Network Working Group
Request for Comments: 2705
Category: Informational

M. Arango
RSL COM
A. Dugan
I. Elliott
Level3 Communications
C. Huitema
Telcordia
S. Pickett
Vertical Networks

October 1999

Media Gateway Control Protocol (MGCP) Version 1.0

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

IESG NOTE:

This document is being published for the information of the community. It describes a protocol that is currently being deployed in a number of products. Implementers should be aware of developments in the IETF Megaco Working Group and ITF-T SG16 who are currently working on a potential successor to this protocol.

Abstract

This document describes an application programming interface and a corresponding protocol (MGCP) for controlling Voice over IP (VoIP) Gateways from external call control elements. MGCP assumes a call control architecture where the call control "intelligence" is outside the gateways and handled by external call control elements.

The document is structured in 6 main sections:

* The introduction presents the basic assumptions and the relation to other protocols such as H.323, RTSP, SAP or SIP.

* The interface section presents a conceptual overview of the MGCP, presenting the naming conventions, the usage of the session description protocol SDP, and the procedures that compose MGCP: Notifications Request, Notification, Create Connection, Modify Connection, Delete Connection, AuditEndpoint, AuditConnection and RestartInProgress.

* The protocol description section presents the MGCP encodings, which are based on simple text formats, and the transmission procedure over UDP.

* The security section presents the security requirement of MGCP, and its usage of IP security services (IPSEC).

* The event packages section provides an initial definition of packages and event names.

* The description of the changes made in combining SGCP 1.1 and IPDC to create MGCP 1.0.

Table of Contents

1. Introduction

This document describes an abstract application programming interface and a corresponding protocol (MGCP) for controlling Telephony Gateways from external call control elements called media gateway controllers or call agents. A telephony gateway is a network element that provides conversion between the audio signals carried on telephone circuits and data packets carried over the Internet or over other packet networks. Example of gateways are:

* Trunking gateways, that interface between the telephone network and a Voice over IP network. Such gateways typically manage a large number of digital circuits.

* Voice over ATM gateways, which operate much the same way as voice over IP trunking gateways, except that they interface to an ATM network.

* Residential gateways, that provide a traditional analog (RJ11) interface to a Voice over IP network. Examples of residential gateways include cable modem/cable set-top boxes, xDSL devices, broad-band wireless devices

* Access gateways, that provide a traditional analog (RJ11) or digital PBX interface to a Voice over IP network. Examples of access gateways include small-scale voice over IP gateways.

* Business gateways, that provide a traditional digital PBX interface or an integrated "soft PBX" interface to a Voice over IP network.

* Network Access Servers, that can attach a "modem" to a telephone circuit and provide data access to the Internet. We expect that, in the future, the same gateways will combine Voice over IP services and Network Access services.

* Circuit switches, or packet switches, which can offer a control interface to an external call control element.

MGCP assumes a call control architecture where the call control "intelligence" is outside the gateways and handled by external call control elements. The MGCP assumes that these call control elements, or Call Agents, will synchronize with each other to send coherent commands to the gateways under their control. MGCP does not define a mechanism for synchronizing Call Agents. MGCP is, in essence, a master/slave protocol, where the gateways are expected to execute commands sent by the Call Agents. In consequence, this document specifies in great detail the expected behavior of the gateways, but

only specify those parts of a call agent implementation, such as timer management, that are mandated for proper operation of the protocol.

MGCP assumes a connection model where the basic constructs are endpoints and connections. Endpoints are sources or sinks of data and could be physical or virtual. Examples of physical endpoints are:

* An interface on a gateway that terminates a trunk connected to a PSTN switch (e.g., Class 5, Class 4, etc.). A gateway that terminates trunks is called a trunk gateway.

* An interface on a gateway that terminates an analog POTS connection to a phone, key system, PBX, etc. A gateway that terminates residential POTS lines (to phones) is called a residential gateway.

An example of a virtual endpoint is an audio source in an audio-content server. Creation of physical endpoints requires hardware installation, while creation of virtual endpoints can be done by software.

Connections may be either point to point or multipoint. A point to point connection is an association between two endpoints with the purpose of transmitting data between these endpoints. Once this association is established for both endpoints, data transfer between these endpoints can take place. A multipoint connection is established by connecting the endpoint to a multipoint session.

Connections can be established over several types of bearer networks:

* Transmission of audio packets using RTP and UDP over a TCP/IP network.

* Transmission of audio packets using AAL2, or another adaptation layer, over an ATM network.

* Transmission of packets over an internal connection, for example the TDM backplane or the interconnection bus of a gateway. This is used, in particular, for "hairpin" connections, connections that terminate in a gateway but are immediately rerouted over the telephone network.

For point-to-point connections the endpoints of a connection could be in separate gateways or in the same gateway.

1.1. Relation with the H.323 standards

MGCP is designed as an internal protocol within a distributed system that appears to the outside as a single VoIP gateway. This system is composed of a Call Agent, that may or may not be distributed over several computer platforms, and of a set of gateways, including at least one "media gateway" that perform the conversion of media signals between circuits and packets, and at least one "signalling gateway" when connecting to an SS7 controlled network. In a typical configuration, this distributed gateway system will interface on one side with one or more telephony (i.e. circuit) switches, and on the other side with H.323 conformant systems, as indicated in the following table:

Functional Plane	Phone switch	Terminating Entity	H.323 conformant systems
Signaling Plane	Signaling exchanges through SS7/ISUP	Call agent	Signaling exchanges with the call agent through H.225/RAS and H.225/Q.931.
			Possible negotiation of logical channels and transmission parameters through H.245 with the call agent.
		Internal synchronization through MGCP	
Bearer Data Transport Plane	Connection through high speed trunk groups	Telephony gateways	Transmission of VOIP data using RTP directly between the H.323 station and the gateway.

In the MGCP model, the gateways focus on the audio signal translation function, while the Call Agent handles the signaling and call processing functions. As a consequence, the Call Agent implements the "signaling" layers of the H.323 standard, and presents itself as an "H.323 Gatekeeper" or as one or more "H.323 Endpoints" to the H.323 systems.

1.2. Relation with the IETF standards

While H.323 is the recognized standard for VoIP terminals, the IETF has also produced specifications for other types of multi-media applications. These other specifications include:

* the Session Description Protocol (SDP), RFC 2327,

* the Session Announcement Protocol (SAP),

* the Session Initiation Protocol (SIP),

* the Real Time Streaming Protocol (RTSP), RFC 2326.

The latter three specifications are in fact alternative signaling standards that allow for the transmission of a session description to an interested party. SAP is used by multicast session managers to distribute a multicast session description to a large group of recipients, SIP is used to invite an individual user to take part in a point-to-point or unicast session, RTSP is used to interface a server that provides real time data. In all three cases, the session description is described according to SDP; when audio is transmitted, it is transmitted through the Real-time Transport Protocol, RTP.

The distributed gateway systems and MGCP will enable PSTN telephony users to access sessions set up using SAP, SIP or RTSP. The Call Agent provides for signaling conversion, according to the following table:

Functional Plane	Phone switch	Terminating Entity	IETF conforming systems
Signaling Plane	Signaling exchanges through SS7/ISUP	Call agent	Signaling exchanges with the call agent through SAP, SIP or RTSP.
			Negotiation of session description parameters through SDP (telephony gateway terminated but passed via the call agent to and from the IETF conforming system)
		Internal synchronization through MGCP	
Bearer Data Transport Plane	Connection through high speed trunk groups	Telephony gateways	Transmission of VoIP data using RTP, directly between the remote IP end system and the gateway.

The SDP standard has a pivotal status in this architecture. We will see in the following description that we also use it to carry session descriptions in MGCP.

1.3. Definitions

Trunk: A communication channel between two switching systems. E.g., a DS0 on a T1 or E1 line.

2. Media Gateway Control Interface

The interface functions provide for connection control and endpoint control. Both use the same system model and the same naming conventions.

2.1. Model and naming conventions

The MGCP assumes a connection model where the basic constructs are endpoints and connections. Connections are grouped in calls. One or more connections can belong to one call. Connections and calls are set up at the initiative of one or several Call Agents.

2.1.1. Types of endpoints

In the introduction, we presented several classes of gateways. Such classifications, however, can be misleading. Manufacturers can arbitrarily decide to provide several types of services in a single packaging. A single product could well, for example, provide some trunk connections to telephony switches, some primary rate connections and some analog line interfaces, thus sharing the characteristics of what we described in the introduction as "trunking", "access" and "residential" gateways. MGCP does not make assumptions about such groupings. We simply assume that media gateways support collections of endpoints. The type of the endpoint determines its functionalities. Our analysis, so far, has led us to isolate the following basic endpoint types:

* Digital channel (DS0),

* Analog line,

* Annoucement server access point,

* Interactive Voice Response access point,

* Conference bridge access point,

* Packet relay,

* Wiretap access point,

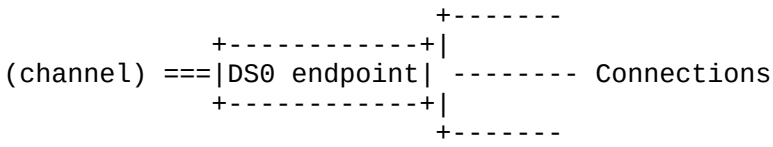
* ATM "trunk side" interface.

In this section, we will develop the expected behavior of such end points.

This list is not limitative. There may be other types of endpoints defined in the future, for example test endpoint that could be used to check network quality, or frame-relay endpoints that could be used to managed audio channels multiplexed over a frame-relay virtual circuit.

2.1.1.1. Digital channel (DS0)

Digital channels provide an 8Khz*8bit service. Such channels are found in trunk and ISDN interfaces. They are typically part of digital multiplexes, such as T1, E1, T3 or E3 interfaces. Media gateways that support such channels are capable of translating the digital signals received on the channel, which may be encoded according to A or mu-law, using either the complete set of 8 bits or only 7 of these bits, into audio packets. When the media gateway also supports a NAS service, the gateway shall be capable of receiving either audio-encoded data (modem connection) or binary data (ISDN connection) and convert them into data packets.

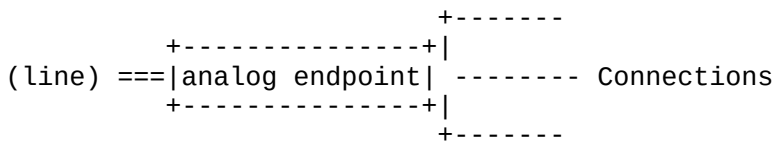


Media gateways should be able to establish several connections between the endpoint and the packet networks, or between the endpoint and other endpoints in the same gateway. The signals originating from these connections shall be mixed according to the connection "mode", as specified later in this document. The precise number of connections that an endpoint support is a characteristic of the gateway, and may in fact vary according with the allocation of resource within the gateway.

In some cases, digital channels are used to carry signalling. This is the case for example of SS7 "F" links, or ISDN "D" channels. Media gateways that support these signalling functions shall be able to send and receive the signalling packets to and from a call agent, using the "back haul" procedures defined by the SIGTRAN working group of the IETF. Digital channels are sometimes used in conjunction with channel associated signalling, such as "MF R2". Media gateways that support these signalling functions shall be able to detect and produce the corresponding signals, such as for example "wink" or "A", according to the event signalling and reporting procedures defined in MGCP.

2.1.1.2. Analog line

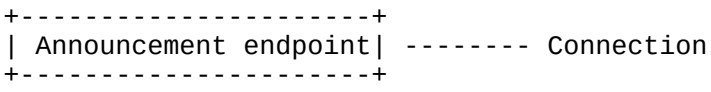
Analog lines can be used either as a "client" interface, providing service to a classic telephone unit, or as a "service" interface, allowing the gateway to send and receive analog calls. When the media gateway also supports a NAS service, the gateway shall be capable of receiving audio-encoded data (modem connection) and convert them into data packets.



Media gateways should be able to establish several connections between the endpoint and the packet networks, or between the endpoint and other endpoints in the same gateway. The audio signals originating from these connections shall be mixed according to the connection "mode", as specified later in this document. The precise number of connections that an endpoint support is a characteristic of the gateway, and may in fact vary according with the allocation of resource within the gateway. A typical gateway should however be able to support two or three connections per endpoint, in order to provide services such as "call waiting" or "three ways calling".

2.1.1.3. Annoucement server access point

An announcement server endpoint provides acces to an announcement service. Under requests from the call agent, the announcement server will "play" a specified announcement. The requests from the call agent will follow the event signalling and reporting procedures defined in MGCP.

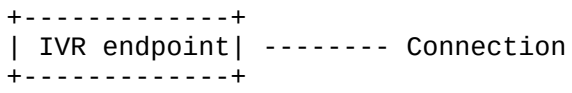


A given announcement endpoint is not supposed to support more than one connection at a time. If several connections were established to the same endpoint, then the same announcements would be played simultaneously over all the connections.

Connections to an announcement server are typically oneway, or "half duplex" -- the announcement server is not expected to listen the audio signals from the connection.

2.1.1.4. Interactive Voice Response access point

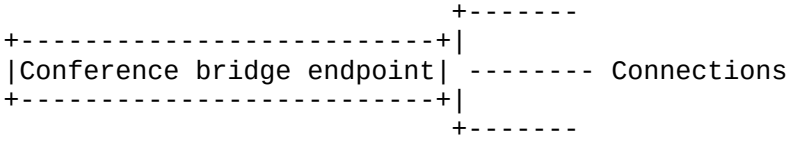
An Interactive Voice Response (IVR) endpoint provides acces to an IVR service. Under requests from the call agent, the IVR server will "play" announcements and tones, and will "listen" to responses from the user. The requests from the call agent will follow the event signalling and reporting procedures defined in MGCP.



A given IVR endpoint is not supposed to support more than one connection at a time. If several connections were established to the same endpoint, then the same tones and announcements would be played simultaneously over all the connections.

2.1.1.5. Conference bridge access point

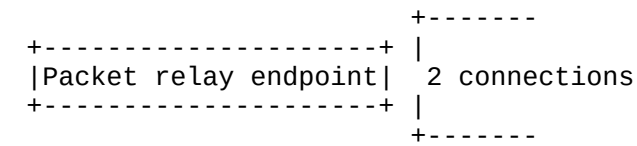
A conference bridge endpoint is used to provide access to a specific conference.



Media gateways should be able to establish several connections between the endpoint and the packet networks, or between the endpoint and other endpoints in the same gateway. The signals originating from these connections shall be mixed according to the connection "mode", as specified later in this document. The precise number of connections that an endpoint support is a characteristic of the gateway, and may in fact vary according with the allocation of resource within the gateway.

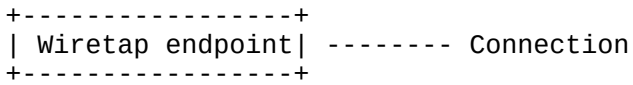
2.1.1.6. Packet relay

A packet relay endpoint is a specific form of conference bridge, that typically only supports two connections. Packets relays can be found in firewalls between a protected and an open network, or in transcoding servers used to provide interoperation between incompatible gateways, for example gateways that do not support compatible compression algorithms, or gateways that operate over different transmission networks such as IP and ATM.



2.1.1.7. Wiretap access point

A wiretap access point provides access to a wiretap service, providing either a recording or a life playback of a connection.

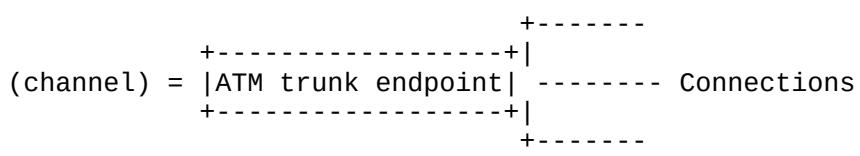


A given wiretap endpoint is not supposed to support more than one connection at a time. If several connections were established to the same endpoint, then the recording or playback would mix the audio signals received on this connections.

