a **Reply.request** to send a message to the sender of the query message, service user 1. The reply will be identified by the same unique context identifier that was provided in the

Message.indication. A PDU is built to contain the new message. The PDU is then sent directly to the destination via a **Packet_Send.request** of the Packet Service.

Upon reception of the PDU, the Packet Service will notify AMS with a **Packet_Receive.indication**, which in turn will notify service user 1 with a **Reply.indication**. Control will now be returned to service user 1, who will then be responsible for applying the reply to the original request.

6.4 TRANSMIT ANNOUNCEMENT

The purpose of the Transmit Announcement use case is to show how many AMS modules interact for the purpose of selective multicast messaging. This is an example of how an FDIR application may notify all Payload Controller applications to shut down. Figure 6-3 shows the scenario for selective multicast messaging.

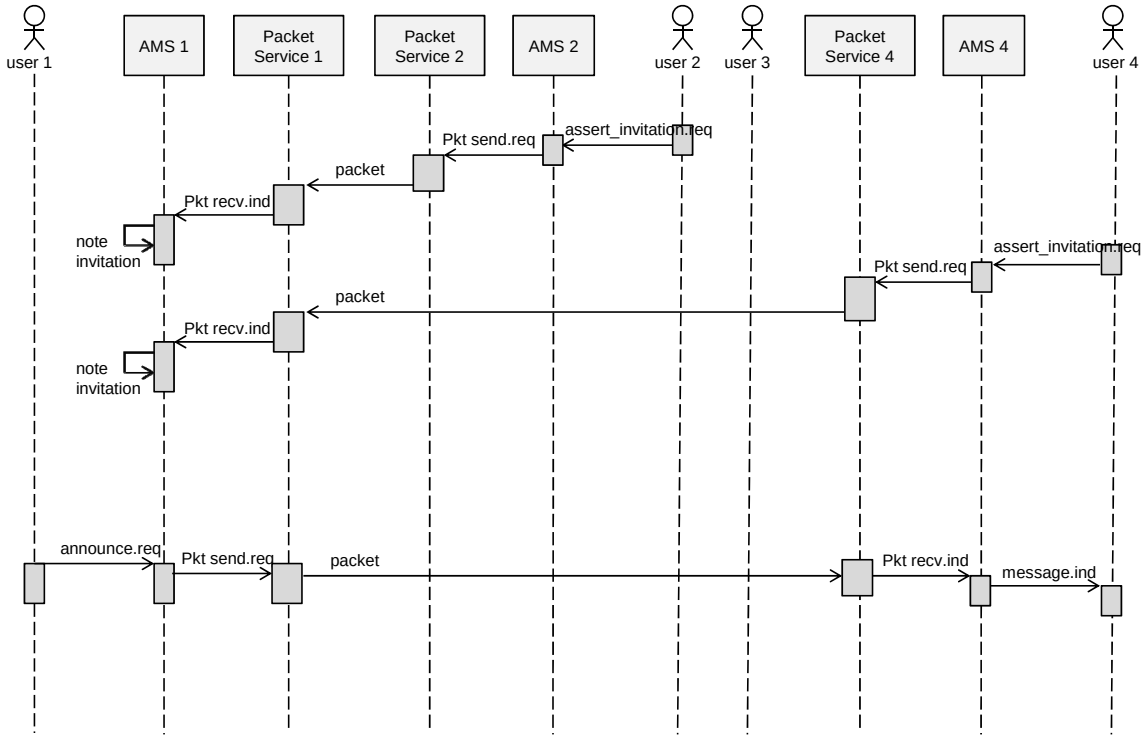


Figure 6-3: Announcement from Onboard Application to other Onboard Application

Service users 2 and 4 have invited ‘shut down’ messages, but only service user 4 is registered as a Payload Controller. Service user 1 issues an **Announce.request** to send a ‘shut down’ message to all Payload Controllers. A PDU is built to contain the new message. The PDU is then sent directly to each Payload Controller module that has invited these messages, via a **Packet_Send.request** of the Packet Service. The message is transmitted only to service user

4's module: even though service user 2 has invited messages on this subject, it does not receive the announced message because it is not a Payload Controller; service user 3 might be a Payload Controller, but because it has not invited 'shut down' messages, its AMS module does not receive the announced message. Control is returned to service user 1 as soon as all message packet transmission has been completed.

Upon reception of the message, service user 4's Packet Service will notify AMS with a **Packet_Receive.indication**, which in turn will notify service user 4 with a **Message.indication**.

Throughout the whole interaction, service user 3 will not receive any messages.

6.5 PUBLISH MESSAGE

The purpose of the Publish Message use case is to show how many AMS modules interact for the purpose of publish/subscribe multicast messaging. This is an example of how a Payload Controller application may publish the latest data to interested applications, e.g., a Mission Autonomy Controller application. Figure 6-4 shows the scenario for publish/subscribe multicast messaging.

