



INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

H.323

Annex R
(07/2001)

SERIES H: AUDIOVISUAL AND MULTIMEDIA SYSTEMS

Infrastructure of audiovisual services – Systems and
terminal equipment for audiovisual services

Packet-based multimedia communications systems

**Annex R: Robustness methods for H.323
entities**

ITU-T Recommendation H.323 – Annex R

(Formerly CCITT Recommendation)

ITU-T H-SERIES RECOMMENDATIONS
AUDIOVISUAL AND MULTIMEDIA SYSTEMS

CHARACTERISTICS OF VISUAL TELEPHONE SYSTEMS	H.100–H.199
INFRASTRUCTURE OF AUDIOVISUAL SERVICES	
General	H.200–H.219
Transmission multiplexing and synchronization	H.220–H.229
Systems aspects	H.230–H.239
Communication procedures	H.240–H.259
Coding of moving video	H.260–H.279
Related systems aspects	H.280–H.299
SYSTEMS AND TERMINAL EQUIPMENT FOR AUDIOVISUAL SERVICES	H.300–H.399
SUPPLEMENTARY SERVICES FOR MULTIMEDIA	H.450–H.499

For further details, please refer to the list of ITU-T Recommendations.

Packet-based multimedia communications systems

ANNEX R

Robustness methods for H.323 entities

Summary

This annex specifies methods that can be used by H.323 entities to implement robustness or tolerance to a specified set of faults. Methods for recovery of call signalling (ITU-T H.225.0) and call control signalling (ITU-T H.245) channels are specified. RAS (ITU-T H.225.0) does not involve a connection and recovery, involving registering with an alternate gatekeeper, is covered elsewhere and so not specified in this annex. Recovery of Annex G service relationships is for further study.

Source

Annex R to ITU-T Recommendation H.323 was prepared by ITU-T Study Group 16 (2001-2004) and approved under the WTSA Resolution 1 procedure on 29 July 2001.

FOREWORD

The International Telecommunication Union (ITU) is the United Nations specialized agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of ITU. ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Assembly (WTSA), which meets every four years, establishes the topics for study by the ITU-T study groups which, in turn, produce Recommendations on these topics.

The approval of ITU-T Recommendations is covered by the procedure laid down in WTSA Resolution 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation, the expression "Administration" is used for conciseness to indicate both a telecommunication administration and a recognized operating agency.

INTELLECTUAL PROPERTY RIGHTS

ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

© ITU 2002

All rights reserved. No part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from ITU.

CONTENTS

	Page
Annex R – Robustness methods for H.323 entities	1
R.1 Introduction and scope.....	1
R.2 Normative references	1
R.3 Definitions	2
R.4 Abbreviations.....	2
R.5 Overview of the two methods.....	2
R.5.1 Method A: State recovery from neighbours	3
R.5.2 Method B: State recovery from a shared repository.....	3
R.5.3 Comparison.....	3
R.6 Common mechanisms.....	4
R.6.1 Detection of TCP based connection lost.....	4
R.6.2 Protocol failure handling	4
R.6.3 Detecting failure – KeepAlive.....	4
R.6.4 Transport address and re-established connections.....	5
R.6.5 Support for extended status	6
R.7 Method A: State recovery from neighbours	6
R.7.1 Introduction	6
R.7.2 Scope	6
R.7.3 The robustness procedure	6
R.7.4 SDL for Method A state machine.....	8
R.8 Method B: State recovery from a shared repository	10
R.8.1 Fault-tolerant platform.....	10
R.8.2 Fault-tolerant cluster.....	10
R.8.3 Call signalling connection re-establishment.....	10
R.8.4 H.245 connection re-establishment	11
R.8.5 Data items shared through shared repository	11
R.8.6 Checkpoints	12
R.9 Interworking between robustness methods.....	12
R.10 Procedures for recovery	12
R.11 GenericData usage	12
R.11.1 GenericData usage in H.225.0 messages.....	13
R.12 Informative Note 1: Background on robustness methods.....	14
R.12.1 Types of robustness methods.....	14
R.12.2 Robust entities	14
R.12.3 Robust system scope.....	14
R.12.4 System termination and failures	14

	Page
R.13 Informative Note 2: Call state sharing between an entity and its backup peer.....	16
R.13.1 Shared-memory	16
R.13.2 Shared-disk	16
R.13.3 Message passing	16

ITU-T Recommendation H.323

Packet-based multimedia communications systems

ANNEX R

Robustness methods for H.323 entities

R.1 Introduction and scope

This annex specifies methods that can be used by H.323 entities to implement robustness or tolerance to a specified set of faults. Methods for recovery of call signalling (ITU-T H.225.0) and call control signalling (ITU-T H.245) channels are specified. RAS (ITU-T H.225.0) does not involve a connection and recovery, involving registering with an alternate gatekeeper, is covered elsewhere and so not specified in this annex. Recovery of Annex G service relationships is for further study.

H.323 calls require the cooperation of two or more H.323 entities. Call state information is distributed among the various entities involved in the call. Call signalling may depend on persistent connections between some of the entities involved. If any entity fails and does not have a backup peer, it may not be possible to establish new calls. If any entity involved in an active call fails and the entity does not have a backup peer or that peer does not have a method of retrieving sufficient call state information, it may not be possible to continue with the call. H.323 provides some support for building robust systems but the mechanisms are distributed throughout this annex and few, if any, procedures for using them are given.

This annex describes two alternative methods consisting of sets of mechanisms along with procedures for using them to build systems that can recover from a significant set of specified failures. One method is more appropriate for small-scale systems, uses simpler entities and does not recover as much call state information. The other method is appropriate for large-scale systems and can recover as much state information as desired but requires more complex entities. The two methods share several mechanisms and can be used concurrently in different parts of a system.

R.2 Normative references

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of the Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published.

- [1] ITU-T H.225.0 (2000), *Call signalling protocols and media stream packetization for packet-based multimedia communication systems*.
- [2] ITU-T H.323 (2000), *Packet-based multimedia communications systems*.
- [3] ITU-T Q.931 (1998), *ISDN user-network interface layer 3 specification for basic call control*.
- [4] ITU-T X.680 (1997), *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*.

R.3 Definitions

In addition to terms defined in ITU-T H.323, the following terms are used:

R.3.1 backup entity or backup peer: A peer of an entity that can take over the entity's functions if the entity fails.

R.3.2 peer entities: Two entities of the same type in an H.323 system, e.g. two gatekeepers. The two entities may cooperate in a call (e.g. originating and terminating gatekeepers in gatekeeper-routed call signalling) or one may provide backup for the other.

R.3.3 robustness methods: Procedures and mechanisms that allow recovery from the failure of one or more H.323 entities. The extent of recovery varies between different robustness methods and may include preservation of active calls in a stable state or only the ability to place new calls. Methods described in this annex are usually able to preserve active calls.

R.3.4 signalling neighbour: Other entities to which a specific entity has direct call signalling or call control signalling connections for a specific call. For example, a gatekeeper using the gatekeeper-routing model may have a direct call signalling connection for a specific call to a gateway and to another gatekeeper. These two other entities would be the gatekeeper's signalling neighbours for that call.

R.3.5 stable calls: A call is considered stable or in a stable state after Connect has been sent or received and media channels in both directions are established (though H.245 or fast connect procedures). A call becomes unstable when Release Complete has been sent or received. Certain Facility commands used to change call signalling connections may also cause a call to be considered unstable. This version of this Recommendation offers methods to preserve only stable calls during recovery.