Connections to an wiretap endpoint are typically oneway, or "half duplex" -- the wiretap server is not expected to signal its presence in a call.

2.1.1.8. ATM "trunk side" interface.

ATM "trunk side" endpoints are typically found when one or several ATM permanent virtual circuits are used as a replacement for the classic "TDM" trunks linking switches. When ATM/AAL2 is used, several trunks or channels are multiplexed on a single virtual circuit; each of these trunks correspond to a single endpoint.



Media gateways should be able to establish several connections between the endpoint and the packet networks, or between the endpoint and other endpoints in the same gateway. The signals originating from these connections shall be mixed according to the connection "mode", as specified later in this document. The precise number of connections that an endpoint support is a characteristic of the gateway, and may in fact vary according with the allocation of resource within the gateway.

2.1.2. Endpoint identifiers

Endpoints identifiers have two components that both are case insensitive:

* the domain name of the gateway that is managing the endpoint,

* a local name within that gateway,

The syntax of the local name depends on the type of endpoint being named. However, the local name for each of these types is naturally hierarchical, beginning with a term which identifies the physical gateway containing the given endpoint and ending in a term which specifies the individual endpoint concerned. With this in mind, the following rules for construction and interpretation of the Entity Name field for these entity types MUST be supported:

1) The individual terms of the naming path MUST be separated by a single slash ("/", ASCII 2F hex).

2) The individual terms are character strings composed of letters, digits or other printable characters, with the exception of characters used as delimitors ("/", "@"), characters used for wildcarding ("*", "$") and white spaces.

3) Wild-carding is represented either by an asterisk ("*") or a dollar sign ("$") for the terms of the naming path which are to be wild-carded. Thus, if the full naming path looks like

```
term1/term2/term3
```

then the Entity Name field looks like this depending on which terms are wild-carded:

```
*/term2/term3 if term1 is wild-carded
term1/*/term3 if term2 is wild-carded
term1/term2/* if term3 is wild-carded
term1/*/* if term2 and term3 are wild-carded,
 etc.
```

In each of these examples a dollar sign could have appeared instead of an asterisk.

4) A term represented by an asterisk is to be interpreted as: "use ALL values of this term known within the scope of the Media Gateway". A term represented by a dollar sign is to be interpreted as: "use ANY ONE value of this term known within the scope of the Media Gateway". The description of a specific command may add further criteria for selection within the general rules given here.

If the Media Gateway controls multiple physical gateways, the first term of the naming MUST identify the physical gateway containing the desired entity. If the Media Gateway controls only a single physical gateway, the first term of the naming string MAY identify that physical gateway, depending on local practice. A local name that is composed of only a wildcard character refers to either all (*) or any ($) endpoints within the media gateway.

In the case of trunking gateways, endpoints are trunk circuits linking a gateway to a telephone switch. These circuits are typically grouped into a digital multiplex, that is connected to the gateway by a physical interface. Such circuits are named in three contexts:

* In the ISUP protocol, trunks are grouped into trunk groups, identified by the SS7 point codes of the switches that the group connects. Circuits within a trunk group are identified by a circuit number (CIC in ISUP).

* In the gateway configuration files, physical interfaces are typically identified by the name of the interface, an arbitrary text string. When the interface multiplexes several circuits, individual circuits are typically identified by a circuit number.

* In MGCP, the endpoints are identified by an endpoint identifier.

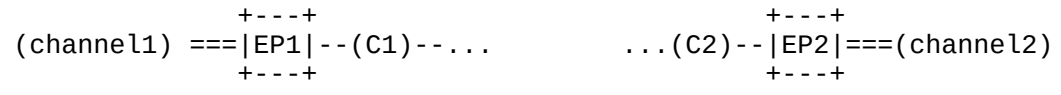
The Call Agents use configuration databases to map ranges of circuit numbers within an ISUP trunk group to corresponding ranges of circuits in a multiplex connected to a gateway through a physical interface. The gateway will be identified, in MGCP, by a domain name. The local name will be structured to encode both the name of the physical interface, for example X35V3+A4, and the circuit number within the multiplex connected to the interface, for example 13. The circuit number will be separated from the name of the interface by a fraction bar, as in:

X35V3+A4/13

Other types of endpoints will use different conventions. For example, in gateways were physical interfaces by construction only control one circuit, the circuit number will be omitted. The exact syntax of such names should be specified in the corresponding server specification.

2.1.3. Calls and connections

Connections are created on the call agent on each endpoint that will be involved in the "call." In the classic example of a connection between two "DS0" endpoints (EP1 and EP2), the call agents controlling the end points will establish two connections (C1 and C2):



Each connection will be designated locally by a connection identifier, and will be characterized by connection attributes.

When the two endpoints are located on gateways that are managed by the same call agent, the creation is done via the three following steps:

1) The call agent asks the first gateway to "create a connection" on the first endpoint. The gateway allocates resources to that connection, and respond to the command by providing a "session description." The session description contains the information necessary for a third party to send packets towards the newly created connection, such as for example IP address, UDP port, and packetization parameters.

2) The call agent then asks the second gateway to "create a connection" on the second endpoint. The command carries the "session description" provided by the first gateway. The gateway allocates resources to that connection, and respond to the command by providing its own "session description."

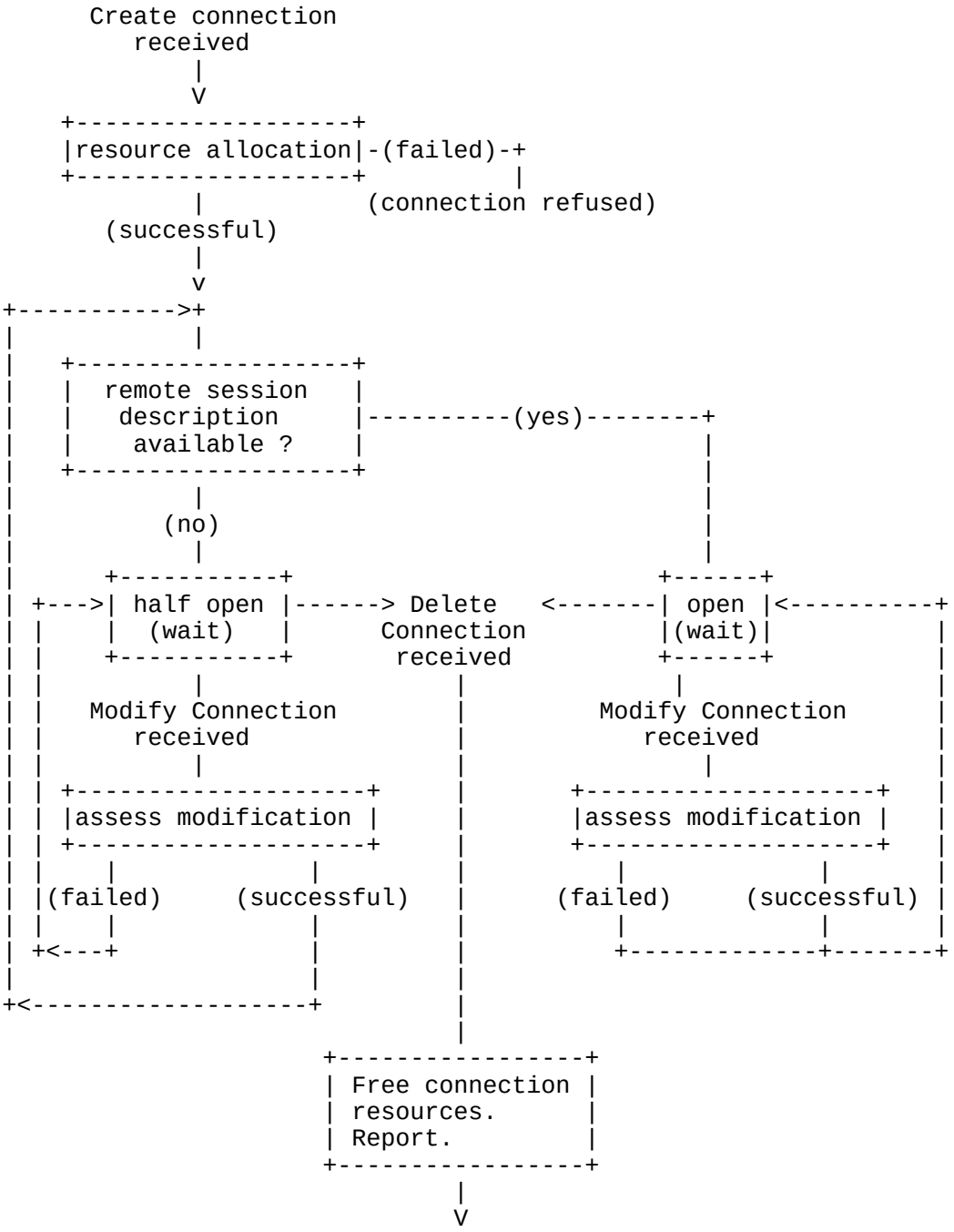
3) The call agent uses a "modify connection" command to provide this second "session description" to the first endpoint. Once this is done, communication can proceed in both directions.

When the two endpoints are located on gateways that are managed by the different call agents, these two call agents shall exchange information through a call-agent to call-agent signalling protocol, in order to synchronize the creation of the connection on the two endpoints.

Once established, the connection parameters can be modified at any time by a "modify connection" command. The call agent may for example instruct the gateway to change the compression algorithm used on a connection, or to modify the IP address and UDP port to which data should be sent, if a connection is "redirected."

The call agent removes a connection by sending to the gateway a "delete connection" command. The gateway may also, under some circumstances, inform a gateway that a connection could not be sustained.

The following diagram provides a view of the states of a connection, as seen from the gateway:


Create connection received
resource allocation
(failed)
(connection refused)
(successful)
remote session description available ?
(yes)
(no)
half open (wait)
open (wait)
Delete Connection received
Modify Connection received
Modify Connection received
assess modification
assess modification
(failed)
(successful)
(failed)
(successful)
Free connection resources. Report.

2.1.3.1. Names of calls

One of the attributes of each connection is the "call identifier."

Calls are identified by unique identifiers, independent of the underlying platforms or agents. These identifiers are created by the Call Agent. They are treated in MGCP as unstructured octet strings.

Call identifiers are expected to be unique within the system, or at a minimum, unique within the collection of Call Agents that control the same gateways. When a Call Agent builds several connections that pertain to the same call, either on the same gateway or in different gateways, these connections that belong to the same call share the same call-id. This identifier can then be used by accounting or management procedures, which are outside the scope of MGCP.

2.1.3.2. Names of connections

Connection identifiers are created by the gateway when it is requested to create a connection. They identify the connection within the context of an endpoint. They are treated in MGCP as unstructured octet strings. The gateway should make sure that a proper waiting period, at least 3 minutes, elapses between the end of a connection that used this identifier and its use in a new connection for the same endpoint. (Gateways may decide to use identifiers that are unique within the context of the gateway.)

2.1.3.3. Management of resources, attributes of connections

Many types of resources will be associated to a connection, such as specific signal processing functions or packetization functions. Generally, these resources fall in two categories:

1) Externally visible resources, that affect the format of "the bits on the network" and must be communicated to the second endpoint involved in the connection.

2) Internal resources, that determine which signal is being sent over the connection and how the received signals are processed by the endpoint.

The resources allocated to a connection, and more generally the handling of the connection, are chosen by the gateway under instructions from the call agent. The call agent will provide these instructions by sending two set of parameters to the gateway:

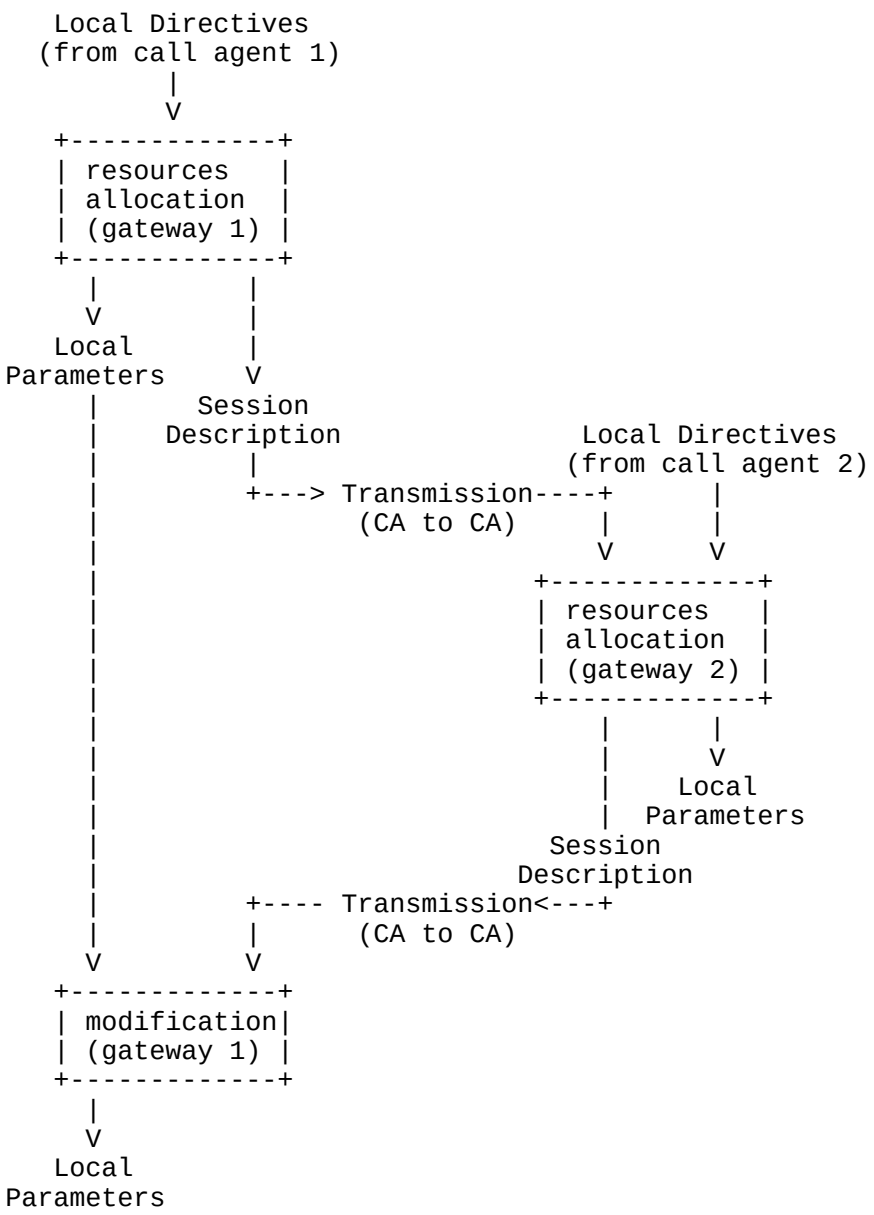
1) The local directives instruct the gateway on the choice of resources that should be used for a connection,

2) When available, the "session description" provided by the other end of the connection.

The local directives specify such parameters as the mode of the connection (e.g. send only, send-receive), preferred coding or packetization methods, usage of echo cancellation or silence suppression. (A detailed list can be found in the specification of the LocalConnectionOptions parameter of the CreateConnection command.) For each of these parameters, the call agent can either specify a value, a range of value, or no value at all. This allow various implementations to implement various level of control, from a very tight control where the call agent specifies minute details of the connection handling to a very loose control where the call agent only specifies broad guidelines, such as the maximum bandwidth, and let the gateway choose the detailed values.

Based on the value of the local directives, the gateway will determine the resources allocated to the connection. When this is possible, the gateway will choose values that are in line with the remote session description - but there is no absolute requirement that the parameters be exactly the same.