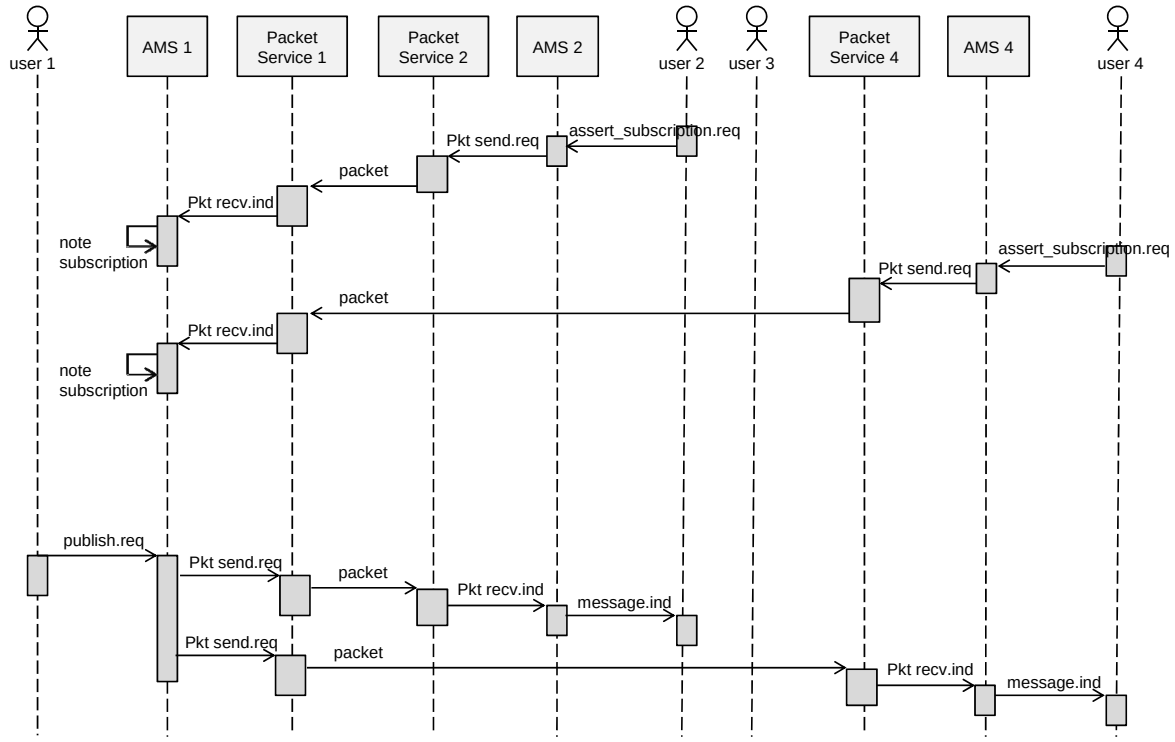


Figure 6-4: Publish Message from Onboard Application to Subscribed Onboard Applications

Service user 2 issues an **Assert_subscription.request** to subscribe to some subject in a message space which includes service user 1. A PDU is built to contain the subscription message. The PDU is then sent directly to the potential publisher via a **Packet_Send.request** on the Packet Service. Control is returned to service user 2 as soon as the message has been sent.

Upon reception of the message, the Packet Service for service user 1 will notify AMS with an **Assert_subscription.indication**. The subscription for service user 2 will be added to the list of known subscribers; service user 1 itself will not be notified.

Service user 4 issues an **Assert_subscription.request** to subscribe to the same subject in a the same message space. This proceeds as before.

Service user 1 issues a **Publish.request** to publish a new message to all subscribed modules, in this case service user 2 and service user 4.

For each subscribed module, a PDU is built to contain the new message. The PDU is then sent directly to each subscriber via a **Packet_Send.request** on the Packet Service. Control is returned to service user 1 as soon as all the message transmission is complete.

Upon reception of the message, each Packet Service will, with a **Packet_Receive.indication**, notify AMS, which in turn will notify service user 2 and service user 4 with a **Message.indication**.

Throughout the whole interaction service user 3 will not receive any messages.

7 REMOTE AMS

7.1 OVERVIEW

RAMS provides the following capabilities:

a) Inter-continuum ‘Send’—a message from a sending module in one continuum can be relayed to a specific module in another continuum.

b) Inter-continuum ‘Announce’—a message from an announcing module in one continuum can be relayed to the inviting modules in another continuum or all other continua.

c) Inter-continuum ‘Publish’—a message from a publisher in one continuum can be relayed to all subscribing modules in other continua. This requires that the subscribing module(s) place a subscription with their local gateway beforehand (see the next capability).

d) Inter-continuum ‘Subscribe’—a module in one continuum can subscribe to messages published in another continuum or all continua.

All of the above capabilities require an underlying infrastructure that is able to deliver messages between continua. This infrastructure is referred to as a *RAMS network*.

Each of these RAMS capabilities, from a module’s point of view, utilizes the same transmission mechanism as the local continua equivalent. The RAMS gateway is responsible for all inter-continua routing based on the domain(s) specified by the module.

7.2 RAMS NETWORK

A RAMS network is a collection of gateways (one for each continuum within the network) that are connected together such that each gateway has a path to all other gateways.

A RAMS network may be configured as either a *mesh* or a *tree*. Figure 7-1 shows an example of a mesh network, while figure 7-2 provides an example of a tree network.

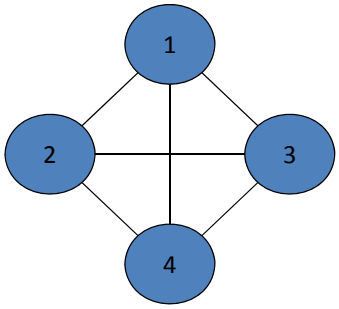


Figure 7-1: RAMS Mesh Network (Example)

In a mesh, each gateway has a direct connection to each of the other gateways. When a gateway needs to send a message to a remote continuum, it sends it directly to the gateway for that continuum. Each gateway keeps track of the gateway assigned to each continuum within the network and the connection settings for each of those gateways.

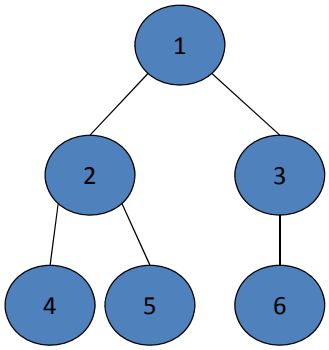


Figure 7-2: RAMS Tree Network (Example)

In a tree, some pairs of gateways may not be connected directly, but there is always exactly one route between any pair of gateways.

When a gateway needs to send a message to a remote continuum, it forwards the message to whichever gateway is the first hop (‘conduit’) to the destination continua. If the remote continuum is a direct neighbor, the conduit will be the gateway assigned to the destination continuum, and no further hops are necessary. If not, the conduit will forward the message to its own conduit to the destination continuum. By proceeding one hop at a time, the message will eventually reach the destination continuum. For example, in figure 7-2, if gateway #4 needs to send a message to #3, it forwards the message to #2, #2 forwards it to #1, and #1 forwards it to #3.

If a gateway needs to propagate a message to ‘all continua’ (i.e., every gateway in the network), it forwards the message to its neighbors, who in turn forward the message to their other neighbors, who in turn forward the message to their other neighbors, etc. For example, in figure 7-2, if gateway #1 needs to propagate a message to ‘all continua’, it forwards the message to its neighbors (#2 and #3), #2 forwards the message to its other neighbors (#4 and #5), and #3 forwards the message to its other neighbors (#6). Each gateway keeps track of the conduit used to reach each of the other continua in the network, who its neighbors are, and the contact info for each neighbor.