R.3.6 tandem entities: Two (or more) peer entities, where all but one act as backup entities for one active entity.

R.3.7 virtual entity: Two (or more) closely-coupled peer entities that collectively appear as a single entity to the rest of an H.323 system while providing fault recovery.

R.4 Abbreviations

This annex uses the following abbreviations:

CRV Call Reference Value

GK Gatekeeper

GW Gateway

RAS Registration, Admission and Status

SCTP Stream Control Transmission Protocol (IETF RFC 2960) (Informative use only)

SDL Specification and Description Language

TCP Transmission Control Protocol

UDP User Datagram Protocol

R.5 Overview of the two methods

Two robustness methods are offered by this version of this annex.

The problem we are trying to solve is to recover from a failed H.323 entity. The goal is to preserve as many active calls as possible. As a minimum we wish to preserve all calls in a "stable" state. Calls not yet fully connected or in the process of tear-down may be lost. It is also a goal to preserve most

relevant billing information, such as call start time, stop time, etc., even if maintained in the failed entity (e.g. routing gatekeeper).

It is assumed that the failed entity has one or more designated backup entities, although the small-scale solution may allow recovery when the failed entity returns to service quickly. Two basic problems must be solved to recover signalling for active calls:

- 1) Redirecting/re-establishing signalling to the backup entity.
- 2) The backup entity must recover sufficient call state information that had resided in the failed entity.

The two methods are primarily distinguished by the method of recovering state information about the active calls and the amount of information recovered.

R.5.1 Method A: State recovery from neighbours

In Method A, each entity is aware of the signalling transport addresses for backup entities for each upstream and downstream signalling neighbour. When entities become aware of the failure of their upstream or downstream signalling neighbour, they attempt to connect to one of the backup entities. The backup entity recovers minimal call state from its signalling neighbour using Status and StatusInquiry messages (enhanced with additional fields). Note that in some cases it may be necessary for the neighbour to query its neighbour for call state if it has not kept all the necessary information locally (e.g. a routing gatekeeper may not have cached open logical channel information).

The recovered call state is sufficient to continue the call (forward call signalling and call control signalling and know of open logical channels) but not sufficient to allow the recovered entity to participate in billing and some other services.

R.5.2 Method B: State recovery from a shared repository

The second architecture depends on a fault-tolerant pseudo-entity. This may be implemented by:

- 1) Using a fault-tolerant platform/OS.
- 2) A pool of non-fault-tolerant entities that share call state information through shared-memory, shared-disks, or through messages. The mechanism for sharing is not specified in this Recommendation.

The real entities in this fault-tolerant pseudo-entity must share sufficient state information with its peer entities to allow recovery of the desired call-state without any assistance from its signalling neighbours. This Recommendation will define the minimum information entities that must be shared. Any additional information that is desired to be recoverable can be shared. We note that method B will require that all entities in the pool constituting the pseudo-entity be from the same vendor since the sharing mechanism is not standard. The group would suggest one or two possible solutions and will consider recommending a standard sharing mechanism in H.323 versions beyond version 4.

More details of this architecture will be given below.

R.5.3 Comparison

Each of these two architectures has advantages, which makes the choice less than obvious. Some of the issues are listed below.

The Recovery from Neighbour approach:

- 1) Allows simpler entities.
- 2) Adds less overhead before a failure (still need keepAlive messages in some cases).

But:

- 1) Requires more changes to H.323 messages.
- 2) Makes recovery somewhat slower (due to the Status and StatusInquiry messages).
- 3) Non-scalable; only suitable for small-scale systems.

The Shared Repository approach:

- 1) Hides most of the recovery process from H.323 and so requires fewer changes to existing messages.
- 2) Makes recovery faster.
- 3) Allows future use of state-maintenance protocols that might be implemented below the H.323 application layer. (See Informative Note 2 in R.13.)
- 4) Can support recovery of billing information and other desired state information.

But:

- 1) Adds significant overhead to all signalling (before failure).
- 2) Requires more complex entities or pseudo-entities.

R.6 Common mechanisms

The two methods share several common mechanisms

R.6.1 Detection of TCP based connection lost

In case of Network failure, the first "automatic" attempt would be on the IP routing protocol level. If it does not succeed, the TCP failure will be reported to both sides (entity and its signalling neighbour, e.g. gatekeeper and endpoint). Either a network failure or failure of the signalling neighbour will appear as a TCP failure.

When the call was set up, it was determined whether the entity's neighbour supported robustness procedures.

In the case where one of the sides does not support the defined Robustness Procedure, it is suggested to release the call because of TCP connection failure.

On the Endpoint side, in case both sides support the Robustness Procedure, it is suggested to maintain a reasonable timeout for the robustness procedure to be initiated by the other side. This timeout is necessary in order to address a potential Network connectivity problem. After the timeout is expired, internal resources (consumed by the call) should be released.

R.6.2 Protocol failure handling

For entities that use this annex, if a protocol failure occurs in an H.245 Control Channel and both signalling neighbours support robustness, the channel and all associated logical channels are **not** closed (contrary to clause 8.6 of H.323). Recovery procedures of this annex are instead attempted.

R.6.3 Detecting failure – KeepAlive

Without a keepalive mechanism, entity failure or failure of the signalling connection will be known only when the connection is used. Annex E provides a keepAlive mechanism to detect the failure even with little traffic. TCP's keepAlive mechanism has too long a timeout to be of use and so with TCP failure might not be detected for an extended time under conditions of low traffic sent to the failed entity. Our small-scale solution depends on failure being detected by both signalling neighbours (connections are always established from the neighbour toward the recovered entity) and so we need keepAlive messages at the H.323-level that can be used with TCP connections. KeepAlive messages are available to be optionally used in H.245. We would specify that Status/Status Inquiry be used periodically over TCP connections to provide this keepAlive

mechanism. Although this issue is common, we will see that it is only a significant problem for the Method A, the state recovery from neighbour method.

The entity closer to the called party (destination side of connection or side that uses call reference flag = 1 in CRV used on connection – See ITU-T Q.931 for definition of the call reference flag) shall send StatusInquiry periodically (this is the direction of least traffic during established calls). The period should vary randomly from a configurable maximum value to one half that value in order to avoid congestion. Two seconds is the recommended default maximum, in order to allow detection of failure before other messages timeout. The maximum value shall be included in the StatusInquiry as timeToLive, so that the recipient can also monitor failure without an additional StatusInquiry/Status exchange in the opposite direction. The recipient system needs only to maintain a timer using the indicated maximum value as a timeout.

When multiplexed channels are used, it is not necessary to send StatusInquiry/Status for each call signaled on the channel. A StatusInquiry or Status message with a CRV IE of 0 (zero) and with the field callIdentifier of 0 (zero) applies to all calls using the channel.

KeepAlive messages, especially at the H.323-level, can add significant signalling overhead. But note that only Method A with TCP connections uses these KeepAlives and Method A is for the small-scale case where the number of connections per entity is low. To minimize the overhead, the use of TCP should be avoided. StatusInquiry/Status keepAlives are **not** needed in our large-scale solution.

R.6.4 Transport address and re-established connections

Both of these solutions (with the possible exception of some fault-tolerant platform solutions) must deal with recovery of the signalling channel using a backup transport address. These must be exchanged when call signalling is established, using the backupCallSignalAddresses fields in Setup and Connect. An entity sends the call signalling address of its backup in both Setup and Connect. An entity receives the call signalling address of the backup entity from its origination-side neighbour when it receives Setup and from its termination-side neighbour when it receives Connect.