Once the resource have been allocated, the gateway will compose a "session description" that describes the way it intends to receive packets. Note that the session description may in some cases present a range of values. For example, if the gateway is ready to accept one of several compression algorithm, it can provide a list of these accepted algorithms.



-- Information flow: local directives & session descriptions --

2.1.3.4. Special case of local connections

Large gateways include a large number of endpoints which are often of different types. In some networks, we may often have to set-up connections between endpoints that are located within the same gateway. Examples of such connections may be:

* Connecting a trunk line to a wiretap device,

* Connecting a call to an Interactive Voice-Response unit,

* Connecting a call to a Conferencing unit,

* Routing a call from on endpoint to another, something often described as a "hairpin" connection.

Local connections are much simpler to establish than network connections. In most cases, the connection will be established through some local interconnecting device, such as for example a TDM bus.

When two endpoints are managed by the same gateway, it is possible to specify the connection in a single command that conveys the name of the two endpoints that will be connected. The command is essentially a "Create Connection" command which includes the name of the second endpoint in lieu of the "remote session description."

2.1.4. Names of Call Agents and other entities

The media gateway control protocol has been designed to allow the implementation of redundant Call Agents, for enhanced network reliability. This means that there is no fixed binding between entities and hardware platforms or network interfaces.

Reliability can be improved by the following precautions:

* Entities such as endpoints or Call Agents are identified by their domain name, not their network addresses. Several addresses can be associated with a domain name. If a command or a response cannot be forwarded to one of the network addresses, implementations should retry the transmission using another address.

* Entities may move to another platform. The association between a logical name (domain name) and the actual platform are kept in the domain name service. Call Agents and Gateways should keep track of the time-to-live of the record they read from the DNS. They should query the DNS to refresh the information if the time to live has expired.

In addition to the indirection provided by the use of domain names and the DNS, the concept of "notified entity" is central to reliability and fail-over in MGCP. The "notified entity" for an endpoint is the Call Agent currently controlling that endpoint. At any point in time, an endpoint has one, and only one, "notified entity" associated with it, and when the endpoint needs to send a command to the Call Agent, it MUST send the command to the current "notified entity" for which endpoint(s) the command pertains. Upon startup, the "notified entity" MUST be set to a provisioned value. Most commands sent by the Call Agent include the ability to explicitly name the "notified entity" through the use of a "NotifiedEntity" parameter. The "notified entity" will stay the same until either a new "NotifiedEntity" parameter is received or the endpoint reboots. If the "notified entity" for an endpoint is empty or has not been set explicitly, the "notified entity" will then default to the source address of the last connection handling command or notification request received for the endpoint. Auditing will thus not change the "notified entity."

2.1.5. Digit maps

The Call Agent can ask the gateway to collect digits dialed by the user. This facility is intended to be used with residential gateways to collect the numbers that a user dials; it may also be used with trunking gateways and access gateways alike, to collect the access codes, credit card numbers and other numbers requested by call control services.

An alternative procedure is for the gateway to notify the Call Agent of the dialed digits, as soon as they are dialed. However, such a procedure generates a large number of interactions. It is preferable to accumulate the dialed numbers in a buffer, and to transmit them in a single message.

The problem with this accumulation approach, however, is that it is hard for the gateway to predict how many numbers it needs to accumulate before transmission. For example, using the phone on our desk, we can dial the following numbers:

0	Local operator
00	Long distance operator
xxxx	Local extension number
8xxxxxxx	Local number
#xxxxxxx	Shortcut to local number at other corporate sites
*xx	Star services
91xxxxxxxxxx	Long distance number
9011 + up to 15 digits	International number

The solution to this problem is to load the gateway with a digit map that correspond to the dial plan. This digit map is expressed using a syntax derived from the Unix system command, egrep. For example, the dial plan described above results in the following digit map:

```
(0T| 00T|[1-7]xxx|8xxxxxxx|#xxxxxxx|*xx|91xxxxxxxxxx|9011x.T)
```

The formal syntax of the digit map is described by the DigitMap rule in the formal syntax description of the protocol (section 3.4). A Digit-Map, according to this syntax, is defined either by a "string" or by a list of strings. Each string in the list is an alternative numbering scheme, specified either as a set of digits or timers, or as regular expression. A gateway that detects digits, letters or timers will:

1) Add the event parameter code as a token to the end of an internal state variable called the "current dial string"

2) Apply the current dial string to the digit map table, attempting a match to each regular expression in the Digit Map in lexical order

3) If the result is under-qualified (partially matches at least one entry in the digit map), do nothing further.

If the result matches, or is over-qualified (i.e. no further digits could possibly produce a match), send the current digit string to the Call Agent. A match, in this specification, can be either a "perfect match," exactly matching one of the specified alternatives, or an impossible match, which occur when the dial string does not match any of the alternative. Unexpected timers, for example, can cause "impossible matches." Both perfect matches and impossible matches trigger notification of the accumulated digits.

Digit maps are provided to the gateway by the Call Agent, whenever the Call Agent instructs the gateway to listen for digits.

2.1.6. Names of events

The concept of events and signals is central to MGCP. A Call Agent may ask to be notified about certain events occurring in an endpoint, e.g. off-hook events, and a call agent may request certain signals to be applied to an endpoint, e.g. dial-tone.

Events and signals are grouped in packages within which they share the same namespace which we will refer to as event names in the following. Packages are groupings of the events and signals supported by a particular type of endpoint. For instance, one package may support a certain group of events and signals for analog access lines, and another package may support another group of events and signals for video lines. One or more packages may exist for a given endpoint-type.

Event names are case insensitive and are composed of two logical parts, a package name and an event name. Both names are strings of letters, hyphens and digits, with the restriction that hyphens shall never be the first or last characters in a name. Package or event names are not case sensitive - values such as "hu", "Hu", "HU" or "hU" should be considered equal.

Examples of package names are "D" (DTMF), "M" (MF), "T" (Trunk) or "L" (Line). Examples of event names can be "hu" (off hook or "hang-up" transition), "hf" (flash hook) or "0" (the digit zero).

In textual representations, the package name, when present, is separated from the event name by a slash ("/"). The package name is in fact optional. Each endpoint-type has a default package associated with it, and if the package name is excluded from the event name, the default package name for that endpoint-type is assumed. For example, for an analog access line, the following two event names are equal:

l/dl dial-tone in the line package for an analog access line.

dl dial-tone in the line package (default) for an analog access line.

This document defines a basic set of package names and event names. Additional package names and event names can be registered with the IANA. A package definition shall define the name of the package, and the definition of each event belonging to the package. The event definition shall include the precise name of the event (i.e., the code used in MGCP), a plain text definition of the event, and, when appropriate, the precise definition of the corresponding signals, for example the exact frequencies of audio signal such as dial tones or DTMF tones.

In addition, implementers can gain experience by using experimental packages. The names of experimental packages must start with the two characters "x-"; the IANA shall not register package names that start with these characters.

Digits, or letters, are supported in many packages, notably "DTMF" and "MF". Digits and letters are defined by the rules "Digit" and "Letter" in the definition of digit maps. This definition refers to the digits (0 to 9), to the asterisk or star ("*") and orthotrope, number or pound sign ("#"), and to the letters "A", "B", "C" and "D", as well as the timer indication "T". These letters can be combined in "digit string" that represent the keys that a user punched on a dial. In addition, the letter "X" can be used to represent all digits, and the sign "$" can be used in wildcard notations. The need to easily express the digit strings has a consequence on the form of event names:

> An event name that does not denote a digit should always contain at least one character that is neither a digit, nor one of the letters A, B, C, D, T or X. (Such names should not contain the special signs "*", "#", "/" or "$".)

A Call Agent may often have to ask a gateway to detect a group of events. Two conventions can be used to denote such groups:

* The wildcard convention can be used to detect any event belonging to a package, or a given event in many packages, or event any event in any package supported by the gateway.

* The regular expression Range notation can be used to detect a range of digits.

The star sign (*) can be used as a wildcard instead of a package name, and the keyword "all" can be used as a wildcard instead of an event name:

> A name such as "foo/all" denotes all events in package "foo"
> A name such as "*/bar" denotes the event "bar" in any package supported by the gateway
> The names "*" or "*/all" denote all events supported by the gate way.

The call agent can ask a gateway to detect a set of digits or letters either by individually describing those letters, or by using the "range" notation defined in the syntax of digit strings. For example, the call agent can:

Use the letter "x" to denote "any letter or digit."
Use the notation "[0-9#]" to denote the digits 0 to 9 and the pound sign.

In some cases, Call Agents will request the gateway to generate or detect events on connections rather than on the end point itself. For example, gateways may be asked to provide a ringback tone on a connection. When an event shall be applied on a connection, the name of the connection is added to the name of the event, using an "at" sign (@) as a delimiter, as in:

G/rt@0A3F58

The wildcard character "*" (star) can be used to denote "all connections". When this convention is used, the gateway will generate or detect the event on all the connections that are connected to the endpoint. An example of this convention could be:

R/qa@*

The wildcard character "$" can be used to denote "the current connection." It should only be used by the call agent, when the event notification request is "encapsulated" within a command creation or modification command. When this convention is used, the gateway will generate or detect the event on the connection that is currently being created or modified. An example of this convention is:

G/rt@$

The connection id, or a wildcard replacement, can be used in conjunction with the "all packages" and "all events" conventions. For example, the notation:

/all@

can be used to designate all events on all connections.

Events and signals are described in packages. The package description must provide, for each events, the following informations:

* The description of the event and its purpose, which should mean the actual signal that is generated by the client (i.e., xx ms FSK tone) as well as the resulting user observed result (i.e., MW light on/off).

* The detailed characteristics of the event, such as for example frequencies and amplitude of audio signals, modulations and repetitions,

* The typical and maximum duration of the event.

Signals are divided into different types depending on their behavior:

* On/off (OO) Once applied, these signals last forever until they are turned off. This may happen either as the result of an event or a new SignalRequests (see later).

* Time-out (TO) Once applied, these signals last until they are either turned off (by an event or SignalRequests) or a signal specific period of time has elapsed. Depending on package specifications, a signal that times out may generate an "operation complete" event.

* Brief (BR) The duration of these signals is so short, that they stop on their own. If an event occurs the signal will not stop, however if a new SignalRequests is applied, the signal will stop. (Note: this point should be debated. One could make a case that events such as strings of DTMF digits should in fact be allowed to complete.)

  TO signals are normally used to alert the endpoints' users, to signal them that they are expected to perform a specific action, such as hang down the phone (ringing). Transmission of these signals should typically be interrupted as soon as the first of the requested events has been produced.

  Package descriptions should describe, for all signals, their type (OO, TO, BR). They should also describe the maximum duration of the TO signals.

2.2. Usage of SDP

The Call Agent uses the MGCP to provision the gateways with the description of connection parameters such as IP addresses, UDP port and RTP profiles. These descriptions will follow the conventions delineated in the Session Description Protocol which is now an IETF proposed standard, documented in RFC 2327.

SDP allows for description of multimedia conferences. This version limits SDP usage to the setting of audio circuits and data access circuits. The initial session descriptions contain the description of exactly one media, of type "audio" for audio connections, "nas" for data access.

2.3. Gateway Control Commands

This section describes the commands of the MGCP. The service consists of connection handling and endpoint handling commands. There are nine commands in the protocol:

* The Call Agent can issue an EndpointConfiguration command to a gateway, instructing the gateway about the coding characteristics expected by the "line-side" of the endpoint.

* The Call Agent can issue a NotificationRequest command to a gateway, instructing the gateway to watch for specific events such as hook actions or DTMF tones on a specified endpoint .

* The gateway will then use the Notify command to inform the Call Agent when the requested events occur.

* The Call Agent can use the CreateConnection command to create a connection that terminates in an "endpoint" inside the gateway.

* The Call Agent can use the ModifyConnection command to change the parameters associated to a previously established connection.

* The Call Agent can use the DeleteConnection command to delete an existing connection. The DeleteConnection command may also be used by a gateway to indicate that a connection can no longer be sustained.

* The Call Agent can use the AuditEndpoint and AuditConnection commands to audit the status of an "endpoint" and any connections associated with it. Network management beyond the capabilities provided by these commands are generally desirable, e.g. information about the status of the gateway. Such capabilities are expected to be supported by the use of the Simple Network Management Protocol (SNMP) and definition of a MIB which is outside the scope of this specification.

* The Gateway can use the RestartInProgress command to notify the Call Agent that the gateway, or a group of endpoints managed by the gateway, is being taken out of service or is being placed back in service.

These services allow a controller (normally, the Call Agent) to instruct a gateway on the creation of connections that terminate in an "endpoint" attached to the gateway, and to be informed about events occurring at the endpoint. An endpoint may be for example:

* A specific trunk circuit, within a trunk group terminating in a gateway,

* A specific announcement handled by an announcement server.

Connections are grouped into "calls". Several connections, that may or may not belong to the same call, can terminate in the same endpoint . Each connection is qualified by a "mode" parameter, which can be set to "send only" (sendonly), "receive only" (recvonly), "send/receive" (sendrecv), "conference" (confrnce), "data", "inactive" (inactive), "loopback", "continuity test" (conttest), "network loop back" (netwloop) or "network continuity test" (netwtest).

The handling of the audio signals received on these connections is determined by the mode parameters:

* Audio signals received in data packets through connections in "receive", "conference" or "send/receive" mode are mixed and sent to the endpoint.

* Audio signals originating from the endpoint are transmitted over all the connections whose mode is "send", "conference" or "send/receive."

* In addition to being sent to the endpoint, audio signals received in data packets through connections in "conference" mode are replicated to all the other connections whose mode is "conference."

The "loopback" and "continuity test" modes are used during maintenance and continuity test operations. There are two flavors of continuity test, one specified by ITU and one used in the US. In the first case, the test is a loopback test. The originating switch will send a tone (the go tone) on the bearer circuit and expect the terminating switch to loopback the circuit. If the originating switch sees the same tone returned (the return tone), the COT has passed. If not, the COT has failed. In the second case, the go and return tones are different. The originating switch sends a certain go tone. The terminating switch detects the go tone, it asserts a different return tone in the backwards direction. When the originating switch detects the return tone, the COT is passed. If the originating switch never detects the return tone, the COT has failed.

If the mode is set to "loopback", the gateway is expected to return the incoming signal from the endpoint back into that same endpoint. This procedure will be used, typically, for testing the continuity of trunk circuits according to the ITU specifications.

If the mode is set to "continuity test", the gateway is informed that the other end of the circuit has initiated a continuity test procedure according to the GR specification. The gateway will place the circuit in the transponder mode required for dual-tone continuity tests.

If the mode is set to "network loopback", the audio signals received from the connection will be echoed back on the same connection.

If the mode is set to "network continuity test", the gateway will process the packets received from the connection according to the transponder mode required for dual-tone continuity test, and send the processed signal back on the connection.

2.3.1. EndpointConfiguration

The EndpointConfiguration commands are used to specify the encoding of the signals that will be received by the endpoint. For example, in certain international telephony configurations, some calls will carry mu-law encoded audio signals, while other will use A-law. The Call Agent will use the EndpointConfiguration command to pass this information to the gateway. The configuration may vary on a call by call basis, but can also be used in the absence of any connection.

```
        ReturnCode
        <-- EndpointConfiguration( EndpointId,
                                   BearerInformation)
```

EndpointId is the name for the endpoint in the gateway where EndpointConfiguration executes, as defined in section 2.1.1. The "any of" wildcard convention shall not be used. If the "all of" wildcard convention is used, the command applies to all the endpoint whose name matches the wildcard.

BearerInformation is a parameter defining the coding of the data received from the line side. These information is encoded as a list of sub-parameters. The only sub-parameter defined in this version of the specification is the encoding method, whose values can be set to "A-law" and "mu-law".