7.3 INTER-CONTINUUM ‘SEND’

For an inter-continuum ‘send’, the RAMS network forwards a message from a module in the source continuum to a specified module in a destination continuum (if the destination module has asserted an invitation on the message’s subject from a domain that includes the source module).

First, the message travels from the sending module (in the source continuum) to its local gateway. Second, the message is forwarded through the RAMS network to the remote gateway that represents the destination continuum. Third, the remote gateway forwards the message to the destination module if the message satisfies an invitation that has been asserted by that module.

7.4 INTER-CONTINUUM 'ANNOUNCE'

For an inter-continuum 'announce', the RAMS network forwards a message from a module in the source continuum to all modules within a specified destination domain that have asserted an invitation on the message's subject from a domain that includes the source module. The destination domain is specified in terms of role, unit, and continuum (either a specific continuum or all continua).

First, the message travels from the sending module (in the source continuum) to its local gateway. Second, the message is forwarded through the RAMS network to the gateways that represent the destination domain (i.e., either a single gateway or all gateways within the network). Third, the destination gateway(s) forward the message to those local modules that are within the destination domain and have asserted an invitation for the given subject from a domain that includes the source module.

7.5 INTER-CONTINUUM 'PUBLISH'

For an inter-continuum 'publish', the RAMS network forwards a message from a module in the source continuum to every module (in all other continua) that has asserted a subscription that is satisfied by the message.

First, the message travels from the publishing module to its local gateway. Second, the message is forwarded through the RAMS network to every gateway that represents a continuum that contains a subscriber (the 'subscribing gateways'). Third, each subscribing gateway forwards the message to each of the local modules that is a subscriber.

The information needed to route published messages to all subscribers is established during the inter-continuum 'subscribe' process. It is a sort of 'trail system' that leads from all publishers to all subscribers, accomplishing the following:

- Each publishing module has a subscription from its local gateway.
- Each gateway that receives the message knows where to forward the message (i.e., to those local modules that are subscribers and/or those neighbor gateways that lead to a subscriber).

For example (see figure 7-3 and follow the arrows), if there is a subscriber in continuum #2, and there are publishers in continuum #3, the trail system goes from continuum #3's local modules, to continuum #3's gateway, to continuum #1's gateway, to continuum #2's gateway, to continuum #2's local subscriber.

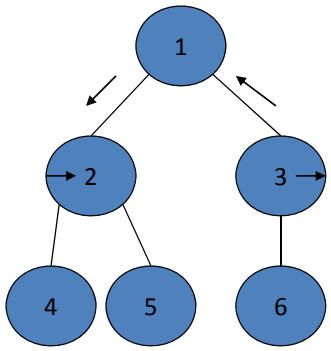


Figure 7-3: Trail System (example)

NOTES

1 Any particular message may match multiple subscriptions. For example, a message on subject 23 from continuum #3 would match one subscription that specified subject 23 from continuum #3, and another subscription that specified subject 23 from 'all continua'.

2 There is a separate trail system for each subscription.

3 In visual terms, the complete trail system used for a particular published message may be a union of multiple trail systems (one for each applicable subscription). To be precise, each gateway that receives the message forwards the message to every neighbor gateway and local module that is a destination for a subscription that is satisfied by the message.

7.6 INTER-CONTINUUM 'SUBSCRIBE'

7.6.1 GENERAL

For an inter-continuum 'subscribe', the RAMS network forwards the subscription as necessary to establish/maintain a trail system (within the RAMS network) that goes from all publishers to all subscribers. These trail systems are used for inter-continuum 'publish'.

Background concepts:

– Each subscription is for a particular subject from a particular publishing domain; the domain is specified in terms of role, unit, and continuum (either a particular continuum or 'all continua'). For many RAMS decisions, the only relevant portion of the subscription's publishing domain is the continuum.

– For a particular subscription, there can be multiple subscribers. For example, if a module in continuum #1 subscribes to subject 23 from continuum #2, and a module in continuum #3 also subscribes to subject 23 from continuum #2, there are two

subscribers. However, it should be noted that both subscribers are associated with the same subscription (subject 23 from continuum #2).

– In this section, the term ‘published message’ refers to a message that satisfies a particular subscription.

7.6.2 FORWARDING AN INTER-CONTINUUM SUBSCRIPTION TO PUBLISHERS

First, a module asserts a subscription via the MAMS protocol. The MAMS protocol propagates each subscription to every module in the local continuum, including the module that functions as the local gateway. Second, if the subscription’s domain extends outside the local continuum, the local gateway forwards the subscription through the RAMS network to every gateway that represents a continuum that is within the subscription’s domain (the ‘publishing gateways’). Third, each publishing gateway forwards the subscription to its local modules by asserting the subscription via the MAMS protocol.

In a mesh network (see figure 7-1), if a local module in continuum #3 subscribes to messages from continuum #4, the subscription travels from the local module to gateway #3, to gateway #4, to the local modules in continuum #4.

In a tree network (see figure 7-2), if a local module in continuum #3 subscribes to messages from continuum #4, the subscription travels from the local module to gateway #3, to gateway #1, to gateway #2, to gateway #4, to the local modules in continuum #4.

7.6.3 ESTABLISHING A TRAIL SYSTEM FOR A SUBSCRIPTION—CONCEPTUAL VIEW

The trails in the trail system connect all publishers to all subscribers. All trails go in a direction that heads toward subscribers.

In response to the first subscriber, a trail system is established that connects the subscriber to all publishers. The effect is different depending on whether the RAMS network is a mesh network (figure 7-4) or a tree network (figure 7-5). In these two examples a local module in continuum #2 subscribes to subject 23 from continuum #3. (It is assumed this is the first subscriber for this subject from this domain.)