R.6.4.1 Establishment of a new TCP connection

An entity that detects loss of a call signalling channel with a signalling neighbour shall try to re-establish the channel using the backup transport address. Alternatively, the entity detecting failure may attempt to probe its original signalling neighbour using methods outside the scope of this Recommendation (e.g. ping) and, if it believes the original signalling neighbour maybe usable, it may try to re-establish the channel to the original signalling neighbour before trying the backup transport address. Implementers choosing this option should be aware that attempting to establish a TCP connection to a non-responding entity may cause significant delays.

The re-established call signalling channel will assume the state of the previous – not behave as a new channel (it will **not** begin with Setup). See further detail below for ensuring synchronization of state between signalling neighbours.

INFORMATIVE – An alternative is to use SCTP for transport rather than TCP. SCTP channels are associated with a list of alternate transport addresses that can be used as needed to maintain the channel with no intervention by the application layer. Note that more information about using SCTP is given in Informative Note 2 in R.13.

R.6.4.2 Association between the call and the new TCP connection

The association between the Call and the new TCP connection (in the endpoint side) shall be done by retrieving the callIdentifier value from the messages received on the new TCP connection.

R.6.4.3 An old TCP connection closure

After the new connection is opened, there might be two TCP connections opened, belonging to the same call on the side that did not fail. There are two options in this case:

- 1) The TCP connection was lost after the SETUP message was sent (and received). In this case the side that did not fail shall identify the situation and close the connection. This should be accomplished by detecting an identical callIdentifier for both connections.
- 2) The TCP connection was lost before the first message was transferred.

In that case, the side that did not fail has no way to find the relation between the first (old) and the second (new) TCP connections. This may be solved by a procedure that will enable the receiving side to close a connection if it is opened for a while and no message was received on it within a pre-defined timeout. (This procedure is not described in this annex.)

R.6.5 Support for extended status

To enhance interoperability between the two methods, all entities supporting robustness shall support the extended Status message including the fastStart field. This will allow an entity with a shared-repository to cooperate with a neighbour requiring Status for state recovery.

R.7 Method A: State recovery from neighbours

R.7.1 Introduction

Currently ITU-T H.323 and ITU-T H.225.0 do not explicitly define procedures for connection failure detection and recovery. The purpose of this method is to introduce a procedure for:

- Detection of TCP based connection failure.
- Synchronization between the two sides of the connection in terms of Call State.
- Definition of recommended behavior on each side in order to renew the Call Signalling connection and proceed with the call as normal in each State of the Call.

The main motivation to sustain a call (when a connection is lost) is in situations where a gatekeeper, that handles a big number of calls, fails due to hardware or software problem. In this case a control can be transferred to a standby gatekeeper (this gatekeeper may hold all the information about the calls by means of some common database). The procedure defined and presented in this annex addresses this case of Gatekeeper failure and enables the managed calls to proceed without any interruption.

This procedure does not address all the aspects of TCP based connection failure and recovery in other possible cases and topologies. Nevertheless, it is possible to address additional cases in a similar manner in the future.

R.7.2 Scope

This proposal addresses TCP based connections only (Q.931 call signalling and H.245 call control channels). UDP (RAS) channels will not be discussed since their failure situations are already covered using the retries mechanism defined for UDP channels.

R.7.3 The robustness procedure

After a failure the H.323 Entity shall re-establish the Call Signalling connection and shall send both STATUS INQUIRY and STATUS messages to the other H.323 Entity. The other H.323 Entity shall respond with a STATUS messages, thus reaching a state where both sides are aware of the Call State of the other side. The Call Signalling connection should be established to one of the entries in **backupCallSignalAddresses** in the order of preference defined by the order of elements in **backupCallSignalAddresses** structure.

In the event that both entities simultaneously initiate Call Signalling connection, the entity with the numerically smaller value of TransportAddress used from **backupCallSignalAddresses** shall close the TCP connection it opened and use the connection opened by the other endpoint. For purposes of comparing the numeric values of TransportAddress from **backupCallSignalAddresses**, each octet of the address shall be individually compared beginning with the first octet of the OCTET STRING and continuing through the OCTET STRING left to right until unequal numeric octet values are found. Comparison shall first be performed on the network-layer address element of the TransportAddress from **backupCallSignalAddresses**, and, if found to be equal, then on the transport (port) address element. See Figure R.1.

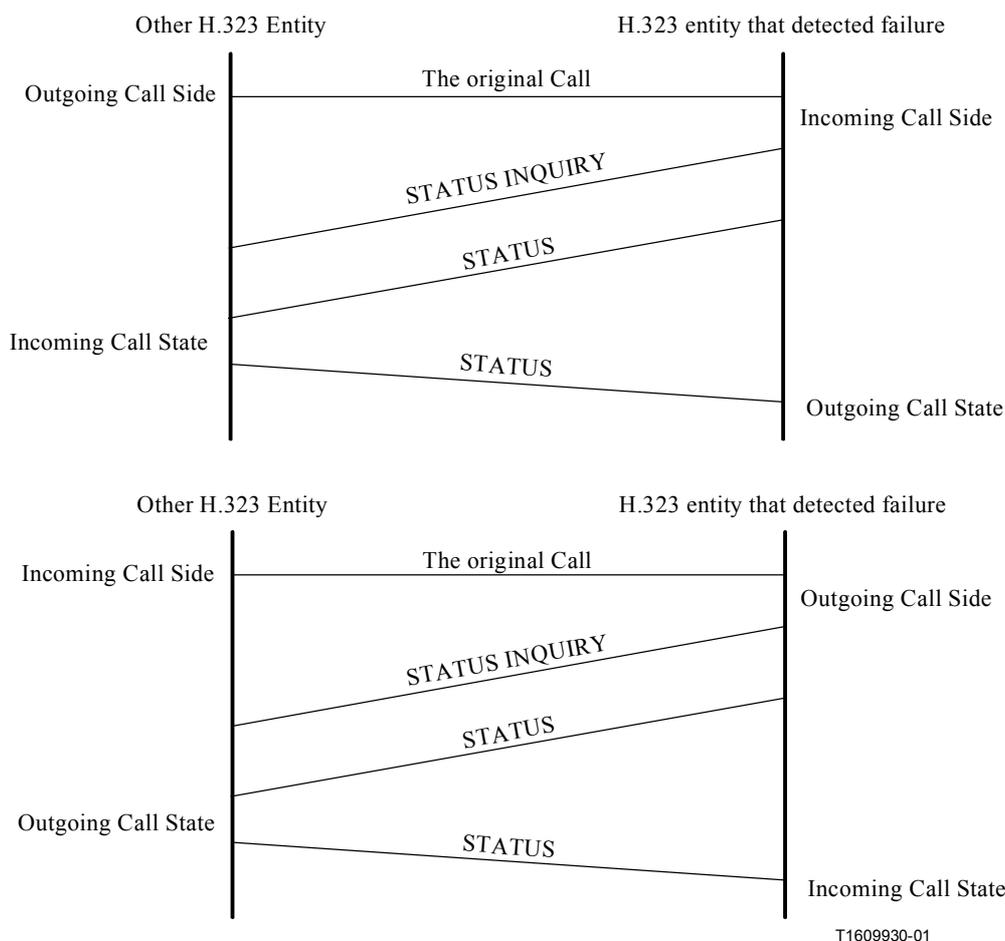


Figure R.1/H.323 – Robustness procedure

Any previous connections that might be still open for the call shall be closed, this applies both to the Call Signalling connection and the Call Control connection.