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

2.3.2. NotificationRequest

The NotificationRequest commands are used to request the gateway to send notifications upon the occurrence of specified events in an endpoint. For example, a notification may be requested for when a gateway detects that an endpoint is receiving tones associated with fax communication. The entity receiving this notification may decide to use a different type of encoding method in the connections bound to this endpoint.

```
    ReturnCode
    <-- NotificationRequest( EndpointId,
                             [NotifiedEntity,]
                             [RequestedEvents,]
                             RequestIdentifier,
                             [DigitMap,]
                             [SignalRequests,]
                             [QuarantineHandling,]
                             [DetectEvents,]
                             [encapsulated EndpointConfiguration])
```

EndpointId is the name for the endpoint in the gateway where NotificationRequest executes, as defined in section 2.1.1.

NotifiedEntity is an optional parameter that specifies where the notifications should be sent. When this parameter is absent, the notifications should be sent to the originator of the NotificationRequest.

RequestIdentifier is used to correlate this request with the notifications that it triggers.

RequestedEvents is a list of events that the gateway is requested to detect and report. Such events include, for example, fax tones, continuity tones, or on-hook transition. To each event is associated an action, which can be:

* Notify the event immediately, together with the accumulated list of observed events,

* Swap audio,

* Accumulate the event in an event buffer, but don’t notify yet,

* Accumulate according to Digit Map,

* Keep Signal(s) active,

* process the Embedded Notification Request,

* Ignore the event.

Some actions can be combined. In particular:

* The "swap audio" action can be combined with "Notify", "Accumulate" and "Ignore."

* The "keep signal active" action can be combined with "Notify", "Accumulate", "Accumulate according to Digit Map", "Ignore" and "Embedded Notification Request."

* The "Embedded Notification Request" can be combined with "Accumulate" and with "Keep signals active." It can also be combined with Notify, if the gateway is allowed to issue several Notify commands in response to a single Notification request.

In addition to the requestedEvents parameter specified in the command, some profiles of MGCP have introduced the concept of "persistent events." According to such profiles, the persistent event list is configured in the endpoint, by means outside the scope of MGCP. The basic MGCP specification does not specify any persistent event.

If a persistent event is not included in the list of RequestedEvents, and the event occurs, the event will be detected anyway, and processed like all other events, as if the persistent event had been requested with a Notify action. Thus, informally, persistent events can be viewed as always being implicitly included in the list of RequestedEvents with an action to Notify, although no glare detection, etc., will be performed.

Non-persistent events are those events explicitly included in the RequestedEvents list. The (possibly empty) list of requested events completely replaces the previous list of requested events. In addition to the persistent events, only the events specified in the requested events list will be detected by the endpoint. If a persistent event is included in the RequestedEvents list, the action specified will then replace the default action associated with the event for the life of the RequestedEvents list, after which the default action is restored. For example, if "Ignore off-hook" was specified, and a new request without any off-hook instructions were received, the default "Notify off-hook" operation then would be restored. A given event MUST NOT appear more than once in a RequestedEvents.

The gateway will detect the union of the persistent events and the requested events. If an event is not specified in either list, it will be ignored.

The Swap Audio action can be used when a gateway handles more than one active connection on an endpoint. This will be the case for three-way calling, call waiting, and possibly other feature scenarios. In order to avoid the round-trip to the Call Agent when just changing which connection is attached to the audio functions of the endpoint, the NotificationRequest can map an event (usually hook flash, but could be some other event) to a local function swap audio, which selects the "next" connection in a round robin fashion. If there is only one connection, this action is effectively a no-op.

If signal(s) are desired to start when an event being looked for occurs, the "Embedded NotificationRequest" action can be used. The embedded NotificationRequest may include a new list of RequestedEvents, SignalRequests and a new digit map as well. The semantics of the embedded NotificationRequest is as if a new NotificationRequest was just received with the same NotifiedEntity, and RequestIdentifier. When the "Embedded NotificationRequest" is activated, the "current dial string" will be cleared; the list of observed events and the quarantine buffer will be unaffected.

MGCP implementations shall be able to support at least one level of embedding. An embedded NotificationRequest that respects this limitation shall not contain another Embedded NotificationRequest.

DigitMap is an optional parameter that allows the Call Agent to provision the gateways with a digit map according to which digits will be accumulated. If this optional parameter is absent, the previously defined value is retained. This parameter must be defined, either explicitly or through a previous command, if the RequestedEvent parameters contain an request to "accumulate according to the digit map." The collection of these digits will result in a digit string. The digit string is initialized to a null string upon reception of the NotificationRequest, so that a subsequent notification only returns the digits that were collected after this request. Digits that were accumulated according to the digit map are reported as any other accumulated event, in the order in which they occur. It is therefore possible that other events be accumulated may be found in between the list of digits.

SignalRequests is a parameter that contains the set of signals that the gateway is asked to apply to the endpoint, such as, for example ringing, or continuity tones. Signals are identified by their name, which is an event name, and may be qualified by parameters.

The action triggered by the SignalRequests is synchronized with the collection of events specified in the RequestedEvents parameter. For example, if the NotificationRequest mandates "ringing" and the event request ask to look for an "off-hook" event, the ringing shall stop as soon as the gateway detect an off hook event. The formal definition is that the generation of all "Time Out" signals shall stop as soon as one of the requested events is detected, unless the "Keep signals active" action is associated to the specified event.

The specific definition of actions that are requested via these SignalRequests, such as the duration of and frequency of a DTMF digit, is out side the scope of MGCP. This definition may vary from location to location and hence from gateway to gateway.

The RequestedEvents and SignalRequests refer to the same event definitions. In one case, the gateway is asked to detect the occurrence of the event, and in the other case it is asked to generate it. The specific events and signals that a given endpoint can detect or perform are determined by the list of event packages that are supported by that end point. Each package specifies a list of events and actions that can be detected or performed. A gateway that is requested to detect or perform an event belonging to a package that is not supported by the specified endpoint shall return an error. When the event name is not qualified by a package name, the default package name for the end point is assumed. If the event name is not registered in this default package, the gateway shall return an error.

The Call Agent can send a NotificationRequest whose requested signal list is empty. It will do so for example when tone generation should stop.

The optional QuarantineHandling parameter specifies the handling of "quarantine" events, i.e. events that have been detected by the gateway before the arrival of this NotificationRequest command, but have not yet been notified to the Call Agent. The parameter provides a set of handling options:

* whether the quarantined events should be processed or discarded (the default is to process them.)

* whether the gateway is expected to generate at most one notification (step by step), or multiple notifications (loop), in response to this request (the default is exactly one.)

When the parameter is absent, the default value is assumed.

We should note that the quarantine-handling parameter also governs the handling of events that were detected but not yet notified when the command is received.

DetectEvents is an optional parameter that specifies a list of events that the gateway is requested to detect during the quarantine period. When this parameter is absent, the events that should be detected in the quarantine period are those listed in the last received DetectEvents list. In addition, the gateway should also detect the events specified in the request list, including those for which the "ignore" action is specified.

Some events and signals, such as the in-line ringback or the quality alert, are performed or detected on connections terminating in the end point rather than on the endpoint itself. The structure of the event names allow the Call Agent to specify the connection (or connections) on which the events should be performed or detected.

The command may carry an encapsulated EndpointConfiguration command, that will apply to the same endpoint. When this command is present, the parameters of the EndpointConfiguration command are inserted after the normal parameters of the NotificationRequest, with the exception of the EndpointId, which is not replicated.

The encapsulated EndpointConfiguration command shares the fate of the NotificationRequest command. If the NotificationRequest is rejected, the EndpointConfiguration is not executed.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary. .NH 3 Notifications

Notifications are sent via the Notify command and are sent by the gateway when the observed events occur.

```
            ReturnCode
            <-- Notify( EndpointId,
                        [NotifiedEntity,]
                        RequestIdentifier,
                        ObservedEvents)
```

EndpointId is the name for the endpoint in the gateway which is issuing the Notify command, as defined in section 2.1.1. The identifier should be a fully qualified endpoint identifier, including the domain name of the gateway. The local part of the name shall not use the wildcard convention.

NotifiedEntity is an optional parameter that identifies the entity to which the notifications is sent. This parameter is equal to the last received value of the NotifiedEntity parameter. The parameter is absent if there was no such parameter in the triggering request. The notification is sent to the "current notified entity" or, if no such entity was ever specified, to the address from which the request was received.

RequestIdentifier is parameter that repeats the RequestIdentifier parameter of the NotificationRequest that triggered this notification. It is used to correlate this notification with the request that triggered it.

ObservedEvents is a list of events that the gateway detected. A single notification may report a list of events that will be reported in the order in which they were detected. The list may only contain the identification of events that were requested in the RequestedEvents parameter of the triggering NotificationRequest. It will contain the events that were either accumulated (but not notified) or treated according to digit map (but no match yet), and the final event that triggered the detection or provided a final match in the digit map.

ReturnCode is a parameter returned by the call agent. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

2.3.3. CreateConnection

This command is used to create a connection between two endpoints.

```
          ReturnCode,
          ConnectionId,
          [SpecificEndPointId,]
          [LocalConnectionDescriptor,]
          [SecondEndPointId,]
          [SecondConnectionId]
          <--- CreateConnection(CallId,
                                EndpointId,
                                [NotifiedEntity,]
                                [LocalConnectionOptions,]
                                Mode,
                                [{RemoteConnectionDescriptor |
                                  SecondEndpointId}, ]
                                [Encapsulated NotificationRequest,]
                                [Encapsulated EndpointConfiguration])
```

A connection is defined by its endpoints. The input parameters in CreateConnection provide the data necessary to build a gateway's "view" of a connection.

CallId is a globally unique parameter that identifies the call (or session) to which this connection belongs. Connections that belong to the same call share the same call-id. The call-id can be used to identify calls for reporting and accounting purposes. It does not affect the handling of connections by the gateway.

EndpointId is the identifier for the connection endpoint in the gateway where CreateConnection executes. The EndpointId can be fully-specified by assigning a value to the parameter EndpointId in the function call or it may be under-specified by using the "anyone" wildcard convention. If the endpoint is underspecified, the endpoint identifier will be assigned by the gateway and its complete value returned in the SpecificEndPointId parameter of the response.

The NotifiedEntity is an optional parameter that specifies where the Notify or DeleteConnection commands should be sent. If the parameter is absent, the Notify or DeleteConnection commands should be sent to the last received Notified Entity, or to originator of the CreateConnection command if no Notified Entity was ever received for the end point.

LocalConnectionOptions is a parameter used by the Call Agent to direct the handling of the connection by the gateway. The fields contained in LocalConnectionOptions are the following:

* Encoding Method,

* Packetization period,

* Bandwidth,

* Type of Service,

* Usage of echo cancellation,

* Usage of silence suppression or voice activity detection,

* Usage of signal level adaptation and noise level reduction, or "gain control."

* Usage of reservation service,

* Usage of RTP security,

* Type of network used to carry the connection.

This set of field can be completed by vendor specific optional or mandatory extensions. The encoding of the first three fields, when they are present, will be compatible with the SDP and RTP profiles:

* The encoding method shall be specified by using one or several valid encoding names, as defined in the RTP AV Profile or registered with the IANA.

* The packetization period is encoded as either the length of time in milliseconds represented by the media in a packet, as specified in the "ptime" parameter of SDP, or as a range value, specifying both the minimum and maximum acceptable packetization periods.

* The bandwidth is encoded as either a single value or a range, expressed as an integer number of kilobit per seconds.

For each of the first three fields, the Call Agent has three options:

* It may state exactly one value, which the gateway will then use for the connection,

* It may provide a loose specification, such as a list of allowed encoding methods or a range of packetization periods,

* It may simply provide a bandwidth indication, leaving the choice of encoding method and packetization period to the gateway.

The bandwidth specification shall not contradict the specification of encoding methods and packetization period. If an encoding method is specified, then the gateway is authorized to use it, even if it results in the usage of a larger bandwidth than specified.

The LocalConnectionOptions parameter may be absent in the case of a data call.

The Type of Service specifies the class of service that will be used for the connection. When the connection is transmitted over an IP network, the parameters encodes the 8-bit type of service value parameter of the IP header. When the Type of Service is not specified, the gateway shall use a default or configured value.

The gateways can be instructed to perform a reservation, for example using RSVP, on a given connection. When a reservation is needed, the call agent will specify the reservation profile that should be used, which is either "controlled load" or "guaranteed service." The

absence of reservation can be indicated by asking for the "best effort" service, which is the default value of this parameter. When reservation has been asked on a connection, the gateway will:

* start emitting RSVP "PATH" messages if the connection is in "send-only", "send-receive", "conference", "network loop back" or "network continuity test" mode (if a remote connection descriptor has been received,)

* start emitting RSVP "RESV" messages as soon as it receives "PATH" messages if the connection is in "receive-only", "send-receive", "conference", "network loop back" or "network continuity test" mode.

The RSVP filters will be deduced from the characteristics of the connection. The RSVP resource profiles will be deduced from the connection's bandwidth and packetization period.

By default, the telephony gateways always perform echo cancellation. However, it is necessary, for some calls, to turn off these operations. The echo cancellation parameter can have two values, "on" (when the echo cancellation is requested) and "off" (when it is turned off.)

The telephony gateways may perform gain control, in order to adapt the level of the signal. However, it is necessary, for example for modem calls, to turn off this function. The gain control parameter may either be specified as "automatic", or as an explicit number of decibels of gain. The default is to not perform gain control, which is equivalent to specifying a gain of 0 decibels.

The telephony gateways may perform voice activity detection, and avoid sending packets during periods of silence. However, it is necessary, for example for modem calls, to turn off this detection. The silence suppression parameter can have two values, "on" (when the detection is requested) and "off" (when it is turned off.) The default is "off."

The Call agent can request the gateway to enable encryption of the audio Packets. It does so by providing an key specification, as specified in RFC 2327. By default, encryption is not used.

The Call Agent may instruct the gateway to prepare the connection on a specified type of network. The type of network is encoded as in the "connection-field" parameter of the SDP standard. Possible values are IN (Internet), ATM and LOCAL. The parameter is optional; if absent, the network is determined by the type of gateway.

RemoteConnectionDescriptor is the connection descriptor for the remote side of a connection, on the other side of the IP network. It includes the same fields as in the LocalConnectionDescriptor, i.e. the fields that describe a session according to the SDP standard. This parameter may have a null value when the information for the remote end is not known yet. This occurs because the entity that builds a connection starts by sending a CreateConnection to one of the two gateways involved in it. For the first CreateConnection issued, there is no information available about the other side of the connection. This information may be provided later via a ModifyConnection call. In the case of data connections (mode=data), this parameter describes the characteristics of the data connection.

The SecondEndpointId can be used instead of the RemoteConnectionDescriptor to establish a connection between two endpoints located on the same gateway. The connection is by definition a local connection. The SecondEndpointId can be fully-specified by assigning a value to the parameter SecondEndpointId in the function call or it may be under-specified by using the "anyone" wildcard convention. If the secondendpoint is underspecified, the second endpoint identifier will be assigned by the gateway and its complete value returned in the SecondEndPointId parameter of the response.

Mode indicates the mode of operation for this side of the connection. The mode are "send", "receive", "send/receive", "conference", "data", "inactive", "loopback", "continuity test", "network loop back" or "network continuity test." The expected handling of these modes is specified in the introduction of the "Gateway Handling Function" section. Some end points may not be capable of supporting all modes. If the command specifies a mode that the endpoint cannot support, and error shall be returned.