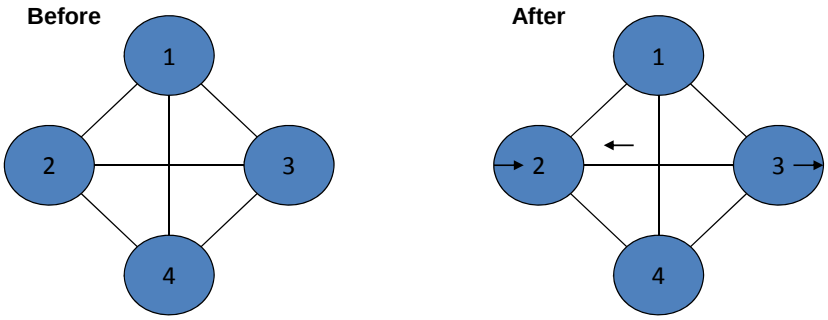


Figure 7-4: First Subscription—Mesh network

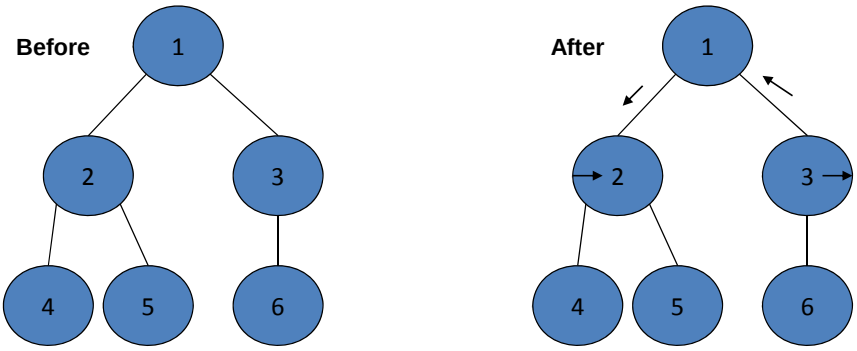


Figure 7-5: First Subscription—Tree Network

For each additional subscriber, trails are added as needed to connect new subscribing gateways into the existing trail system. In this mesh network example (figure 7-6), a local module in continuum 1 subscribes to subject 23 from continuum 3. There is already a subscriber in continuum 2.

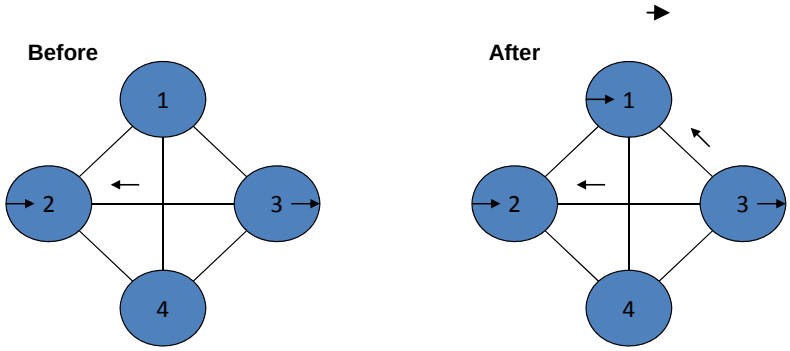


Figure 7-6: Additional Subscriber—Mesh Network

In this tree network example (figure 7-7), a local module in continuum 4 subscribes to subject 23 from continuum 3. There is already a subscriber in continuum 2.

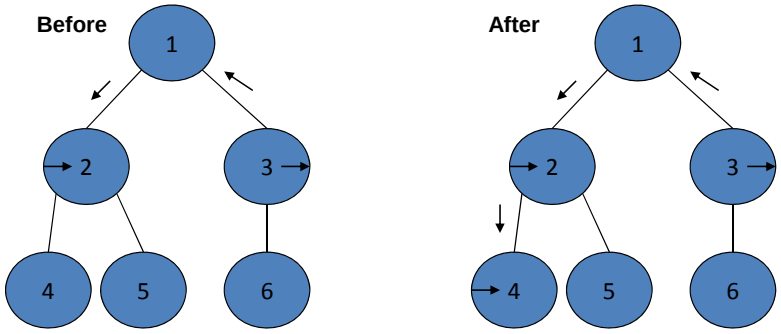


Figure 7-7: Additional Subscriber—Tree Network

Similarly, each time a subscriber cancels, any trails that exist solely to reach that subscriber are removed. (If the last remaining subscriber cancels, the entire trail system is removed.) In this mesh network example (figure 7-8), a local module in continuum 2 cancels its subscription to subject 23 from continuum 3. There is still a subscriber in continuum 1.

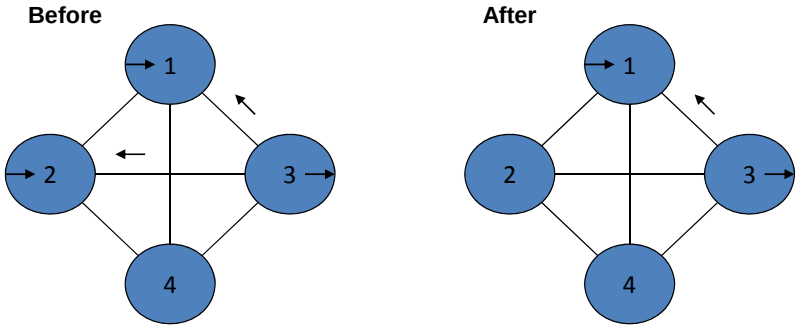


Figure 7-8: Subscriber Cancellation—Mesh Network

In this tree network example (figure 7-9), a local module in continuum 2 cancels its subscription to subject 23 from continuum 3. There is still a subscriber in continuum 4.

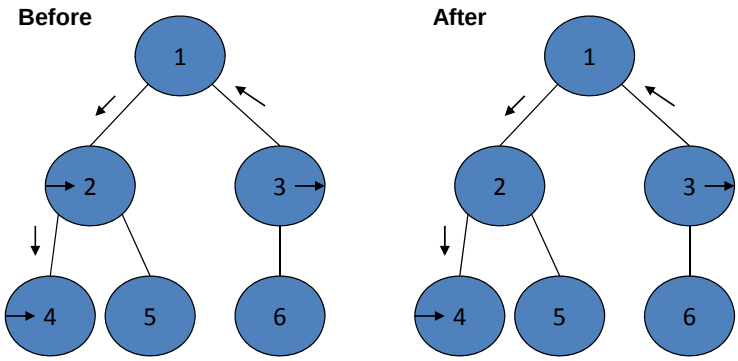


Figure 7-9: Subscriber Cancellation—Tree Network

When an entire new continuum joins the RAMS network, trails are added for each subscription whose domain includes the new continuum (because there are new publishers to connect to) (see figure 7-10). In these examples, the module in continuum 1 has subscribed to subject 23 from all continua and a new continuum 4 is added.