To facilitate synchronization of the logical channels state, the new fields **IncludeFastStart** in STATUS INQUIRY and **RobustnessFastStart** in STATUS message may be used. Sender of the STATUS message should include the **RobustnessFastStart** field containing the currently active receive and transmit channels with the receive addresses for media and media control streams. Sender of the STATUS INQUIRY message may request inclusion of the **RobustnessFastStart** field in the STATUS message by setting **IncludeFastStart** to TRUE.

If an intermediate entity needs to synchronize the logical channel state it should send the STATUS INQUIRY message to one of the call legs, should wait to STATUS message with fast start field, should issue the STATUS and STATUS INQUIRY message to the other leg of the call, should wait for STATUS message with the second leg logical channel information and the should send the STATUS message to the first leg of the call.

This procedure is used to synchronize the states of the logical channels that were opened through both fast start procedure and H.245 logical channel establishment procedure.

In situations where the call before failure has not reached the active state, the call should be dropped.

R.7.4 SDL for Method A state machine

See Figure R.2.

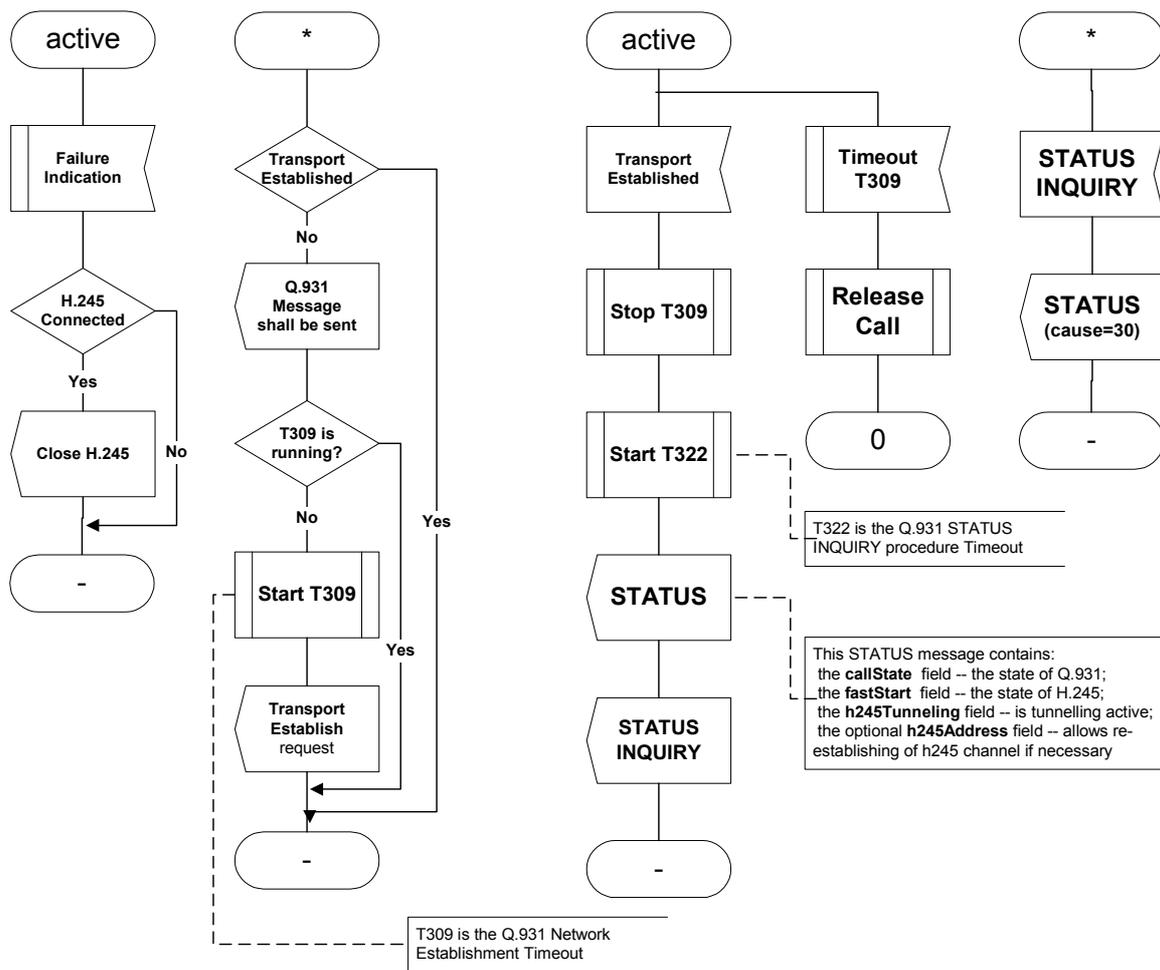


Figure R.2/H.323 – Method A state machine (sheet 1 of 2)