The gateway returns a ConnectionId, that uniquely identifies the connection within one endpoint, and a LocalConnectionDescriptor, which is a session description that contains information about addresses and RTP ports, as defined in SDP. The LocalConnectionDescriptor is not returned in the case of data connections. The SpecificEndPointId is an optional parameter that identifies the responding endpoint. It can be used when the EndpointId argument referred to a "any of" wildcard name. When a SpecificEndPointId is returned, the Call Agent should use it as the EndpointId value is successive commands referring to this call.

When a SecondEndpointId is specified, the command really creates two connections that can be manipulated separately through ModifyConnection and DeleteConnection commands. The response to the creation provides a SecondConnectionId parameter that identifies the second connection.

After receiving a "CreateConnection" request that did not include a RemoteConnectionDescriptor parameter, a gateway is in an ambiguous situation. Because it has exported a LocalConnectionDescriptor parameter, it can potentially receive packets. Because it has not yet received the RemoteConnectionDescriptor parameter of the other gateway, it does not know whether the packets that it receives have been authorized by the Call Agent. It must thus navigate between two risks, i.e. clipping some important announcements or listening to insane data. The behavior of the gateway is determined by the value of the Mode parameter:

* If the mode was set to ReceiveOnly, the gateway should accept the voice signals and transmit them through the endpoint.

* If the mode was set to Inactive, Loopback, Continuity Test, the gateway should refuse the voice signals.

* If the mode was set to Network Loopback or Network Continuity Test, the gateway should perform the expected echo or Response.

Note that the mode values SendReceive, Conference, Data and SendOnly don’t make sense in this situation. They should be treated as errors, and the command should be rejected (Error code 517).

The command may optionally contain an encapsulated Notification Request command, in which case a RequestIdentifier parameter will be present, as well as, optionally, the RequestedEvents DigitMap, SignalRequests, QuarantineHandling and DetectEvents parameters. The encapsulated NotificationRequest is executed simultaneously with the creation of the connection. For example, when the Call Agent wants to initiate a call to an residential gateway, it should:

* ask the residential gateway to prepare a connection, in order to be sure that the user can start speaking as soon as the phone goes off hook,

* ask the residential gateway to start ringing,

* ask the residential gateway to notify the Call Agent when the phone goes off-hook.

This can be accomplished in a single CreateConnection command, by also transmitting the RequestedEvent parameters for the off hook event, and the SignalRequest parameter for the ringing signal.

When these parameters are present, the creation and the NotificationRequests should be synchronized, which means that bothshould be accepted, or both refused. In our example, the CreateConnection may be refused if the gateway does not have sufficient resources, or cannot get adequate resources from the local network access, and the off-hook Notification-Request can be refused in the glare condition, if the user is already off-hook. In this example, the phone should not ring if the connection cannot be established, and the connection should not be established if the user is already off hook.

The NotifiedEntity parameter, if present, applies to both the CreateConnection and the NotificationRequest command. It defines the new "notified entity" for the endpoint.

The command may carry an encapsulated EndpointConfiguration command, that will apply to the same endpoint. When this command is present, the parameters of the EndpointConfiguration command are inserted after the normal parameters of the CreateConnection with the exception of the EndpointId, which is not replicated. The EndpointConfiguration command may be encapsulated together with an encapsulated NotificationRequest command.

The encapsulated EndpointConfiguration command shares the fate of the CreateConnection command. If the CreateConnection is rejected, the EndpointConfiguration is not executed.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

2.3.4. ModifyConnection

This command is used to modify the characteristics of a gateway’s "view" of a connection. This "view" of the call includes both the local connection descriptors as well as the remote connection descriptor.

```
      ReturnCode,
      [LocalConnectionDescriptor]
       <--- ModifyConnection(CallId,
                             EndpointId,
                             ConnectionId,
                             [NotifiedEntity,]
                             [LocalConnectionOptions,]
                             [Mode,]
                             [RemoteConnectionDescriptor,]
                             [Encapsulated NotificationRequest,]
                             [Encapsulated EndpointConfiguration])
```

The parameters used are the same as in the CreateConnection command, with the addition of a ConnectionId that identifies the connection within the endpoint. This parameter is returned by the CreateConnection function, as part of the local connection descriptor. It uniquely identifies the connection within the context of the endpoint.

The EndpointId should be a fully qualified endpoint identifier. The local name shall not use the wildcard convention.

The ModifyConnection command can be used to affect parameters of a connection in the following ways:

* Provide information about the other end of the connection, through the RemoteConnectionDescriptor.

* Activate or deactivate the connection, by changing the value of the Mode parameter. This can occur at any time during the connection, with arbitrary parameter values.

* Change the sending parameters of the connection, for example by switching to a different coding scheme, changing the packetization period, or modifying the handling of echo cancellation.

Connections can only be activated if the RemoteConnectionDescriptor has been provided to the gateway. The receive only mode, however, can be activated without the provision of this descriptor.

The command will only return a LocalConnectionDescriptor if the local connection parameters, such as RTP ports, were modified. (Usage of this feature is actually for further study.)

The command may optionally contain an encapsulated Notification Request command, in which case a RequestIdentifier parameter will be present, as well as, optionnally, the RequestedEvents DigitMap, SignalRequests, QuarantineHandling and DetectEvents parameters. The

encapsulated NotificationRequest is executed simultaneously with the modification of the connection. For example, when a connection is accepted, the calling gateway should be instructed to place the circuit in send-receive mode and to stop providing ringing tones.

This can be accomplished in a single ModifyConnection command, by also transmitting the RequestedEvent parameters, for the on hook event, and an empty SignalRequest parameter, to stop the provision of ringing tones.

When these parameters are present, the modification and the NotificationRequests should be synchronized, which means that both should be accepted, or both refused. The NotifiedEntity parameter, if present, applies to both the ModifyConnection and the NotificationRequest command.

The command may carry an encapsulated EndpointConfiguration command, that will apply to the same endpoint. When this command is present, the parameters of the EndpointConfiguration command are inserted after the normal parameters of the ModifyConnection with the exception of the EndpointId, which is not replicated. The EndpointConfiguration command may be encapsulated together with an encapsulated NotificationRequest command.

The encapsulated EndpointConfiguration command shares the fate of the ModifyConnection command. If the ModifyConnection is rejected, the EndpointConfiguration is not executed.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

2.3.5. DeleteConnection (from the Call Agent)

This command is used to terminate a connection. As a side effect, it collects statistics on the execution of the connection.

```
     ReturnCode,
     Connection-parameters
     <-- DeleteConnection(CallId,
                          EndpointId,
                          ConnectionId,
                          [Encapsulated NotificationRequest,]
                          [Encapsulated EndpointConfiguration])
```

The endpoint identifier, in this form of the DeleteConnection command, shall be fully qualified. Wildcard conventions shall not be used.

In the general case where a connection has two ends, this command has to be sent to both gateways involved in the connection. Some connections, however, may use IP multicast. In this case, they can be deleted individually.

After the connection has been deleted, any loopback that has been requested for the connection should be cancelled. When all connections to an endpoint have been deleted, that endpoint should be placed in inactive mode.

In response to the DeleteConnection command, the gateway returns a list of parameters that describe the status of the connection. These parameters are:

Number of packets sent:

The total number of RTP data packets transmitted by the sender since starting transmission on this connection. The count is not reset if the sender changes its synchronization source identifier (SSRC, as defined in RTP), for example as a result of a Modify command. The value is zero if the connection was set in "receive only" mode.

Number of octets sent:

The total number of payload octets (i.e., not including header or padding) transmitted in RTP data packets by the sender since starting transmission on this connection. The count is not reset if the sender changes its SSRC identifier, for example as a result of a ModifyConnection command. The value is zero if the connection was set in "receive only" mode.

Number of packets received:

The total number of RTP data packets received by the sender since starting reception on this connection. The count includes packets received from different SSRC, if the sender used several values. The value is zero if the connection was set in "send only" mode.

Number of octets received:

The total number of payload octets (i.e., not including header or padding) transmitted in RTP data packets by the sender since starting transmission on this connection. The count includes packets received from different SSRC, if the sender used several values. The value is zero if the connection was set in "send only" mode.

Number of packets lost:

The total number of RTP data packets that have been lost since the beginning of reception. This number is defined to be the number of packets expected less the number of packets actually received, where the number of packets received includes any which are late or duplicates. The count includes packets received from different SSRC, if the sender used several values. Thus packets that arrive late are not counted as lost, and the loss may be negative if there are duplicates. The count includes packets received from different SSRC, if the sender used several values. The number of packets expected is defined to be the extended last sequence number received, as defined next, less the initial sequence number received. The count includes packets received from different SSRC, if the sender used several values. The value is zero if the connection was set in "send only" mode. This parameter is omitted if the connection was set in "data" mode.

Interarrival jitter:

An estimate of the statistical variance of the RTP data packet interarrival time measured in milliseconds and expressed as an unsigned integer. The interarrival jitter J is defined to be the mean deviation (smoothed absolute value) of the difference D in packet spacing at the receiver compared to the sender for a pair of packets. Detailed computation algorithms are found in RFC 1889. The count includes packets received from different SSRC, if the sender used several values. The value is zero if the connection was set in "send only" mode. This parameter is omitted if the connection was set in "data" mode.

Average transmission delay:

An estimate of the network latency, expressed in milliseconds. This is the average value of the difference between the NTP timestamp indicated by the senders of the RTCP messages and the NTP timestamp of the receivers, measured when this messages are received. The average is obtained by summing all the estimates, then dividing by the number of RTCP messages that have been received. This parameter is omitted if the connection was set in "data" mode.
When the gateway’s clock is not synchronized by NTP, the latency value can be computed as one half of the round trip delay, as measured through RTCP.
When the gateway cannot compute the one way delay or the round trip delay, the parameter conveys a null value.

For a detailed definition of these variables, refer to RFC 1889.

When the connection was set up over an ATM network, the meaning of these parameters may change:

Number of packets sent: The total number of ATM cells transmitted since starting transmission on this connection.

Number of octets sent:
The total number of payload octets transmitted in ATM cells.

Number of packets received:
The total number of ATM cells received since starting reception on this connection.

Number of octets received:
The total number of payload octets received in ATM cells.

Number of packets lost:
Should be determined as the number of cell losts, or set to zero if the adaptation layer does not enable the gateway to assess losses.

Interarrival jitter:
Should be understood as the interarrival jitter between ATM cells.

Average transmission delay:
The gateway may not be able to assess this parameter over an ATM network. It could simply report a null value.

When the connection was set up over an LOCAL interconnect, the meaning of these parameters is defined as follows:

Number of packets sent:
Not significant.

Number of octets sent:
The total number of payload octets transmitted over the local connection.

Number of packets received:
Not significant.

Number of octets received:
The total number of payload octets received over the connection.

Number of packets lost:
Not significant. A value of zero is assumed.

Interarrival jitter:
Not significant. A value of zero is assumed.

Average transmission delay:
Not significant. A value of zero is assumed.

The standard set of connection parameters can be extended by the creation of extension parameters.

The command may optionally contain an encapsulated Notification Request command, in which case a RequestIdentifier parameter will be present, as well as, optionnally, the RequestedEvents DigitMap, SignalRequests, QuarantineHandling and DetectEvents parameters. The encapsulated NotificationRequest is executed simultaneously with the deletion of the connection. For example, when a user hang-up is notified, the gateway should be instructed to delete the connection and to start looking for an off hook event.

This can be accomplished in a single DeleteConnection command, by also transmitting the RequestedEvent parameters, for the off hook event, and an empty SignalRequest parameter.

When these parameters are present, the DeleteConnection and the NotificationRequests should be synchronized, which means that both should be accepted, or both refused.

The command may carry an encapsulated EndpointConfiguration command, that will apply to the same endpoint. When this command is present, the parameters of the EndpointConfiguration command are inserted after the normal parameters of the DeleteConnection with the exception of the EndpointId, which is not replicated. The EndpointConfiguration command may be encapsulated together with an encapsulated NotificationRequest command.

The encapsulated EndpointConfiguration command shares the fate of the DeleteConnection command. If the DeleteConnection is rejected, the EndpointConfiguration is not executed.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

2.3.6. DeleteConnection (from the VoIP gateway)

In some circumstances, a gateway may have to clear a connection, for example because it has lost the resource associated with the connection, or because it has detected that the endpoint no longer is capable or willing to send or receive voice. The gateway terminates the connection by using a variant of the DeleteConnection command:

```
ReturnCode,
<-- DeleteConnection( CallId,
                      EndpointId,
                      ConnectionId,
                      Reason-code,
                      Connection-parameters)
```

In addition to the call, endpoint and connection identifiers, the gateway will also send the call's parameters that would have been returned to the Call Agent in response to a DeleteConnection command. The reason code indicates the cause of the disconnection.

ReturnCode is a parameter returned by the call agent. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

2.3.7. DeleteConnection (multiple connections, from the Call Agent)

A variation of the DeleteConnection function can be used by the Call Agent to delete multiple connections at the same time. The command can be used to delete all connections that relate to a Call for an endpoint:

```
ReturnCode,
<-- DeleteConnection( CallId,
                      EndpointId)
```

It can also be used to delete all connections that terminate in a given endpoint:

```
ReturnCode,
<-- DeleteConnection( EndpointId)
```

Finally, Call Agents can take advantage of the hierarchical naming structure of endoints to delete all the connections that belong to a group of endpoints. In this case, the "local name" component of the EndpointID will be specified using the "all value" wildcarding convention. The "any value" convention shall not be used. For example, if endpoints names are structured as the combination of a physical interface name and a circuit number, as in "X35V3+A4/13",

the Call Agent may replace the circuit number by a wild card character "*", as in "X35V3+A4/*". This "wildcard" command instructs the gateway to delete all the connections that where attached to circuits connected to the physical interface "X35V3+A4".

After the connections have been deleted, the endpoint should be placed in inactive mode. Any loopback that has been requested for the connections should be cancelled.

This command does not return any individual statistics or call parameters.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

2.3.8. Audit Endpoint

The AuditEndPoint command can be used by the Call Agent to find out the status of a given endpoint.

```
            ReturnCode,
              EndPointIdList|{
              [RequestedEvents,]
              [DigitMap,]
              [SignalRequests,]
              [RequestIdentifier,]
              [NotifiedEntity,]
              [ConnectionIdentifiers,]
              [DetectEvents,]
              [ObservedEvents,]
              [EventStates,]
              [BearerInformation,]
              [RestartReason,]
              [RestartDelay,]
              [ReasonCode,]
              [Capabilities]}
                      <--- AuditEndPoint(EndpointId,
                                         [RequestedInfo])
```

The EndpointId identifies the endpoint that is being audited. The "all of" wildcard convention can be used to start auditing of a group of endpoints. If this convention is used, the gateway should return the list of endpoint identifiers that match the wildcard in the EndPointIdList parameter. It shall not return any parameter specific to one of these endpoints.

When a non-wildcard EndpointId is specified, the (possibly empty) RequestedInfo parameter describes the information that is requested for the EndpointId specified. The following endpoint info can be audited with this command:

RequestedEvents, DigitMap, SignalRequests, RequestIdentifier, NotifiedEntity, ConnectionIdentifiers, DetectEvents, ObservedEvents, EventStates, RestartReason, RestartDelay, ReasonCode, and Capabilities.

The response will in turn include information about each of the items for which auditing info was requested:

* RequestedEvents: The current value of RequestedEvents the endpoint is using including the action associated with each event. Persistent events are included in the list.

* DigitMap: the digit map the endpoint is currently using.

* SignalRequests: A list of the; Time-Out signals that are currently active, On/Off signals that are currently "on" for the endpoint (with or without parameter), and any pending Brief signals. Time-Out signals that have timed-out, and currently playing Brief signals are not included.

* RequestIdentifier, the RequestIdentifier for the last Notification Request received by this endpoint (includes NotificationRequest encapsulated in Connection handling primitives). If no notification request has been received, the value zero will be returned.

* QuarantineHandling, the QuarantineHandling for the last NotificationRequest received by this endpoint.