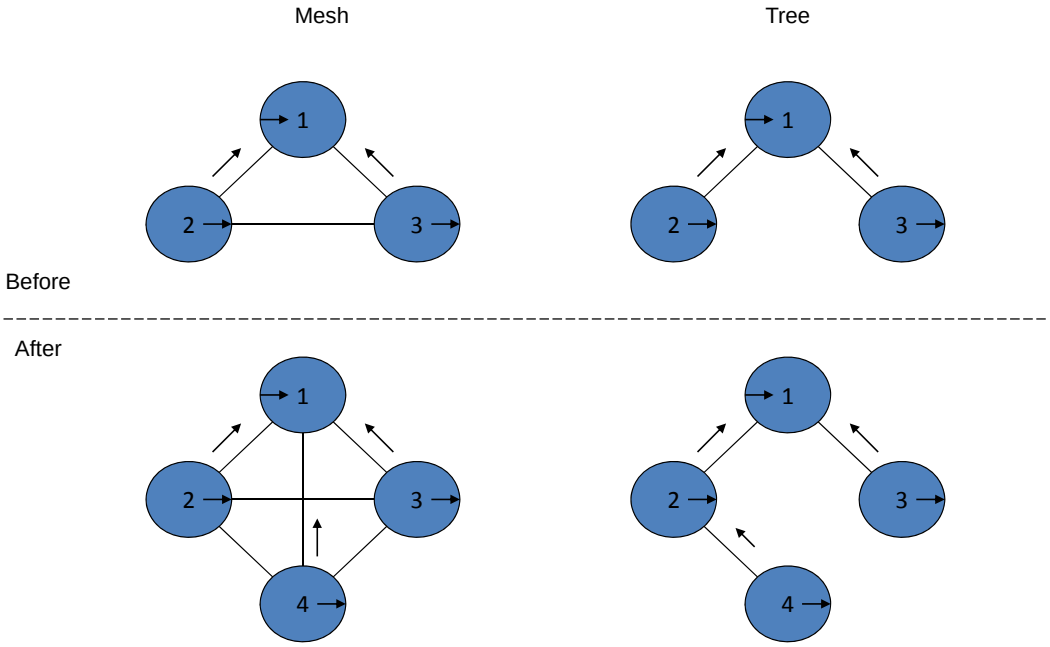


Figure 7-10: Adding a New Continuum to a RAMS Network

When a continuum leaves the RAMS network, any trails that exist solely for that continuum's publishers or subscribers are removed.

Therefore, at any given time, the trail system is the minimum set of trails required to connect all publishers to all subscribers.

7.6.4 ESTABLISHING A TRAIL SYSTEM FOR A SUBSCRIPTION—DETAILED VIEW

For each subscription, each gateway that plays a role in delivering published messages to subscribers is referred to as a *relevant* gateway. At any given time, each relevant gateway has a *petition* for the subscription. Each petition contains the following information:

- the subscription itself (including the publishing domain—either a specific continuum or 'all continua');
- which local modules have asked the petitioning gateway to forward published messages to them (these are the 'Destination Module Set' or DMS);
- which neighbor gateways have asked the petitioning gateway to forward published messages to them (these are the 'Destination Gateway Set' or DGS).

Using the above information, plus the knowledge of the network topology type (i.e., mesh or tree), and the conduit to each reachable continuum, for a given petition, each gateway can determine the following: 'Which continua must forward published messages directly to me so that I can fulfill the requests made by the members of my DMS and DGS?' (These are the 'Source Continuum Set' or SCS.)

NOTE – If this gateway's continuum-ID is in this set, then this gateway's local modules are publishers. Members whose continuum-ID is different from this gateway's indicate neighbor gateways.

7.6.5 CALCULATING A PETITION'S SOURCE CONTINUUM SET

There are many possible algorithms for calculating a petition's SCS; they should all arrive at the same answer, but through different logical paths. If the goal is to have the algorithm be as intuitive as possible, perhaps the following is a good choice (see figures 7-11 and 7-12 for examples):

The set is initially empty.

Concept: If a gateway has at least one local subscriber, then the gateway must extend trail(s) outwards towards the entire publishing domain. To be more precise:

Rule 1: If there is at least one member in the DMS:

a. If the publishing domain is a single continuum, that continuum's conduit is added to the set.

b. If the publishing domain is 'all continua', all neighbors are added to the set.

Concept: If a gateway has at least one remote subscriber and the publishing domain includes its local continuum, then the gateway must extend a trail inward to its local modules. To be more precise:

Rule 2: If there is at least one member in the DGS, and the publishing domain includes the gateway's local continuum, then the local continuum is added to the set.

NOTE – If the network is a mesh, no further calculations are necessary. If the network is a tree, a gateway may be an intermediate hop between publishers and subscribers. In that case, the next concept applies.

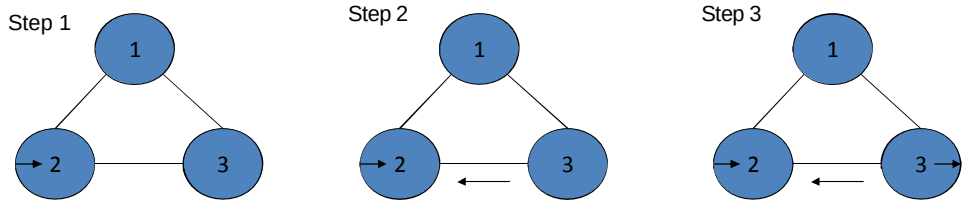
Concept: If the network is a tree, and a gateway has at least one remote subscriber, and the publishing domain includes at least one continuum other than the gateway's local continuum, then the gateway may need to extend trail(s) toward the publishing domain. To be more precise:

Rule 3: If the network is a tree, and there is at least one member in the DGS:

a. If the publishing domain is a single continuum that is not the local continuum, the continuum's conduit is added to the set.

b. If the publishing domain is 'all continua', all neighbors are added to the set.

The following examples shown in figures 7-11 and 7-12 illustrate the use of the calculated SCS in establishing trail systems for RAMS message propagation.