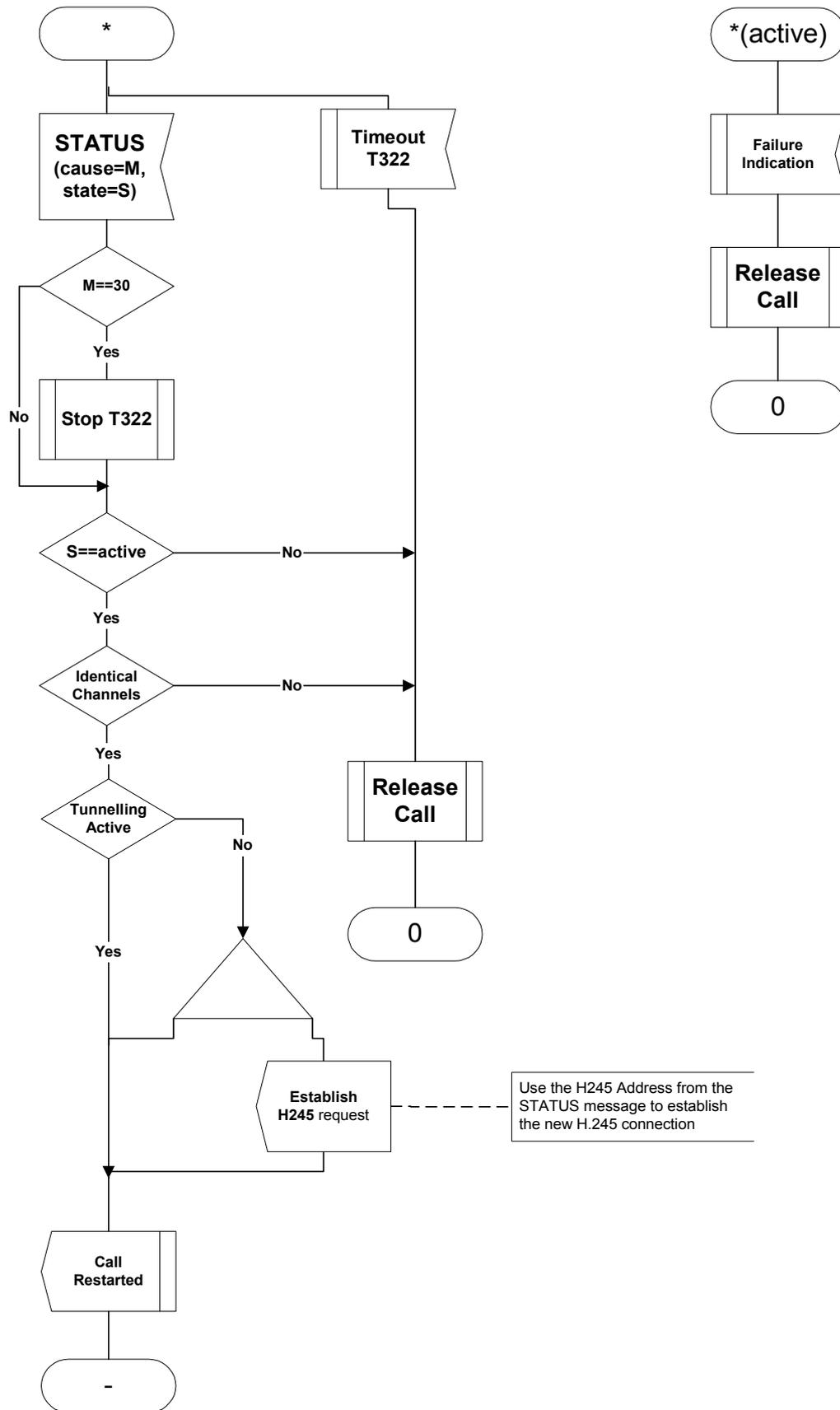


Figure R.2/H.323 – Method A state machine (sheet 2 of 2)

R.8 Method B: State recovery from a shared repository

This method depends on a fault-tolerant entity or pseudo-entity and (if the backup entity requires a different signalling address) a mechanism to re-establish call signalling to the backup. There are several ways this can be done. The fault-tolerant mechanism will not be standardized in this version of this Recommendation but we will suggest some solutions. We may recommend standardizing the solution in a future version of this Recommendation. There are some emerging IETF protocols that may help solve this problem but these are not yet in a state that could be referenced by H.323v4 (November, 2000).

R.8.1 Fault-tolerant platform

One solution is to implement the robust entity on a fault-tolerant platform that uses hardware and OS support. Such a solution would make state recovery completely transparent to H.323. If the platform also maintains a constant transport address then it is a fault-tolerant virtual entity, the signalling channel will not fail and no application level procedures are needed. If the transport address changes, then the mechanism of this clause will be needed.

R.8.2 Fault-tolerant cluster

Another solution is to establish a cluster (two or more) of non-fault-tolerant tandem entities, that collectively behave as a fault-tolerant pseudo-entity. The entities of the cluster would arrange to share specified call state information sufficient to allow a peer to take over in the event of the failure of the active entity. Solutions can include:

- 1) active/spare ("1+1");
- 2) single spare shared by several active entities (spare sharing state information with each active entity that it might substitute for) ("N+1");
- 3) and others configurations.

Although state information is shared, allowing the cluster to appear like a fault-tolerant virtual entity, it will not be able to maintain a constant call signalling transport address and so must use one of the mechanisms of R.8.3 to re-establish the call signalling channel.

One key problem for the cluster model is how to share state. State information must be synchronized at key times in the call, to which the system can safely fall back. We will call these times *checkpoints*. This Recommendation specifies the checkpoints and the minimal data items that must be shared. We do not suggest a standard solution for sharing in this version of this Recommendation but in Informative Note 2 in R.13, we will discuss several solutions to illustrate the practicality of this model.

R.8.3 Call signalling connection re-establishment

Sharing of backup signalling addresses is the same as with Method A. Re-establishment of call signalling connections is similar but has differences since the backup entity has sufficient information to re-establish the connection on the second side rather than wait for the other neighbour to also detect failure.

When a backup entity takes over for a failed peer and receives a message over a new connection, it would retrieve the call state (using the callIdentifier as key). This will allow it to continue supporting the call, including routing signalling, maintaining billing information, etc. An entity detecting a failure shall not re-establish the connection until it has a message to send over the connection. The backup entity will have new channels for each call that used the failed peer, unless multiplexed channels are used. The policy to re-establish only when needed will spread the re-establishments over time.

Delaying re-establishment until the channel is needed for a message and the fact that the backup entity has sufficient information to establish the new channel on the other side means that a keepAlive mechanism is not needed for Method B.

Since both the recovered entity and its signalling neighbour may re-establish the connection, there is a potential race condition but we avoid the need for keepAlive messages with TCP connections. Since traffic is greater in one direction than the other and re-establishment occurs only when there is message traffic, the race condition will be rare. We can resolve the race condition by the same methods used for H.245 channel establishment. The entity with the numerically smaller h245Address shall close the TCP connection it opened and use the connection opened by the other endpoint.

For multiplexed signalling channels, detecting a failure on any call shall imply failure of the channel. When a new channel is established it shall be used for the same set of calls as the failed channel. Note that this implies that the list of calls sharing a channel must be part of the data shared between an entity and its backup entity or entities through the shared repository. After a failure, the multiplexed channel is re-established when there is a message to send for any of the calls sharing the channel. A similar race condition exists to that in non-multiplexed channels. If two signalling channels are found handling the same set of calls or any calls from the same set, it shall drop one connection.

If an entity receives a new signalling connection with a callIdentifier matching that of an existing connection, it shall verify that the connection is from either the same entity as the earlier connection or the backup call signalling address for that same entity. If either is true, the entity receiving the new connection shall consider the earlier connection as failed and close it.

R.8.4 H.245 connection re-establishment

After the Call Signalling channel has been re-established and the robustness procedure has reached a stable state, if H.245 tunnelling was in use, the entities can continue tunnelling H.245 messages using the new Call Signalling channel.

If a separate H.245 connection was being used it may have also failed alone or along with the Call Signalling channel. If the entity has detected failure on an H.245 channel, it shall drop its connection without closing it (not sending EndSessionCommand, which would indicate to the other end that the call was over). It shall then attempt to establish a new connection by sending its h245Address in a Facility message to its signalling neighbour. An entity receiving Facility with an h245Address for a call for which it already has an H.245 channel (possibly failed but not detected) shall close that existing channel and open the new one. Neither entity shall perform H.245 initialization procedures (master slave determination and terminal capability exchange) for the new channel.

R.8.5 Data items shared through shared repository

As a minimum, the following data shall be shared through a shared repository:

- 1) backupCallSignallingAddresses;
- 2) hasSharedRepository;
- 3) callIdentifier;
- 4) openLogicalChannel structures, from H.245 or fastStart.

Additional data may be shared to support recovery of unstable calls or to allow recovery of additional data that changes during stable calls (e.g. call detail records, call timing data, billing data, authorization tokens).

R.8.6 Checkpoints

In this version of this Recommendation we only maintain calls that are in the stable state. Thus the only checkpoint needed is when entering the stable state. This occurs when Connect has been sent or received and media channels in both directions are established (through H.245 or fast connect procedures).

Entities may use additional checkpoints to support recovery of unstable calls or to allow recovery of additional data that changes during stable calls.

R.9 Interworking between robustness methods

Signalling neighbours must agree on the robustness method to be used between them. It is **not** necessary that the same method be used end-to-end.

Support for robustness (any of the methods) is indicated by the originating side entity by including RobustnessGenericData field in Setup. In addition support for Method B (Shared Repository) is indicated in hasSharedRepository field of Setup. The terminating side entity indicates its support for robustness and Method B by the same fields in Connect. The choice of Method A versus Method B is then made as indicated in clause **R.10 Procedures for Recovery**.