* DetectEvents, the list of events that are currently detected in quarantine mode.

* NotifiedEntity, the current notified entity for the endpoint.

* ConnectionIdentifiers, the list of ConnectionIdentifiers for all connections that currently exist for the specified endpoint.

* ObservedEvents: the current list of observed events for the endpoint.

* EventStates: For events that have auditable states associated with them, the event corresponding to the state the endpoint is in, e.g., off-hook if the endpoint is off-hook. The definition of the individual events will state if the event in question has an auditable state associated with it.

* BearerInformation: the value of the last received BearerInformation parameter for this endpoint.

* RestartReason: the value of the restart reason parameter in the last RestartInProgress command issued by the endpoint, "restart" indicating a fully functional endpoint.

* RestartDelay: the value of the restart delay parameter if a RestartInProgress command was issued by the endpoint at the time of the response, or zero if the command would not include this parameter.

* ReasonCode:the value of the Reason-Code parameter in the last RestartInProgress or DeleteConnection command issued by the gateway for the endpoint, or the special value 000 if the endpoint’s state is nominal.

* The capabilities for the endpoint similar to the LocalConnectionOptions parameter and including event packages and connection modes. If there is a need to specify that some parameters, such as e.g., silence suppression, are only compatible with some

* codecs, then the gateway will return several capability sets:

  Compression Algorithm: a list of supported codecs. The rest of the parameters will apply to all codecs specified in this list.

  Packetization Period: A single value or a range may be specified.

  Bandwidth: A single value or a range corresponding to the range for packetization periods may be specified (assuming no silence suppression).

  Echo Cancellation: Whether echo cancellation is supported or not.

  Silence Suppression: Whether silence suppression is supported or not.

  Type of Service: Whether type of service is supported or not.

Event Packages: A list of event packages supported. The first event package in the list will be the default package.

Modes: A list of supported connection modes.

The Call Agent may then decide to use the AuditConnection command to obtain further information about the connections.

If no info was requested and the EndpointId refers to a valid endpoint, the gateway simply returns a positive acknowledgement.

If no NotifiedEntity has been specified in the last NotificationRequest, the notified entity defaults to the source address of the last NotificationRequest command received for this connection.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

2.3.9. Audit Connection

The AuditConnection command can be used by the Call Agent to retrieve the parameters attached to a connection:

```
          ReturnCode,
          [CallId,]
          [NotifiedEntity,]
          [LocalConnectionOptions,]
          [Mode,]
          [RemoteConnectionDescriptor,]
          [LocalConnectionDescriptor,]
          [ConnectionParameters]
                    <--- AuditConnection(EndpointId,
                                         ConnectionId,
                                         RequestedInfo)
```

The EndpointId parameter specifies the endpoint that handles the connection. The wildcard conventions shall not be used.

The ConnectionId parameter is the identifier of the audited connection, within the context of the specified endpoint.

The (possibly empty) RequestedInfo describes the information that is requested for the ConnectionId within the EndpointId specified. The following connection info can be audited with this command:

CallId, NotifiedEntity, LocalConnectionOptions, Mode, RemoteConnectionDescriptor, LocalConnectionDescriptor, ConnectionParameters

The AuditConnectionResponse will in turn include information about each of the items auditing info was requested for:

* CallId, the CallId for the call the connection belongs to.

* NotifiedEntity, the current notified entity for the Connection.

* LocalConnectionOptions, the LocalConnectionOptions that was supplied for the connection.

* Mode, the current mode of the connection.

* RemoteConnectionDescriptor, the RemoteConnectionDescriptor that was supplied to the gateway for the connection.

* LocalConnectionDescriptor, the LocalConnectionDescriptor the gateway supplied for the connection.

* ConnectionParameters, the current value of the connection parameters for the connection.

If no info was requested and the EndpointId is valid, the gateway simply checks that the connection exists, and if so returns a positive acknowledgement.

If no NotifiedEntity has been specified for the connection, the notified entity defaults to the source address of the last connection handling command received for this connection.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

2.3.10. Restart in progress

The RestartInProgress command is used by the gateway to signal that An endpoint, or a group of endpoint, is taken in or out of service.

```
       ReturnCode,
       [NotifiedEntity]
             <------- RestartInProgress ( EndPointId,
                                          RestartMethod,
                                          [RestartDelay,]
                                          [Reason-code])
```

The EndPointId identifies the endpoint that are taken in or out of service. The "all of" wildcard convention may be used to apply the command to a group of endpoint, such as for example all endpoints that are attached to a specified interface, or even all endpoints that are attached to a given gateway. The "any of" wildcard convention shall not be used.

The RestartMethod parameter specified the type of restart. Three values have been defined:

* A "graceful" restart method indicates that the specified endpoints will Be taken out of service after the specified delay. The established connections are not yet affected, but the Call Agent should refrain to establish new connections, and should try to gracefully tear down the existing connections.

* A "forced" restart method indicates that the specified endpoints are taken abruptely out of service. The established connections, if any, are lost.

* A "restart" method indicates that service will be restored on the endpoints after the specified "restart delay." There are no connections that are currently established on the endpoints.

* A "disconnected" method indicates that the endpoint has become disconnected and is now trying to establish connectivity. The "restart delay" specifies the number of seconds the endpoint has been disconnected. Established connections are not affected.

* A "cancel-graceful" method indicates that a gateway is canceling a previously issued "graceful" restart command.

The optional "restart delay" parameter is expressed as a number of seconds. If the number is absent, the delay value should be considered null. In the case of the "graceful" method, a null delay indicates that the call agent should simply wait for the natural termination of the existing connections, without establishing new connections. The restart delay is always considered null in the case of the "forced" method.

A restart delay of null for the "restart" method indicates that service has already been restored. This typically will occur after gateway startup/reboot.

The optional reason code parameter the cause of the restart.

Gateways SHOULD send a "graceful" or "forced" RestartInProgress message as a courtesy to the Call Agent when they are taken out of service, e.g., by being shutdown, or taken out of service by a network management system, although the Call Agent cannot rely on always receiving such messages. Gateways MUST send a "restart" RestartInProgress message with a null delay to their Call Agent when they are back in service according to the restart procedure specified in Section 4.3.4 - Call Agents can rely on receiving this message. Also, gateways MUST send a "disconnected" RestartInProgress message to their current "notified entity" according to the "disconnected" procedure specified in Section 4.3.5. The "restart delay" parameter MUST NOT be used with the "forced" restart method.

The RestartInProgress message will be sent to the current notified entity for the EndpointId in question. It is expected that a default Call Agent, i.e., notified entity, has been provisioned for each endpoint so, after a reboot, the default Call Agent will be the notified entity for each endpoint. Gateways should take full advantage of wild- carding to minimize the number of RestartInProgress messages generated when multiple endpoints in a gateway restart and the endpoints are managed by the same Call Agent.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number optionally followed by commentary.

A NotifiedEntity may additionally be returned with the response from the Call Agent:

* If the response indicated success (return code 200 - transaction executed), the restart procedure has completed, and the NotifiedEntity returned is the new "notified entity" for the endpoint(s).

* If the response from the Call Agent indicated an error, the restart procedure is not yet complete, and must therefore be initiated again. If a NotifiedEntity parameter was returned, it then specifies the new "notified entity" for the endpoint(s), which must consequently be used when retrying the restart procedure.

2.4. Return codes and error codes.

All MGCP commands are acknowledged. The acknowledgment carries a return code, which indicates the status of the command. The return code is an integer number, for which four ranges of values have been defined:

* values between 100 and 199 indicate a provisional response,

* values between 200 and 299 indicate a successful completion,

* values between 400 and 499 indicate a transient error,

* values between 500 and 599 indicate a permanent error.

The values that have been already defined are listed in the following list:

100 The transaction is currently being executed. An actual completion message will follow on later.

200 The requested transaction was executed normally.

250 The connection was deleted.

400 The transaction could not be executed, due to a transient error.

401 The phone is already off hook

402 The phone is already on hook

403 The transaction could not be executed, because the endpoint does not have sufficient resources at this time

404 Insufficient bandwidth at this time

500 The transaction could not be executed, because the endpoint is unknown.

01 The transaction could not be executed, because the endpoint is not ready.

502 The transaction could not be executed, because the endpoint does not have sufficient resources

510 The transaction could not be executed, because a protocol error was detected.

11 The transaction could not be executed, because the command contained an unrecognized extension.

512 The transaction could not be executed, because the gateway is not equipped to detect one of the requested events.

513 The transaction could not be executed, because the gateway is not equipped to generate one of the requested signals.

514 The transaction could not be executed, because the gateway cannot send the specified announcement.

515 The transaction refers to an incorrect connection-id (may have been already deleted)

516 The transaction refers to an unknown call-id.

517 Unsupported or invalid mode.

518 Unsupported or unknown package.

519 Endpoint does not have a digit map.

520 The transaction could not be executed, because the endpoint is "restarting".

521 Endpoint redirected to another Call Agent.

522 No such event or signal.

523 Unknown action or illegal combination of actions

524 Internal inconsistency in LocalConnectionOptions

525 Unknown extension in LocalConnectionOptions

526 Insufficient bandwidth

527 Missing RemoteConnectionDescriptor

528 Incompatible protocol version

529 Internal hardware failure

530 CAS signaling protocol error.

531 failure of a grouping of trunks (e.g. facility failure).

2.5. Reason Codes

Reason-codes are used by the gateway when deleting a connection to inform the Call Agent about the reason for deleting the connection. They may also be used in a RestartInProgress command, to inform the gateway of the Restart's reason. The reason code is an integer number, and the following values have been defined:

000 Endpoint state is nominal. (This code is used only in response to audit requests.)

900 Endpoint malfunctioning

901 Endpoint taken out of service

902 Loss of lower layer connectivity (e.g., downstream sync)

3. Media Gateway Control Protocol

The MGCP implements the media gateway control interface as a set of transactions. The transactions are composed of a command and a mandatory response. There are eight types of command:

* CreateConnection
* ModifyConnection
* DeleteConnection
* NotificationRequest
* Notify
* AuditEndpoint
* AuditConnection
* RestartInProgress

The first four commands are sent by the Call Agent to a gateway. The Notify command is sent by the gateway to the Call Agent. The gateway may also send a DeleteConnection as defined in 2.3.6. The Call Agent may send either of the Audit commands to the gateway. The Gateway may send a RestartInProgress command to the Call Agent.

3.1. General description

All commands are composed of a Command header, optionally followed by a session description.

All responses are composed of a Response header, optionally followed by a session description.

Headers and session descriptions are encoded as a set of text lines, separated by a carriage return and line feed character (or, optionnally, a single line-feed character). The headers are separated from the session description by an empty line.

MGCP uses a transaction identifier to correlate commands and responses. The transaction identifier is encoded as a component of the command header and repeated as a component of the response header (see section 3.2.1, 3.2.1.2 and 3.3).

3.2. Command Header

The command header is composed of:

* A command line, identifying the requested action or verb, the transaction identifier, the endpoint towards which the action is requested, and the MGCP protocol version,

* A set of parameter lines, composed of a parameter name followed by a parameter value.

Unless otherwise noted or dictated by other referenced standards, each component in the command header is case insensitive. This goes for verbs as well as parameters and values, and all comparisons MUST treat upper and lower case as well as combinations of these as being equal.

3.2.1. Command line

The command line is composed of:

* The name of the requested verb,

* The identification of the transaction,

* The name of the endpoint that should execute the command (in notifications or restarts, the name of the endpoint that is issuing the command),

* The protocol version.

These four items are encoded as strings of printable ASCII characters, separated by white spaces, i.e. the ASCII space (0x20) or tabulation (0x09) characters. It is recommended to use exactly one ASCII space separator.

3.2.1.1. Coding of the requested verb

The verbs that can be requested are encoded as four letter upper or lower case ASCII codes (comparisons should be case insensitive) as defined in the following table:

Verb	Code
EndpointConfiguration	EPCF
CreateConnection	CRCX
ModifyConnection	MDCX
DeleteConnection	DLCX
NotificationRequest	RQNT
Notify	NTFY
AuditEndpoint	AUEP
AuditConnection	AUCX
RestartInProgress	RSIP

The transaction identifier is encoded as a string of up to 9 decimal digits. In the command lines, it immediately follows the coding of the verb.

New verbs may be defined in further versions of the protocol. It may be necessary, for experimentation purposes, to use new verbs before they are sanctioned in a published version of this protocol. Experimental verbs should be identified by a four letter code starting with the letter X, such as for example XPER.

3.2.1.2. Transaction Identifiers

MGCP uses a transaction identifier to correlate commands and responses. A gateway supports two separate transaction identifier name spaces:

- a transaction identifier name space for sending transactions, and
- a transaction identifier name space for receiving transactions.

At a minimum, transaction identifiers for commands sent to a given gateway MUST be unique for the maximum lifetime of the transactions within the collection of Call Agents that control that gateway. Thus,

regardless of the sending Call Agent, gateways can always detect duplicate transactions by simply examining the transaction identifier. The coordination of these transaction identifiers between Call Agents is outside the scope of this specification though.

Transaction identifiers for all commands sent from a given gateway MUST be unique for the maximum lifetime of the transactions regardless of which Call Agent the command is sent to. Thus, a Call Agent can always detect a duplicate transaction from a gateway by the combination of the domain-name of the endpoint and the transaction identifier.

The transaction identifier is encoded as a string of up to nine decimal digits. In the command lines, it immediately follows the coding of the verb.

Transaction identifiers have values between 1 and 999999999. An MGCP entity MUST NOT reuse a transaction identifier more quickly than three minutes after completion of the previous command in which the identifier was used.

3.2.1.3. Coding of the endpoint identifiers and entity names

The endpoint identifiers and entity names are encoded as case insensitive e-mail addresses, as defined in RFC 821. In these addresses, the domain name identifies the system where the endpoint is attached, while the left side identifies a specific endpoint on that system.

Examples of such addresses can be:

hrd4/56@gw23.example.net	Circuit number 56 in interface "hrd4" of the Gateway 23 of the "Example" network
Call-agent@ca.example.net	Call Agent for the "example" network
Busy-signal@ann12.example.net	The "busy signal" virtual endpoint in the announcement server number 12.

The name of notified entities is expressed with the same syntax, with the possible addition of a port number as in:

```
Call-agent@ca.example.net:5234
```

In case the port number is omitted, the default MGCP port (2427) will be used.

3.2.1.4. Coding of the protocol version

The protocol version is coded as the key word MGCP followed by a white space and the version number, and optionally followed by a profile name.. The version number is composed of a major version, coded by a decimal number, a dot, and a minor version number, coded as a decimal number. The version described in this document is version 1.0.

The profile name, if present, is represented by a white-space separated strings of visible (printable) characters extending to the end of the line. Profile names may be defined for user communities who want to apply restrictions or other profiling to MGCP.

In the initial messages, the version will be coded as:

MGCP 1.0

3.2.2. Parameter lines

Parameter lines are composed of a parameter name, which in most cases is composed of a single upper case character, followed by a colon, a white space and the parameter value. The parameter that can be present in commands are defined in the following table:

Parameter name	Code	Parameter value
ResponseAck	K	see description
BearerInformation	B	see description
CallId	C	Hexadecimal string, at most 32 chars.
ConnectionId	I	Hexadecimal string, at most 32 chars.
NotifiedEntity	N	An identifier, in RFC 821 format, composed of an arbitrary string and of the domain name of the requesting entity, possibly completed by a port number, as in: Call-agent@ca.example.net:5234
RequestIdentifier	X	Hexadecimal string, at most 32 chars.
LocalConnectionOptions	L	See description
Connection Mode	M	See description
RequestedEvents	R	See description
SignalRequests	S	See description
DigitMap	D	A text encoding of a digit map
ObservedEvents	O	See description
ConnectionParameters	P	See description
ReasonCode	E	An arbitrary character string
SpecificEndpointID	Z	An identifier, in RFC 821 format, composed of an arbitrary string, followed by an "@" followed by the domain name of the gateway to which this endpoint is attached.
Second Endpoint ID	Z2	Endpoint Id.
SecondConnectionId	I2	Connection Id.
RequestedInfo	F	See description
QuarantineHandling	Q	See description
DetectEvents	T	See Description
RestartMethod	RM	See description
RestartDelay	RD	A number of seconds, encoded as a decimal number
EventStates	ES	See description
Capabilities	A	See description
RemoteConnection Descriptor	RC	Session Description
LocalConnection Descriptor	LC	Session Description

The parameters are not necessarily present in all commands. The following table provides the association between parameters and commands. The letter M stands for mandatory, O for optional and F for forbidden.