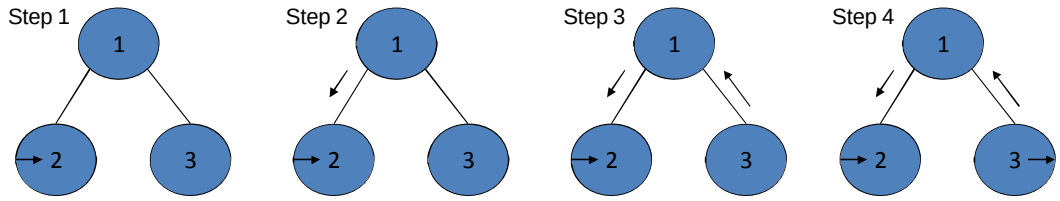
Step 1—A local node in continuum #2 asserts a subscription for subject 23 from continuum #3.

Step 2—Gateway #2 receives the assertion, adds the local node to its DNS, and calculates the required SCS. Citing rule 1a, it adds gateway #3 to its SCS and sends an assertion.

Step 3—Gateway #3 receives the assertion, adds gateway #2 to its DGS, and calculates the required SCS. Citing rule 2, it adds its own continuum (#3) to the SCS and issues a local subscription assertion.

The resulting trail system is a single trail that goes from publishers in continuum #3 to gateway #3 to gateway #2 to the subscriber in continuum #2. It should be noted that trails are established in a direction that leads from subscribers to publishers, and then traveled in the reverse direction to publish messages.

Figure 7-11: Using SCS to Establish a Trail System (Mesh Network)



Step 1 – A local node in continuum #2 asserts a subscription for subject 23 from continum #3.

Step 2 – Gateway #2 receives the assertion, adds the local node to its DNS, and calculates the required SCS. Citing rule 1a, it adds gateway #1 to its SCS and sends an assertion.

Step 3 – Gateway #1 receives the assertion, adds gateway #2 to its DGS, and calculates the required SCS. Citing rule 3a, it adds gateway #3 to the SCS.

Step 4 – Gateway #3 receives the assertion, adds gateway #1 to its DGS, and calculates the required SCS. Citing rule 2, it adds its own continuum (#3) to the SCS and issues a local subscription assertion.

The resulting trail system is a single trail that goes from publishers in continuum #3 to gateway #3 to gateway #1 to gateway #2 to the subscriber in continuum #2.

Figure 7-12: Using SCS to Establish a Trail System (Tree Network)

While rule 3b intuitively may seem correct, there is one situation in which it adds a trail that is both unnecessary and useless. Figure 7-13, in which a local module in continuum 2 has subscribed to subject 23 from all continua, shows an example of this situation—the diagram on the left shows the correct trail system; the diagram on the right shows the trail system produced when the above rules are applied. Rule 3b causes gateway #1 to add gateway #2 to its SCS. This trail is unnecessary and useless—it would cause gateway #2 to forward published messages to gateway #1, and gateway #1 to forward them back to gateway #2 (and nowhere else).

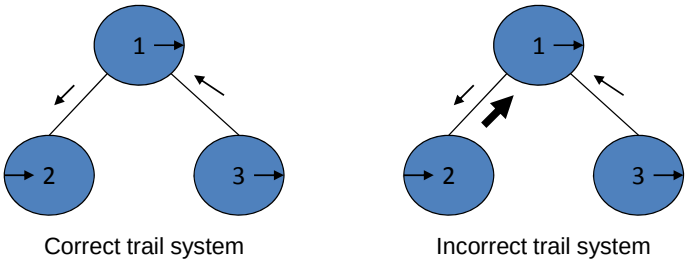


Figure 7-13: Example of Need for Rule 3c

This can be fixed by adding rule 3c, which basically says, ‘if there is only one destination, that destination is not a required source’:

c. If the DMS is empty, and the DGS has exactly one member, and that member is in the SCS (as calculated so far), then it is removed.

7.6.6 RESPONDING TO CHANGES IN A PETITION'S SOURCE CONTINUUM SET

Whenever a petition's SCS grows, an assertion is sent to each new member:

– If the new member is the local continuum, then the petition's subscription is asserted locally (via the MAMS protocol).

– If the new member is a neighbor gateway, then a petition assertion is sent to that neighbor.

Whenever a petition's SCS shrinks, a cancellation is sent to each removed member:

– If the removed member is the local continuum, then the petition's subscription is cancelled locally (via the MAMS protocol).

– If the removed member is a neighbor gateway, then a petition cancellation is sent to that neighbor.

7.6.7 DIFFERENCES BETWEEN THE INTUITIVE CONCEPTS AND THE ACTUAL SPECIFICATIONS

The specifications for RAMS differ from the intuitive approach described above in the following ways:

– The intuitive approach uses a single calculation for adding or removing trails—i.e., 'for a given set of destinations there is a required set of sources'. The specifications use two separate calculations—one for adding trails, and one for removing trails. Adding trails involves computing an 'assertion set', and removing trails involves computing a 'cancellation set'.

– The intuitive approach lumps all sources together—i.e., the SCS can include the local continuum and/or neighbor gateways. The specifications use a Source Gateway Set (SGS) that includes only neighbor gateways. Calculations for computing the 'assertion set' and 'cancellation set' produce the SGS. Additional (separate) logic is used to add or remove the local continuum.

7.7 ADDING A CONTINUUM TO A RAMS NETWORK

7.7.1 GENERAL

When a new gateway starts up and wants to join the network, the following things must happen:

a) All the existing network members must be made aware of the new member.

b) The new member must be made aware of all the existing members.

c) The new member must be made aware of any existing subscriptions whose publishing domain includes the new member's continuum (for example, any subscription whose publishing domain is 'all continua' will have to be propagated to the new gateway).

7.7.2 TELLING THE EXISTING MEMBERS ABOUT THE NEW MEMBER

In the trail system that is used to deliver published messages, the trails are generated by propagating subscriptions from subscribers to publishers. The trails are then traveled (in reverse) to deliver messages from publishers to subscribers.

The same approach can be used to establish/maintain trail systems that provide paths from each gateway to all the other gateways. Basically, each gateway is the lone subscriber for a subscription whose subject is 'my continuum' and whose publishing domain is 'all continua'—i.e., each gateway wants all the other gateways to be able to reach it. The trail network generated by propagating that subscription will lead from the 'subscribing gateway' to every other gateway in the network. The trails are then traveled (in reverse) to deliver messages from anywhere in the RAMS network to the new gateway.