If an entity routing call signalling supports Method B (has a shared repository), it may be required to use Method B on one connection and Method A on the other connection it has for the same call. In this case, it follows the rules in the **Procedures for Recovery** clause independently on the two connections. If a backup entity with a shared repository receives StatusInquiry, it may reply with Status using information in the shared repository.

R.10 Procedures for recovery

- 1) If neighbour does not support Method B (Shared Repository) and TCP signalling is used, then StatusInquiry keepAlives shall be used. If entity has Shared Repository (even though neighbour does not), then it shall send StatusInquiry periodically. If entity does not have a Shared Repository, then only the entity nearer the called party shall send StatusInquiry periodically.
- 2) If an entity has a message to send on a call signalling channel (including a keepAlive StatusInquiry) and it detects failure, then it shall attempt to establish a channel to the first address in backupCallSignalAddresses (backup entity).
- 3) After a call signalling channel is re-established, if the neighbour does not have Shared Repository, Method A shall be used and the establishing entity shall send a Status (with the fastStart field) before the message waiting to be sent.
- 4) The establishing entity may also send a StatusInquiry before the message, if it wishes to audit state consistency.
- 5) If an entity with Shared Repository receives StatusInquiry, it shall send StatusInquiry to its neighbour on the other side to retrieve necessary state information (including fastStart data) unless it keeps all such data in its repository.
- 6) If an entity not having Shared Repository receives a StatusInquiry wait until it receives a Status from its neighbour on the other side (sending StatusInquiry, if necessary, to the other neighbour if signalling channel on other side is available).

R.11 GenericData usage

The data fields necessary to implement this annex's features are carried in GenericData fields of various messages as defined below. RobustnessData shall be encoded and the resulting binary data carried as a raw instance of GenericData in the specified messages.

```

RobustnessData ::= SEQUENCE
{
    backupCallSignalAddresses    SEQUENCE OF TransportAddress,
                                -- empty when not required
    h245Address                  TransportAddress OPTIONAL,
    fastStart                     SEQUENCE OF OCTET STRING OPTIONAL,
    timeToLive                    TimeToLive OPTIONAL
    hasSharedRepository           NULL OPTIONAL,
    includeFastStart              NULL OPTIONAL,
    ...
}

```

The GenericIdentifier shall be 1:

```
robustnessId GenericIdentifier ::= standard:1
```

In addition a featureDescriptor carrying the robustnessId shall be included in desiredFeatures of messages specified below.

R.11.1 GenericData usage in H.225.0 messages

Entities supporting robustness shall use GenericData related fields as follows (see Table R.1):

Table R.1/H.323 – Usage of GenericData fields for robustness data

Message	include RobustnessData in GenericData	Required fields						robustness FeatureDescr in desiredFeatures of featureSet
		hasShared Repository	backupCallSig Addresses	robustness FastStart	include FastStart	robustness TimeToLive	robustness H245Addr	
RRQ	M	M						M
RCF	M	M						M
ARQ								M
ACF								M
Setup	M	M	M					M#
Connect	M	M	M					M
Status+	M			M			M	
StatusInquiry+	M				M	M	M	
M mandatory – all others forbidden. + when used for robustness procedures. # desiredFeatures is not inside featureSet in Setup.								

All entities supporting robustness procedures shall support Status with the added RobustnessData field to enhance interoperability between the method A and B.

R.12 Informative Note 1: Background on robustness methods

This clause describes types of system failures and types of robustness from a general viewpoint. Not all types of system failures described are addressed by robustness methods in the current version of this annex. This more general view is provided to give context to the methods currently defined and help the reader understand which types of system failure are addressed. It also serves as a list of failures that might be addressed in future versions of this annex.

R.12.1 Types of robustness methods

System robustness can be provided in several ways:

- 1) hardware/operating system redundancy methods (possibly including several NIC cards);
- 2) tandem entities;
- 3) virtual entities.

R.12.2 Robust entities

Entities to be considered for robustness include essentially all H.323 entities:

- 1) Gatekeepers;
- 2) Border Elements;
- 3) Multipoint Controllers;
- 4) Possibly Multipoint Processors (for media stream failure);
- 5) Gateways (including IP-to-IP Gateways);
- 6) Firewall proxies; and
- 7) certain types of endpoints.

Not all robustness models may be suitable for all system components.

R.12.3 Robust system scope

The scope of robustness or the part of a system implementing robustness can include one or more of:

- 1) H.323 Zones (intra-zone, with one or more Gatekeepers).
- 2) H.323 Intra-Domain (intra-domain, inter-zone with several Gatekeepers).
- 3) H.323 Inter-Domains (inter-domain, with several Gatekeepers and Border Elements).

R.12.4 System termination and failures

Orderly system termination (such as an MC leaving a conference) should be catered for as well as system failure. Terminating orderly in principle allows the terminating endpoint to notify its peers thereby potentially simplifying detection but also requiring additional/slightly different mechanisms. It should be noted that the notification may not succeed due to repeated packet loss, so that the border to system failures is almost seamless.

System failure aspects are addressed in the following clauses:

R.12.4.1 Types of failures

The methods of this annex only address failures that can be detected from a protocol "on the wire" point of view. Failure of a processor on a multi-processor system with otherwise shared memory is not visible to the outside and hence is not a failure addressed by these methods. Failure of a NIC card on the other hand requires the use of a different transport address and hence is visible and is to be dealt with. The following types of failures will be visible to signalling neighbours and are targets for this work.

- 1) Full system component failure (power failure, software crash);
- 2) Partial system component failure (failure of one out of many communication interfaces);
- 3) Full network link failure (a system component is no longer reachable); and
- 4) Partial link network failure (not all system components can reach each other, but some can still communicate; this particularly includes partial connectivity and half-link failure).

It should be noted that various of these failure modes may be not only hard to detect (symmetrically) and may be indistinguishable from one another (see below).

- 5) Malicious attacks on the system – should be looked at in the context of the H.323 security work.

R.12.4.2 Failure detection

- 1) Time to detect a failure.
- 2) Ways of detecting a failure (explicit permanent surveillance vs detection upon invoking a function).
- 3) Entities responsible for/involved in detecting a failure.
- 4) Appearance of a failure to a system component/a set of system components.
- 5) Possibility to determine the type of failure.
- 6) Consistency/timing of failure detection among various system components.
- 7) Failure detection may not be transitive, i.e. from "A can/cannot talk to B" and "B can/cannot talk to C" cannot necessarily be concluded that also "A can/cannot talk to C".
- 8) How much overhead is acceptable?

R.12.4.3 Failure handling

- 1) Time to repair.
- 2) Entity initiating the repair process.
- 3) Possibility to repair the failure.
- 4) Consequences if the failure cannot be repaired.
- 5) How to ensure consistent handling of a failure by all involved entities?
- 6) How to deal with inconsistent views/detection of failures by various components (failed vs not)?
- 7) How to deal with different timing of failure detection?
- 8) How to deal with inconsistent state when handling a failure?
- 9) How to deal with gaps in state information when handling a failure?
- 10) Consequences on the overall system operation (e.g. an ongoing call).
- 11) How much overhead is acceptable?
- 12) How to deal with multiple simultaneous failures?

R.12.4.4 Failure scenarios

This clause lists many identified failure scenarios of H.323 systems. The robustness methods of this annex do not allow recovery from all of these failures but they are listed here for completeness and to give context to the list of failures that are covered by the robustness methods.