Parameter name	EP CF	CR CX	MD CX	DL CX	RQ NT	NT FY	AU EP	AU CX	RS IP
ResponseAck	O	O	O	O	O	O	O	O	O
BearerInformation	M	O	O	O	O	F	F	F	F
CallId	F	M	M	O	F	F	F	F	F
ConnectionId	F	F	M	O	F	F	F	M	F
RequestIdentifier	F	O+	O+	O+	M	M	F	F	F
LocalConnection Options	F	O	O	F	F	F	F	F	F
Connection Mode	F	M	M	F	F	F	F	F	F
RequestedEvents	F	O	O	O	O*	F	F	F	F
SignalRequests	F	O	O	O	O*	F	F	F	F
NotifiedEntity	F	O	O	O	O	O	F	F	F
ReasonCode	F	F	F	O	F	F	F	F	O
ObservedEvents	F	F	F	F	F	M	F	F	F
DigitMap	F	O	O	O	O	F	F	F	F
Connection parameters	F	F	F	O	F	F	F	F	F
Specific Endpoint ID	F	F	F	F	F	F	F	F	F
Second Endpoint ID	F	O	F	F	F	F	F	F	F
RequestedInfo	F	F	F	F	F	F	M	M	F
QuarantineHandling	F	O	O	O	O	F	F	F	F
DetectEvents	F	O	O	O	O	F	F	F	F
EventStates	F	F	F	F	F	F	F	F	F
RestartMethod	F	F	F	F	F	F	F	F	M
RestartDelay	F	F	F	F	F	F	F	F	O
SecondConnectionID	F	F	F	F	F	F	F	F	F
Capabilities	F	F	F	F	F	F	F	F	F
RemoteConnection Descriptor	F	O	O	F	F	F	F	F	F
LocalConnection Descriptor	F	F	F	F	F	F	F	F	F

Note (+) that the RequestIdentifier parameter is optional in connection creation, modification and deletion commands, but that it becomes mandatory if the command contains an encapsulated notification request.

Note (*) that the RequestedEvents and SignalRequests parameters are optional in the NotificationRequest. If these parameters are omitted, the corresponding lists will be considered empty.

If implementers need to experiment with new parameters, for example when developing a new application of MGCP, they should identify these parameters by names that start with the string "X-" or "X+", such as for example:

X-FlowerOfTheDay: Daisy

Parameter names that start with "X+" are critical parameter extensions. An MGCP entity that receives a critical parameter extension that it cannot understand should refuse to execute the command. It should respond with an error code 511 (Unrecognized extension).

Parameter names that start with "X-" are non critical parameter extensions. An MGCP entity that receives a non critical parameter extension that it cannot understand can safely ignore that parameter.

3.2.2.1. Response Acknowledgement

The response acknowledgement attribute is used to managed the "at-most-once" facility described in the "transmission over UDP" section. It contains a comma separated list of "confirmed transaction-id ranges".

Each "confirmed transaction-id ranges" is composed of either one decimal number, when the range includes exactly one transaction, or two decimal numbers separated by a single hyphen, describing the lower and higher transaction identifiers included in the range.

An example of response acknowledgement is:

K: 6234-6255, 6257, 19030-19044

3.2.2.2. Local connection options

The local connection options describe the operational parameters that the Call Agent suggests to the gateway. These parameters are:

* The packetization period in milliseconds, encoded as the keyword "p", followed by a colon and a decimal number. If the Call Agent specifies a range of values, the range will be specified as two decimal numbers separated by an hyphen.

* The preferred type of compression algorithm, encoded as the keyword "a", followed by a colon and a character string. If the Call Agent specifies a list of values, these values will be separated by a semicolon.

* The bandwidth in kilobits per second (1000 bits per second), encoded as the keyword "b", followed by a colon and a decimal number. If the Call Agent specifies a range of values, the range will be specified as two decimal numbers separated by an hyphen.

* The echo cancellation parameter, encoded as the keyword "e", followed by a colon and the value "on" or "off".

* The gain control parameter, encoded as the keyword "gc", followed by a colon a value which can be either the keyword "auto" or a decimal number (positive or negative) representing the number of decibels of gain.

* The silence suppression parameter, encoded as the keyword "s", followed by a colon and the value "on" or "off".

* The type of service parameter, encoded as the keyword "t", followed by a colon and the value encoded as two hexadecimal digits.

* The resource reservation parameter, encoded as the keyword "r", followed by a colon and the value "g" (guaranteed service), "cl" (controlled load) or "be" (best effort).

* The encryption key, encoded as the keyword "k" followed by a colon and a key specification, as defined for the parameter "K" of SDP (RFC 2327).

* The type of network, encoded as the keyword "nt" followed by a colon and the type of network encoded as the keyword "IN", "ATM" or "LOCAL".

Each of the parameters is optional. When several parameters are present, the values are separated by a comma.

Examples of connection descriptors are:

```
L: p:10, a:PCMU
L: p:10, a:G726-32
L: p:10-20, b:64
L: b:32-64, e:off
```

These set of attributes may be extended by extension attributes.

Extension attributes are composed of an attribute name, followed by a semi-colon and by an attribute value. The attribute name should start by the two characters "x+", for a mandatory extensions, or "x-", for a non mandatory extension. If a gateway receives a mandatory extension attribute that it does not recognize, it should reject the command with an error code 525 (Unknown extension in LocalConnectionOptions).

3.2.2.3. Capabilities

Capabilities inform the Call Agent about endpoints' capabilities when audited. The encoding of capabilities is based on the Local Connection Options encoding for the parameters that are common to both. In addition, capabilities can also contain a list of supported packages, and a list of supported modes.

The parameters used are:

* A list of supported codecs. The following parameters will apply to all codecs specified in this list. If there is a need to specify that some parameters, such as e.g. silence suppression, are only compatible with some codecs, then the gateway will return several LocalConnectionOptions parameters, one for each set of codecs.

Packetization Period:
A range may be specified.

Bandwidth:
A range corresponding to the range for packetization periods may be specified (assuming no silence suppression). If absent, the values will be deduced from the codec type.

Echo Cancellation:
"on" if echo cancellation is supported for this codec, "off" otherwise. The default is support.

Silence Suppression:
"on" if silence suppression is supported for this codec, "off" otherwise. The default is support.

Gain Control:
"0" if gain control is not supported. The default is support.

Type of Service:
The value "0" indicates no support for type of service, all other values indicate support for type of service. The default is support.

Resource Reservation:
The parameter indicates the reservation services that are supported, in addition to best effort. The value "g" is encoded when the gateway supports both the guaranteed and the controlled load service, "cl" when only the controlled load service is supported. The default is "best effort."

Encryption Key:
Encoding any value indicates support for encryption. Default is no support.

Type of network:
The keyword "nt", followed by a colon and a semicolon separated list of supported network types. This parameter is optional.

Event Packages
The event packages supported by this endpoint encoded as the keyword "v", followed by a colon and a character string. If a list of values is specified, these values will be separated by a semicolon. The first value specified will be the default package for that endpoint.

Modes
The modes supported by this endpoint encoded as the keyword "m", followed by a colon and a semicolon-separated list of supported connection modes for this endpoint.

3.2.2.4. Connection parameters

Connection parameters are encoded as a string of type and value pairs, where the type is a either letter identifier of the parameter or an extension type, and the value a decimal integer. Types are separated from value by an ‘=’ sign. Parameters are encoded from each other by a comma.

The connection parameter types are specified in the following table:

Connection parameter name	Code	Connection parameter value
Packets sent	PS	The number of packets that were sent on the connection.
Octets sent	OS	The number of octets that were sent on the connection.
Packets received	PR	The number of packets that were received on the connection.
Octets received	OR	The number of octets that were received on the connection.
Packets lost	PL	The number of packets that were not received on the connection, as deduced from gaps in the sequence number.
Jitter	JI	The average inter-packet arrival jitter, in milliseconds, expressed as an integer number.
Latency	LA	Average latency, in milliseconds, expressed as an integer number.

Extension parameters names are composed of the string "X-" followed by a two letters extension parameter name. Call agents that received unrecognized extensions shall silently ignore these extensions.

An example of connection parameter encoding is:

```
P: PS=1245, OS=62345, PR=0, OR=0, PL=0, JI=0, LA=48
```

3.2.2.5. Reason Codes

Reason codes are three-digit numeric values. The reason code is optionally followed by a white space and commentary, e.g.:

```
900 Endpoint malfunctioning
```

A list of reason-codes can be found in Section 2.5.

3.2.2.6. Connection mode

The connection mode describes the mode of operation of the connection. The possible values are:

Mode	Meaning
M: sendonly	The gateway should only send packets
M: recvonly	The gateway should only receive packets
M: sendrecv	The gateway should send and receive packets
M: confrnce	The gateway should place the connection in conference mode
M: inactive	The gateway should neither send nor receive packets
M: loopback	The gateway should place the circuit in loopback mode.
M: conttest	The gateway should place the circuit in test mode.
M: netwloop	The gateway should place the connection in network loopback mode.
M: netwtest	The gateway should place the connection in network continuity test mode.
M: data	The gateway should use the circuit for network access for data (e.g., PPP, SLIP, etc.).

3.2.2.7. Coding of event names

Event names are composed of an optional package name, separated by a slash (/) from the name of the actual event. The event name can optionally be followed by an at sign (@) and the identifier of a connection on which the event should be observed. Event names are used in the RequestedEvents, SignalRequests and ObservedEvents parameter.

Each signal has one of the following signal-types associated with: On/Off (OO), Time-out (TO), Brief (BR). (These signal types are specified in the package definitions, and are not present in the messages.) On/Off signals can be parameterized with a "+" to turn the signal on, or a "-" to turn the signal off. If an on/off signal is not parameterized, the signal is turned on. Both of the following will turn the vmwi signal on:

vmwi(+), vmwi

The following are valid examples of event names:

L/hu	on-hook transition, in the line package
F/0	digit 0 in the MF package
fh	Flash-hook, assuming that the line package is a default package for the end point.
G/rt@0A3F58	Ring back signal on connection "0A3F58".

In addition, the range and wildcard notation of events can be used, instead of individual names, in the RequestedEvents and DetectEvents parameters. The star sign can be used to denote "all connections", and the dollar sign can be used to denote the "current" connection. The following are valid examples of such notations:

M/[0-9]	Digits 0 to 9 in the MF package
fh	Flash-hook, assuming that the line package is a default package for the end point.
[0-9*#A-D]	All digits and letters in the DTMF packages (default for endpoint).
T/$	All events in the trunk packages.
R/qa@*	The quality alert event in all connections
R/rt@$	Ringback on current connection

3.2.2.8. RequestedEvents

The RequestedEvent parameter provides the list of events that have been requested. The event codes are described in the previous section.

Each event can be qualified by a requested action, or by a list of actions. The actions, when specified, are encoded as a list of keywords, enclosed in parenthesis and separated by commas. The codes for the various actions are:

Action	Code
Notify immediately	N
Accumulate	A
Treat according to digit map	D
Swap	S
Ignore	I
Keep Signal(s) active	K
Embedded Notification Request	E

When no action is specified, the default action is to notify the event. This means that, for example, ft and ft(N) are equivalent. Events that are not listed are ignored.

The digit-map action can only be specified for the digits, letters and interdigit timers in the MF and DTMF packages, or in other packages that would define the encoding of digits and timers.

The requested list is encoded on a single line, with event/action groups separated by commas. Examples of RequestedEvents encoding are:

```
R: hu(N), hf(S,N)
R: hu(N), [0-9#T](D)
```

In the case of the "enable" action, the embedded notification request parameters are encoded as a list of up to three parameter groups, separated by commas. Each group start by a one letter identifier, followed by a list of parameters enclosed between parenthesis. The first optional parameter group, identified by the letter "R", is the enabled value of the RequestedEvents parameter. The second optional group, identified by the letter "S", is the enabled value of the SignalRequests parameter. The third optional group, identified by the letter "D", is the enabled value of the DigitMap. (Note that some existing implementation may encode these three components in a different order.)

If the RequestedEvents is not present, the parameter will be set to a null value. If the SignalRequest is not present, the parameter will be set to a null value. If the DigitMap is absent, the current value should be used. The following are valid examples of embedded requests:

```
R: hd(E(R([0-9#T](D),hu(N)),S(dl),D([0-9].[#T])))
R: hd(E(R([0-9#T](D),hu(N)),S(dl)))
```

3.2.2.9. SignalRequests

The SignalRequests parameter provides the name of the signals that have been requested. Each signal is identified by a name, as indicated in the previous section.

Several signals, such as for example announcement or ADSI display, can be qualified by additional parameters:

* the name and parameters of the announcement,

* the string that should be displayed.

These parameters will be encoded as a set of UTF8 character strings, spearated by comams and enclosed within parenthesis, as in:

```
S: adsi("123456 Francois Gerard")
S: ann(no-such-number, 1234567)
```

When several signals are requested, their codes are separated by a comma, as in:

```
S: asdi(123456 Your friend), rg
```

3.2.2.10. ObservedEvent

The observed event parameters provides the list of events that have been observed. The event codes are the same as those used in the NotificationRequest. Events that have been accumulated according to the digit map may be grouped in a single string; they should be reported as lists of isolated events if other events where detected during the digit accumulation. Examples of observed actions are:

```
O: L/hu
O: 8295555T
O: 8,2,9,5,5,L/hf,5,5,T
O: L/hf, L/hf, L/hu
```

3.2.2.11. RequestedInfo

The RequestedInfo parameter contains a comma separated list of parameter codes, as defined in the "Parameter lines" section. For example, if one wants to audit the value of the NotifiedEntity, RequestIdentifier, RequestedEvents, SignalRequests, DigitMap, QuarantineHandling and DetectEvents parameters, The value of the RequestedInfo parameter will be:

```
F:N,X,R,S,D,Q,T
```

The capabilities request, in the AuditEndPoint command, is encoded by the keyword "A", as in:

```
F:A
```

3.2.2.12. QuarantineHandling

The quarantine handling parameter contains a list of comma separated keywords:

* The keyword "process" or "discard" to indicate the treatment of quarantined events. If neither process or discard is present, process is assumed.

* The keyword "step" or "loop" to indicate whether exactly at most one notification is expected, or whether multiple notifications are allowed. If neither step or loop is present, step is assumed. The following values are valid examples:

```
Q:loop
Q:process
Q:discard,loop
```

3.2.2.13. DetectEvents

The DetectEvent parameter is encoded as a comma separated list of events, such as for example:

```
T: hu,hd,hf,[0-9#*]
```

It should be noted, that no actions can be associated with the events.

3.2.2.14. EventStates

The EventStates parameter is encoded as a comma separated list of events, such as for example:

```
ES: hu
```

It should be noted, that no actions can be associated with the events.

3.2.2.15. RestartMethod

The RestartMethod parameter is encoded as one of the keywords "graceful", "forced", "restart", "disconnected" or "cancel-graceful" as for example:

```
RM:restart
```

3.2.2.16. Bearer Information

The values of the bearer informations are encoded as a comma separated list of attributes, represented by an attribute name, separated by a colon from an attribute value.