A petition that is used for membership changes is referred to as a 'declaration petition', and is distinguished by having a negative subject number. For example, the subscription associated with a declaration petition for continuum #7 will have a subject of -7. The publishing domain of a declaration petition is always 'all continua'. At any time, there will be one declaration petition for each gateway that is in the network, and each of those petitions will have been propagated to every other gateway in the network.

Whenever a gateway receives an assertion of a declaration petition, it knows that there is a new reachable continuum, and it knows that the conduit to the new reachable continuum is whichever neighbor gateway sent the petition.

7.7.3 PROPAGATING EXISTING SUBSCRIPTIONS TO THE NEW MEMBER AND TELLING THE NEW MEMBER ABOUT EXISTING MEMBERS

The trail systems of existing subscriptions whose domain includes the new continuum will have to be expanded to reach publishers from the new continuum. Conceptually, the mechanism for doing this is as follows: whenever a gateway learns of a new reachable continuum, it recalculates the sources required for each of its existing petitions and sends any required assertions. (For details, refer to the previous sections on 'Calculating a Petition's Source Continuum Set', 'Responding to Changes in a Petition's Source Continuum Set', and 'Differences between the Intuitive Approach and the Actual Specifications'). It should be noted that the 'existing petitions' include the declaration petition for each existing gateway—this is how the new gateway finds out how to reach each of the existing gateways.

7.8 REMOVING A CONTINUUM FROM A RAMS NETWORK

7.8.1 CONCEPTUAL VIEW

RAMS refers to a continuum that leaves a RAMS network as a *retracting* continuum. Visually, each gateway can be pictured as the body of an octopus; the octopus has as many legs as the gateway has neighbors. Each leg consists of a neighbor and all the other gateways that are reachable via that neighbor. Figure 7-14 shows how when a gateway is removed from a network, each of the legs becomes a (smaller) network.

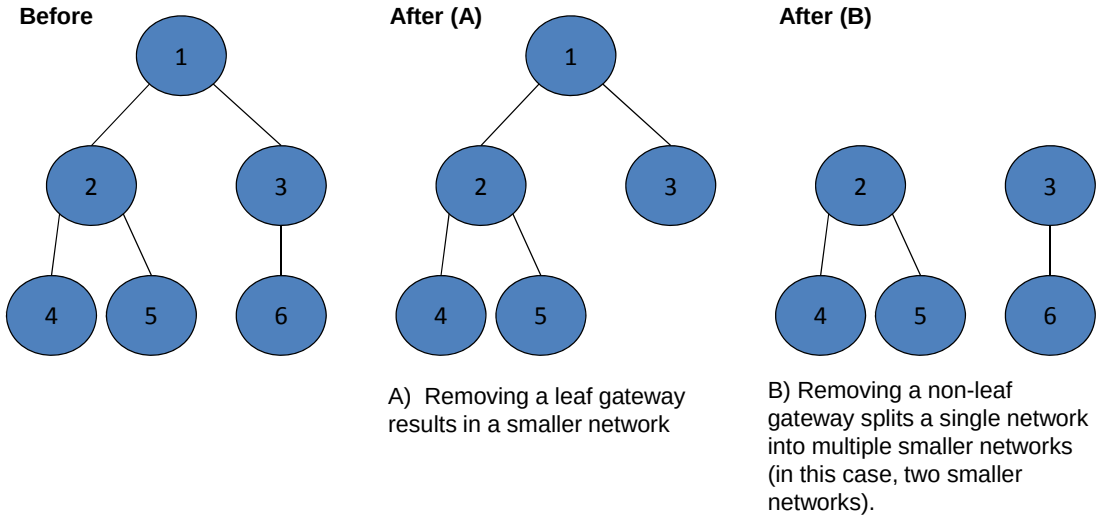


Figure 7-14: Removing a Gateway from a RAMS Network

When an existing continuum (gateway) leaves a RAMS network, the following must happen:

a) All of the existing network members must be made aware that the retracting continuum is no longer a member (for example, in figure 7-14 case B, everyone must be made aware that continuum #1 is no longer a member).

b) All of the existing network members must be made aware that any network member that was reachable via the retracting continuum is no longer a member (for example, in figure 7-14 case B, gateways #2, #4, and #5 must be made aware that continua #3 and #6 are no longer members, and gateways #3 and #6 must be made aware that continua #2, #4, and #5 are no longer members).

c) The retracting member must be made aware that all of the other network members are no longer reachable (however, its own local modules have not necessarily gone away).

d) For each existing subscription, any trails that go to or from the retracting member must be removed. This includes trails from publishers in the retracting continuum, trails to subscribers in the retracting continuum, and (if the network is a tree) trails that pass through the retracting continuum on their way to other continua.

8 TRANSPORT SERVICES

8.1 OVERVIEW

The mechanism for communications between entities in AMS is referred to as a *transport service*. AMS is designed to operate independently of the transport service, or services, chosen.

A single transport service for all configuration (MAMS) traffic, known as the *Primary Transport Service (PTS)*, is defined in any given continuum. Any additional transport services are referred to as *Supplementary Transport Services (STSes)*.

STSes are intended to provide optimized communications channels for AAMS messages between supported modules in a message space. All supported STS and corresponding network (or equivalent) addresses are specified by each module during registration as part of that module's *delivery vector*. An STS may be chosen in place of the PTS based on availability, network congestion, or efficiency related to application requirements.

8.2 CHOOSING A TRANSPORT SERVICE

The AMS protocol does not explicitly impose limitations on the choices for PTS or STS in a potential implementation. It does, however, make certain assumptions on the capabilities of the chosen transport services that the developer should be aware of in choosing an appropriate transport service.

Header specifications for the two types of intra-continuum message traffic supported by AMS, AAMS, and MAMS, are not self-identifying. This means that the two protocols used are not directly compatible and, if using the same transport service, must be configured to use different port numbers (or equivalent endpoint identifiers). A proposed way to address this is that transport services should instead include in all message headers an identifying message type, so that the same transport service endpoint could be used to send and receive messages of both types. This extension is not currently defined in the AMS specification.

It is assumed that each module will have a unique network address. The PTS may require an entity to identify the originator of a given message to enable a direct message reply, primarily for fault notifications. A standard network-based transport service, such as UDP or TCP/IP, is recommended for use as the PTS. Alternative transport services, such as message queues, can work well as supplementary transport services but could introduce additional complexities if used as the PTS.