- 1) (Gatekeeper – endpoint): No relationship yet/anymore.
- 2) (Gatekeeper – endpoint): discovered but not registered.
- 3) (Gatekeeper – endpoint): discovered and registered.

- 4) In the process of call establishment:
 - a) direct;
 - b) gatekeeper-routed.
- 5) During a call/conference: ITU-T D.160: "stable state" – discuss what this means for the various protocols:
 - a) direct;
 - b) gatekeeper-routed.
- 6) In the process of call teardown:
 - a) direct;
 - b) gatekeeper-routed.

Consider the implications that arise from the various new protocols under development (H.450.x family, Annex K, Annex L of this Recommendation etc.)

Consider media streams as well as RAS/call signalling/conference control communication relationships.

R.13 Informative Note 2: Call state sharing between an entity and its backup peer

This Note suggests ways to implement call state sharing between an entity and another that serves as its backup peer. The selection of a method is not part of this Recommendation. Since the method is not standardized, peers from different vendors may not be able to serve as robust backup peers.

R.13.1 Shared-memory

If members of the cluster are physically located in the same cabinet, they may be able to use a shared (or reflective) memory device. This is similar to many fault-tolerant platforms, but might simply write to shared memory at each checkpoint rather than running a fault-tolerant OS.

R.13.2 Shared-disk

If the members of the cluster are physically located near each other, they can use a shared-disk and write state information at each checkpoint.

R.13.3 Message passing

The active entity can send a message updating the shared state to each of the other members of the cluster at each checkpoint. This implements a distributed shared memory sometimes referred to as a *bulletin board*. The messages can be sent using distinct UDP messages, multicast messages, persistent TCP links or fault-tolerant message passing protocol such as ASAP (which supports a send-to-group multicast mechanism not requiring multicast IP). This is discussed in more detail and with some suggested checkpoints in APC-1772.

R.13.3.1 SCTP/ASAP

This clause will illustrate, with an H.323 call example, the use of ASAP and SCTP for robustness purposes in an H.323 system. It will give in brief:

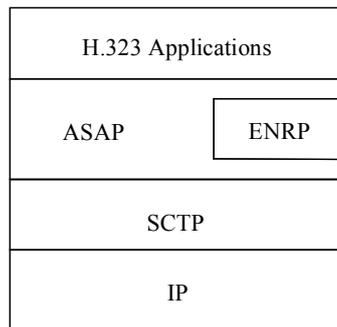
- 1) an architectural overview of an H.323 system using ASAP/SCTP;
- 2) a view of the necessary protocol stacks in the respective H.323 nodes; and
- 3) fail-over scenarios of an example H.323 call with two gatekeepers and two endpoints.

R.13.3.1.1 References

- [1] IETF RFC 2960 (2000), *Stream Control Transmission Protocol*.
- [2] R. R. Stewart, and Q. Xie: *Aggregate Server Access Protocol (ASAP)*, <draft-xierserpool-asap-00.txt>, IETF, October 2000.
- [3] Q. Xie, and R. R. Stewart: *Endpoint Name Resolution Protocol (ENRP)*, <draft-xie-rserpool-enrp-00.txt>, IETF, October 2000.

R.13.3.1.2 Protocol stacks

In general, an H.323 application using ASAP/SCTP [1] to [3] for fault tolerance will have the following protocol stack:



T1609950-01

This can provide fast fail-over, transparent to the upper layer application, at both link and session levels:

- 1) link level (SCTP) – multi-homing support, surviving network failures;
- 2) session level (ASAP) – server pool support (2N, N+K, etc.), surviving process/node failures.

In addition, ASAP provides:

- location transparency;
- load sharing;
- Plug-n-Play, i.e. hot scalability;
- avoid single point of failure.

R.13.3.1.3 An architectural view of an H.323 system

Figure R.3 shows an H.323 system built upon ASAP/SCTP model.

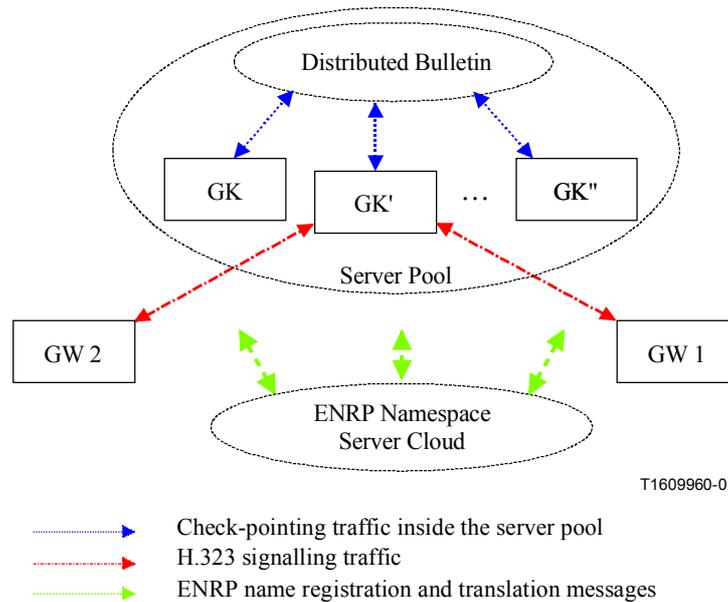


Figure R.3/H.323 – H.323 system built upon ASAP/SCTP model

In the system, all the H.323 components, including the GW1, GW2, and GKs employ the ASAP/SCTP stacks as shown in the previous clause. In this example, we assume that the H.323 gatekeeper is implemented as a server pool (the figure depicts the internals of the server pool), while the gateways may or may not be implemented as server pools.

As shown in the figure, inside the gatekeeper server pool we have multiple instances of functionally identical H.323 gatekeepers, GK, GK', ... GK''. The GK instances share call state and other call recovery critical information among themselves using an internal distributed bulletin board. The mechanism and implementation of the distributed bulletin board is vendor specific and thus out of the scope of either ASAP or SCTP (the bulletin board, however, can use ASAP/SCTP to gain fault tolerance and scalability for itself).

All the ASAP/SCTP nodes, including GWs and GKs, rely on either a single ENRP namespace server cloud or a group of bridged ENRP clouds for name registration and name translation services [2]. To form the gatekeeper server pool, all GK instances register to the ENRP namespace under the same name. However, each individual GK instance may choose to register with a different load handling capability.

Each H.323 call message will be delivered by ASAP to one of the GK instance in the server pool. The selection of the receiver GK instance is based on both the load sharing policy in effect and the current status of each GK instances in the server pool. It is sometimes very desirable to have all the H.323 signalling messages related to a call be handled by the same GK instance for the entire life cycle of the call, and only let another GK instance take over the call in case the original handler dies. We call this relationship between the call and the server instance "loose binding". ASAP is designed to support this type of "loose binding" relationship very easily [2] and [3].

Moreover, when a GK instance is handling a call, it should publish to the distributed bulletin board (i.e. "checkpoint") all critical call state information every time the call reaches a certain stage in its life cycle. This information will help the alternate GK instance to recover the call in case the original call handler crashes.

R.13.3.1.4 An example H.323 call

For the purposes of describing a call, signalling flows are used in Figure R.4.