The following subsection provides additional information and recommendations on the use of message queues as a transport service based on experiences with existing implementations.

8.3 MESSAGE QUEUES

Message queues provide the most efficient method of communications in an onboard message bus. AMS transport service implementations have been written using both the Vxworks message queues (vxmq) and POSIX message queues (Pxmq). POSIX provides support for priority-based message queues and a level of operating-system independence.

Message queues are not generally recommended for use as a PTS if alternative network stack-based communications are available. Use of message queues as a PTS is possible, but it requires implementation-specific extensions to support allocation and management of these resources. Additional efforts may be required to ensure proper reallocation of resources upon termination of an AMS entity.

One approach to allow the use of queues as a PTS modifies the module registration procedure to obtain its associated message queues from a central resource (a public read-only queue) initialized by the system—or the first configuration server—on startup prior to any other AMS activity. This method allows for messaging resources to be pre-allocated to help ensure a deterministic system with finite resources. Additional procedures may be required for recovery of endpoint resources from 'dead' modules that do not terminate gracefully, depending on system requirements or operating procedures.

Optionally, this method enables module IDs to be inferred directly from the module identifier. Registrars continue to manage module status and the listing of active or inactive modules, but may not be responsible for assigning module identifiers if the user chooses to link module ID and queue identifier directly. This direct association is useful if named, globally addressable message queues are used.

The following are suggested naming conventions for named (POSIX) message queues if used as a primary transport service. Named message queues are primarily used in Memory Management Unit (MMU)-protected operating environments. The AAMS module endpoint recommendation is also valid when queues are used as an STS.

– The module '*/AmsModuleList*' may be used for listing available (and allocated) potential module endpoints. Nascent modules will query this location to determine their endpoint addresses prior to initialization. The data used should contain a unique identifier implying the AAMS and MAMS endpoint names, or explicit identifiers for each required queue. Additional list queues may be used if required for the management of multiple registrars.

– Registrars' MAMS endpoints may be identified as '*/AmsRS#*', where # is a unique identifier given to each registrar (i.e., from an RS list queue) upon initialization. The configuration server should be located at '*/AmsCS*'.

– Modules may be identified as '*/AmsN#*', with # representing the module ID. The associated MAMS endpoint when used as a PTS may be identified as '*/AmsNM#*', where # is the module ID.

9 LATENCY IN RECONNECTION TO A REPLACEMENT CONFIGURATION SERVER

In a real-time system it is important to be able to calculate how long failovers and reconfigurations take. The following is an algorithm for estimating the worst-case elapsed time between failure of an AMS configuration server and the restoration of module registration activity.

Subsection 4.2.2 of the AMS Blue Book (discussing configuration server interrogation) defines a standard method for interrogating the configuration server, expressing the interrogation in terms of an abstract procedure-specific query and procedure-specific response.

In the case of configuration server failover, which is what the AMS Blue Book's subsection 4.2.7.1.2 initiates, the applicable procedure is registrar (re-)initialization as defined in the book's subsection 4.2.3. That method is, in essence, walking through the list of all well-known network locations for the configuration server; for each location, the registrar sends the applicable query, waits N1 seconds for a response, and on lack of response moves on to the next, cycling endlessly until either the registrar is terminated or a configuration server starts up at a well-known network location.

So the worst-case total registrar recovery time in the event of a configuration server failure should ideally be (N6 x N3) seconds to discover the crash + ((Q - 1) x N1) seconds to locate the new configuration server, where Q is the length of the well-known network connections list.

Of course, that is assuming that the configuration server actually got restarted at location J before the registrar reached element J in the list. This is not necessarily true.

More generally, then:

- Say the well-known network locations for the configuration server are L1, L2, …, LQ, and the location of the configuration server prior to crash is LC, where 1 <= C <= Q.

- Say that the registrar somehow *__reports__* the configuration server crash when it detects that crash after (N6 x N3) seconds, and the configuration server is then automatically restarted at location LD within the next (R x N1) seconds following detection of the configuration server crash, for some value of R, R >= 1, but not necessarily 1. This generally will be true no matter what mechanism (automated or manual) is used to respond to the server crash and restart the server.

- Let Z be the nominal number of network locations the registrar will have to interrogate in order to reinitialize. Our first guess is that Z = D - C + (D < C ? Q : 0) [we might wrap around]. But if Z = R, then the configuration service will not quite have gotten restarted at LD by the time the registrar interrogates LD; so the minimum value for Z is really R + 1. So in reality Z = max(R + 1, D - C + (D < C ? Q : 0)).

- So the worst-case total registrar recovery time is, in general, (N6 x N3) + (N1 * (max(R + 1, D - C + (D < C ? Q : 0)) - 1)).

ANNEX A

ACRONYMS

AAMS	Application AMS
AMS	Asynchronous Message Service
C3I	Command, Control, Communications & Information
CCSDS	Consultative Committee on Space Data Systems
CS	Configuration Server
DGS	Destination Gateway Set’
DMS	Destination Module Set
DTN	Delay/Disruption Tolerant Networking
FDIR	Fault Detection Isolation and Recovery
FIFO	First In First Out
LAN	Local Area Network
MADP	Meta-AMS Delivery Point
MAMS	Meta-AMS
MIB	Management Information Base
MMU	Memory Management Unit
MOIMS	Mission Operations & Information Management Services
MPDU	Meta-AMS PDU
MSB	Most Significant Bit
MTS	Message Transfer Service
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
PTS	Primary Transport Service
QoS	Quality of Service

RAMS	Remote AMS
RPDU	RAMS PDU
RS	Registration Server
SAP	Service Access Point
SCS	Source Continuum Set
SCPS-TP	Space Communications Protocol Specifications-Transmission Protocol
SDU	Service Data Unit
SGS	Source Gateway Set
SIS	Space Internetworking Services
SM&C	Spacecraft Monitoring and Control
SMCP	System Monitor and Control Protocol
SOIS	Spacecraft Onboard Information Services
STS	Supplementary Transport Service
TCP/IP	Transmission Control Protocol/Internet Protocol
TS	Transport Service
UDP	User Datagram Protocol
WAN	Wide Area Network