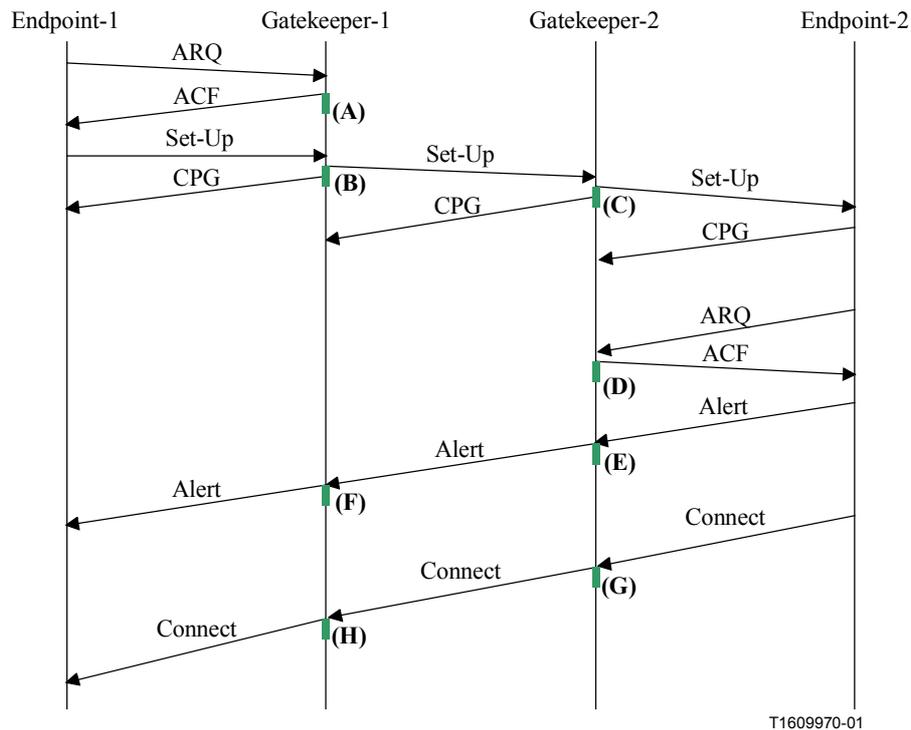


Figure R.4/H.323 – Example H.323 call

Please note that the references in this call flow are quite old and the second gatekeeper is extrapolated. There may be some differences in the way the current H.323 specification would have a call flow, but the point here is to emphasize how ASAP/SCTP would be used. Even if minor items are incorrect in the above figure, this does not invalidate the example.

R.13.3.1.4.1 General description

The call starts with Endpoint-1 requesting bandwidth. The endpoint would in this case use ASAP to query a gatekeeper, known by a name or possibly by a well-known IP address and port. In either case, an ENRP name translation query (not shown) would propagate to the endpoint the set of all gatekeepers (primary and any redundant) in the server pool. This information would be populated into a ASAP layer local cache in Endpoint-1 for future reference in case of a failure. This same caching would occur in all ASAP endpoints in the chain transparent to the call itself. Note that caching is an optional feature. Being that it is an option, endpoints not implementing it can still obtain an alternate gatekeeper, an additional query would be needed to the ENRP server at the time of failure detection.

Note we now hit point (A), at this step the gatekeeper allocates bandwidth and checkpoints this bandwidth utilization information message into a "bulletin board" area. This "bulletin board" area could be any of the following:

- a piece of distributed shared memory being maintained by a separate subsystem;
- a piece of reflective memory specifically built for this purpose;
- a distributed commercial database;
- some other creative invention.

Please note that the point here is that the redundant/peer gatekeepers have to share call states in some way. Any existing or future mechanism conceived to share call state can be used.

Gatekeeper-1 populates its ARQ related state and pushes this information to the "bulletin board" and responds to the request in the normal manner, i.e. with an ACF.

Endpoint-1 now reacts and sends the set-up message to Gatekeeper-1. Upon reception of the set-up message Gatekeeper-1 selects the next gatekeeper, Gatekeeper-2, and forwards its set-up, pushing the state information about the call (point **(B)**), possibly tied in some way to the previous information (perhaps with some form of cross-reference, i.e. Call-X is using Y bandwidth represented by the ARQ information). After pushing the information at point **(B)**, Gatekeeper-1 sends out the call proceeding message to Endpoint-1.

Gatekeeper-2 receives the set-up message from its peer gatekeeper, selects the destination endpoint forwards the set-up and pushes state information at point **(C)** for the call. After pushing its state to the bulletin board, it sends Gatekeeper-1 a call proceeding message.

Endpoint-2, upon reception of the set-up, sends back a call proceeding message and asks its gatekeeper for bandwidth with its own ARQ message.

This causes Gatekeeper-2 to allocate bandwidth, push state at point **(D)** and send back the ACF message. Upon reception of which, Endpoint-2 sends an Alerting message to Gatekeeper-2.

Upon reception of the Alerting message, Gatekeeper-2 would push a small update to its bulletin board (point **(E)**), i.e. that the call is in Alerting, and forward an Alerting message to Gatekeeper-1.

Gatekeeper-1 will repeat the same procedure, updating its state at point **(F)** and forwarding the Alerting message.

Endpoint-2 at some point answers the call, sending a Connect message to Gatekeeper-2. Gatekeeper-2, upon reception of the Connect message, will push another small update to the state at point **(G)** indicating that the call is now in an answered state and forward the connect message to Gatekeeper-1.

Upon reception of the Connect message, Gatekeeper-1 will perform the same operation, saving its state at point **(H)** and sending the connect message on to Endpoint-1.

R.13.3.1.4.2 Failure scenarios

The above descriptions assume the maximum level of redundancy and state/call preservation. In this scenario any failure of either Gatekeeper becomes transparent to either endpoint. If a failure occurs, the message would be re-routed by ASAP to an alternate. The alternate would need to take the following actions on any message it received that it did not have a call object/block for:

- look up the call in the "bulletin board";
- pull the state information and construct a call control block or object to the call;
- continue processing the message on behalf of the dead peer.

Endpoints become completely transparent to failure scenarios. No knowledge is placed in the endpoint itself (other than ASAP) to recover from a Gatekeeper failure.

R.13.3.1.4.3 State saving issues

As stated above, the example assumes a maximum state saving model. In this mode updates to state would need to be minimized to the smallest amount of information possible. In particular, state should be limited to the smallest set of information necessary to re-construct the call AND updates should be as small as possible. In some cases an operator may not wish to have this level of redundancy. To achieve a robust system with less state, the following state sharing points could be eliminated:

- At points **(A)** and **(D)** – If the gatekeeper uses some other methodology to calculate bandwidth utilization (besides tracking the number of calls by count) these steps could be completely skipped with no harm. It may be that the operator has NO concern for admission control and its gatekeepers do not perform this, in these cases this step is not necessary.
- At points **(F)** and **(E)** – These points are optional in that they may not provide any information worth saving, i.e. the call is ringing versus still setting up.
- At points **(B)** and **(C)** – If the operator is NOT interested in saving anything but stable calls, these points can be eliminated. In this case, any calls that were being set up would be lost if a failure did occur.

Trade-offs, such as the above, are outside the scope of using ASAP/SCTP and are strictly an operator/manufacture decision as to how much state may be saved by a given implementation and what controls/options the operator may have.

SERIES OF ITU-T RECOMMENDATIONS

Series A	Organization of the work of ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	Transmission systems and media, digital systems and networks
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Cable networks and transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure and Internet protocol aspects
Series Z	Languages and general software aspects for telecommunication systems