The only attribute that is defined is the "encoding" (code "e"), whose defined values are "A" (A-law) and "mu" (mu-law).

An example of bearer information encoding is:

```
B: e:mu
```

3.3. Format of response headers

The response header is composed of a response line, optionally followed by headers that encode the response parameters.

An example of response header could be:

```
200 1203 OK
```

The response line starts with the response code, which is a three digit numeric value. The code is followed by a white space, the transaction identifier, and an optional commentary preceded by a white space.

The following table describe the parameters whose presence is mandatory or optional in a response header, as a function of the command that triggered the response. The letter M stands for mandatory, O for optional and F for forbidden.

Parameter name	EP CF	CR CX	MD CX	DL CX	RQ NT	NT FY	AU EP	AU CX	RS IP
ResponseAck	F	F	F	F	F	F	F	F	F
BearerInformation	F	F	F	F	F	F	O	F	F
CallId	F	F	F	F	F	F	F	O	F
ConnectionId	F	O*	F	F	F	F	F	F	F
RequestIdentifier	F	F	F	F	F	F	O	F	F
LocalConnection Options	F	F	F	F	F	F	O	O	F
Connection Mode	F	F	F	F	F	F	F	O	F
RequestedEvents	F	F	F	F	F	F	O	F	F
SignalRequests	F	F	F	F	F	F	O	F	F
NotifiedEntity	F	F	F	F	F	F	F	F	O
ReasonCode	F	F	F	F	F	F	O	F	F
ObservedEvents	F	F	F	F	F	F	O	F	F
DigitMap	F	F	F	F	F	F	O	F	F
Connection Parameters	F	F	F	O	F	F	F	O	F
Specific Endpoint ID	F	O	F	F	F	F	F	F	F
RequestedInfo	F	F	F	F	F	F	F	F	F
QuarantineHandling	F	F	F	F	F	F	O	F	F
DetectEvents	F	F	F	F	F	F	O	F	F
EventStates	F	F	F	F	F	F	O	F	F
RestartMethod	F	F	F	F	F	F	O	F	F
RestartDelay	F	F	F	F	F	F	O	F	F
Capabilities	F	F	F	F	F	F	O	F	F
SecondConnectionId	F	O	F	F	F	F	F	F	F
SecondEndpointID	F	O	F	F	F	F	F	F	F
LocalConnection Descriptor	F	M	O	F	F	F	F	O*	F
RemoteConnection Descriptor	F	F	F	F	F	F	F	O*	F

In the case of a CreateConnection message, the response line is followed by a Connection-Id parameter. It may also be followed a Specific-Endpoint-Id parameter, if the creation request was sent to a wildcarded Endpoint-Id. The connection-Id parameter is marked as optional in the Table. In fact, it is mandatory with all positive responses, when a connection was created, and forbidden when the response is negative, when no connection as created.

In the case of a DeleteConnection message, the response line is followed by a Connection Parameters parameter, as defined in section 3.2.2.2.

A LocalConnectionDescriptor should be transmitted with a positive response (code 200) to a CreateConnection. It may be transmitted in response to a ModifyConnection command, if the modification resulted in a modification of the session parameters. The LocalConnectionDescriptor is encoded as a "session description," as defined in section 3.4. It is separated from the response header by an empty line.

When several session descriptors are encoded in the same response, they are encoded one after each other, separated by an empty line. This is the case for example when the response to an audit connection request carries both a local session description and a remote session description, as in:

```
200 1203 OK
C: A3C47F21456789F0
N: [128.96.41.12]
L: p:10, a:PCMU;G726-32
M: sendrecv
P: PS=1245, OS=62345, PR=780, OR=45123, PL=10, JI=27,LA=48

v=0
c=IN IP4 128.96.41.1
m=audio 1296 RTP/AVP 0

v=0
c=IN IP4 128.96.63.25
m=audio 1296 RTP/AVP 0 96
a=rtpmap:96 G726-32/8000
```

In this example, according to the SDP syntax, each description starts with a "version" line, (v=...). The local description is always transmitted before the remote description. If a connection descriptor is requested, but it does not exist for the connection audited, that connection descriptor will appear with the SDP protocol version field only.

3.4. Formal syntax description of the protocol

In this section, we provided a formal description of the protocol syntax, following the "Augmented BNF for Syntax Specifications" defined in RFC 2234.

```
MGCPMessage = MGCPCommand / MGCPResponse

MGCPCommand = MGCPCommandLine 0*(MGCPParameter) [EOL *SDPinformation]

MGCPCommandLine = MGCPVerb 1*(WSP) <transaction-id> 1*(WSP)
                        <endpointName> 1*(WSP) MGCPversion EOL

MGCPVerb = "EPCF" / "CRCX" / "MDCX" / "DLCX" / "RQNT"
         / "NTFY" / "AUEP" / "AUCX" / "RSIP" / extensionVerb

extensionVerb = "X" 3(ALPHA / DIGIT)

transaction-id = 1*9(DIGIT)

endpointName =  localEndpointName "@" DomainName
LocalEndpointName = LocalNamePart 0*("/" LocalNamePart)
LocalNamePart = AnyName / AllName / NameString
AnyName = "$"
AllNames = "*"
NameString = 1*(range-of-allowed-characters)
DomainName = 1*256(ALPHA / DIGIT / "." / "-") ; as defined in RFC 821

MGCPversion = "MGCP" 1*(WSP) 1*(DIGIT) "." 1*(DIGIT)
              [1*(WSP) ProfileName]
ProfileName = 1*(range-of-allowed-characters)

MGCPParameter = ParameterValue EOL

ParameterValue = ("K" ":" 0*WSP <ResponseAck>) /
                 ("B" ":" 0*WSP <BearerInformation>) /
                 ("C" ":" 0*WSP <CallId>) /
                 ("I" ":" 0*WSP <ConnectionId>) /
                 ("N" ":" 0*WSP <NotifiedEntity>) /
                 ("X" ":" 0*WSP <RequestIdentifier>) /
                 ("L" ":" 0*WSP <LocalConnectionOptions>) /
                 ("M" ":" 0*WSP <ConnectionMode>) /
                 ("R" ":" 0*WSP <RequestedEvents>) /
                 ("S" ":" 0*WSP <SignalRequests>) /
                 ("D" ":" 0*WSP <DigitMap>) /
                 ("O" ":" 0*WSP <ObservedEvents>) /
                 ("P" ":" 0*WSP <ConnectionParameters>) /
                 ("E" ":" 0*WSP <ReasonCode>) /
```

```
                ("Z" ":" 0*WSP <SpecificEndpointID>) /
                ("Z2" ":" 0*WSP <SecondEndpointID>) /
                ("I2" ":" 0*WSP <SecondConnectionID>) /
                ("F" ":" 0*WSP <RequestedInfo>) /
                ("Q" ":" 0*WSP <QuarantineHandling>) /
                ("T" ":" 0*WSP <DetectEvents>) /
                ("RM" ":" 0*WSP <RestartMethod>) /
                ("RD" ":" 0*WSP <RestartDelay>) /
                ("A" ":" 0*WSP <Capabilities>) /
                ("ES" ":" 0*WSP <EventStates>) /
                    (extensionParameter ":" 0*WSP <parameterString>)

ResponseAck =  confirmedTransactionIdRange
               *[ ","  confirmedTransactionIdRange ]

confirmedTransactionIdRange = 1*9DIGIT [ "-" 1*9DIGIT ]

BearerInformation = BearerAttribute 0*("," 0*WSP BearerAttribute)
BearerAttribute = ("e" ":" <BearerEncoding>)
BearerEncoding = "A" / "mu"

CallId = 1*32(HEXDIG)

// The audit request response may include a list of identifiers
ConnectionId = 1*32(HEXDIG) 0*("," 1*32(HEXDIG))
SecondConnectionID = ConnectionId

NotifiedEntity = [LocalName "@"] DomainName [":" portNumber]
LocalName = 1*32(suitableCharacter)
portNumber = 1*5(DIGIT)

RequestIdentifier = 1*32(HEXDIG)

LocalConnectionOptions = [ LocalOptionValue 0*(WSP)
                  0*("," 0*(WSP) LocalOptionValue 0*(WSP)) ]
LocalOptionValue = ("p" ":" <packetizationPeriod> )
                  / ("a" ":" <compressionAlgorithm> )
                  / ("b" ":" <bandwidth> )
                  / ("e" ":" <echoCancellation> )
                  / ("gc" ":" <gainControl> )
                  / ("s" ":" <silenceSuppression> )
                  / ("t" ":" <typeOfService> )
                  / ("r" ":" <resourceReservation> )
                  / ("k" ":" <encryptionmethod>[":"<encryptionKey>])
                  / ("nt" ":" <typeOfNetwork> )
                  / (localOptionExtensionName ":"
                  / localOptionExtensionValue)
```

```
Capabilities = [ CapabilityValue 0*(WSP)
                 0*("," 0*(WSP) CapabilityValue 0*(WSP)) ]


CapabilityValue = LocalOptionValue
                / ("v" ":" <supportedPackages>)
                / ("m" ":" <supportedModes> )


packetizationPeriod = 1*4(DIGIT)["-" 1*4(DIGIT)]
compressionAlgorithm = algorithmName 0*(";" algorithmName)
algorithmName = 1*32(SuitableCharacter)
bandwidth = 1*4(DIGIT)["-" 1*4(DIGIT)]
echoCancellation = "on" / "off"
gainControl = "auto" / ["-"]1*4(DIGIT)
silenceSuppression = "on" / "off"
typeOfService = 2HEXDIG
resourceReservation = "g" / "cl" / "be"

;encryption parameters are coded as in SDP (RFC 2327)
encryptiondata = ( "clear" ":" <encryptionKey> )
               / ( "base64" ":" <encodedEncryptionKey> )
               / ( "uri" ":" <URItoObtainKey> )
               / ( "prompt" ) ; defined in SDP, not usable in MGCP!
encryptionKey = 1*(SuitableCharacter / SP)
encodedEncryptionKey = 1*(ALPHA / DIGIT / "+" / "/" / "=")
URItoObtainKey = 1*(SuitableCharacter) / quotedString

typeOfNetwork = "IN" / "ATM" / "LOCAL"
supportedModes= ConnectionMode 0*(";" ConnectionMode)
supportedPackages = packageName 0*(";" packageName)

localOptionExtensionName = "x" ("+"/"-") 1*32(SuitableCharacter)
localOptionExtensionValue = 1*32(SuitableCharacter) / quotedString


ConnectionMode = "sendonly" / "recvonly" / "sendrecv" /
                 "confrnce" / "inactive" / "loopback" /
                 "conttest" / "netwloop" / "netwtest" / "data"

RequestedEvents = [requestedEvent 0*("," 0*(WSP) requestedEvent)]
requestedEvent = eventName [ "(" requestedActions ")" ]

eventName = [ (packageName / "*") "/" ] (eventId / "all" / eventRange)
            [ "@" (ConnectionId / "$" / "*") ]
packageName = 1*(ALPHA / DIGIT / HYPHEN)
eventId = 1*(SuitableCharacter)
eventRange = "[" 1*(DIGIT / DTMFLetter / "*" / "#" /
```

```
                        (DIGIT "-" DIGIT)/(DTMFLetter "-"
                         DTMFLetter)) "]"

requestedActions = requestedAction 0*("," 0*(WSP) requestedAction)
requestedAction = "N" / "A" / "D" / "S" / "I" / "K" /
                  "E" "(" EmbeddedRequest ")"

EmbeddedRequest =   (      "R" "(" EmbeddedRequestList ")"
                      ["," "S" "(" EmbeddedSignalRequest ")" ]
                      ["," "D" "(" EmbeddedDigitMap ")" ] )
                /   (      "S" "(" EmbeddedSignalRequest ")"
                      ["," "D" "(" EmbeddedDigitMap ")" ] )
                /   (      "D" "(" EmbeddedDigitMap ")" )

EmbeddedRequestList = RequestedEvents
EmbeddedSignalRequest = SignalRequests
EmbeddedDigitMap = DigitMap

SignalRequests = [ SignalRequest 0*("," 0*(WSP) SignalRequest ) ]
SignalRequest = eventName [ "(" eventParameters ")" ]
eventParameters = eventParameter 0*("," 0*(WSP) eventParameter)
eventParameter = eventParameterString / quotedString
eventParameterString = 1*(SuitableCharacter)

DigitMap = DigitString  / "(" DigitStringList ")"
DigitStringList = DigitString 0*( "|" DigitString )
DigitString = 1*(DigitStringElement)
DigitStringElement = DigitPosition ["."]
DigitPosition = DigitMapLetter / DigitMapRange
DigitMapLetter = DIGIT / "#" / "*" / "A" / "B" / "C" / "D" / "T"
DigitMapRange =  "x" / "[" 1*DigitLetter "]"
DigitLetter ::= *((DIGIT "-" DIGIT ) / DigitMapLetter)

ObservedEvents = SignalRequests
EventStates = SignalRequests

ConnectionParameters = [ConnectionParameter
                        0*( "," 0*(WSP) ConnectionParameter )
ConnectionParameter = ( "PS" "=" packetsSent )
                    / ( "OS" "=" octetsSent )
                    / ( "PR" "=" packetsReceived )
                    / ( "OR" "=" octetsReceived )
                    / ( "PL" "=" packetsLost )
                    / ( "JI" "=" jitter )
                    / ( "LA" "=" averageLatency )
                    / ( ConnectionParameterExtensionName "="
                        ConnectionParameterExtensionValue )
packetsSent = 1*9(DIGIT)
```

```
octetsSent = 1*9(DIGIT)
packetsReceived = 1*9(DIGIT)
octetsReceived = 1*9(DIGIT)
packetsLost = 1*9(DIGIT)
jitter = 1*9(DIGIT)
averageLatency = 1*9(DIGIT)
ConnectionParameterExtensionName = "X" "-" 2*ALPHA
ConnectionParameterExtensionValue = 1*9(DIGIT)

ReasonCode = 3DIGIT [SPACE 1*(%x20-7E)]

SpecificEndpointID = endpointName
SecondEndpointID = endpointName

RequestedInfo = [infoCode 0*("," infoCode)]

infoCode = "B" / "C" / "I" / "N" / "X" / "L" / "M" /
           "R" / "S" / "D" / "O" / "P" / "E" / "Z" /
           "Q" / "T" / "RC" / "LC" / "A" / "ES" / "RM" / "RD"

QuarantineHandling = loopControl / processControl /
              (loopControl "," processControl )
loopControl = "step" / "loop"
processControl = "process" / "discard"

DetectEvents = [eventName 0*("," eventName)]

RestartMethod = "graceful" / "forced" / "restart" / "disconnected"

RestartDelay = 1*6(DIGIT)

extensionParameter = "X" ("-"/"+") 1*6(ALPHA / DIGIT)
parameterString = 1*(%x20-7F)

MGCPResponse = MGCPResponseLine 0*(MGCPParameter)
                [EOL *SDPinformation]

MGCPResponseLine = (<responseCode> 1*(WSP) <transaction-id>
                          [1*(WSP) <responseString>] EOL)
responseCode = 3DIGIT
responseString = *(%x20-7E)

SuitableCharacter= DIGIT / ALPHA / "+" / "-" / "_" / "&" /
                   "!" / "'" / "|" / "=" / "#" / "?" / "/" /
                   "." / "$" / "*" / ";" / "@" / "[" / "]" /
                   "^" / "`" / "{" / "}" / "~"

quotedString = DQUOTE visibleString
```

```
                0*(quoteEscape visibleString) DQUOTE
quoteEscape = DQUOTE DQUOTE
visibleString = (%x00-21 / %x23-FF)
EOL = CRLF / LF

SDPinformation = ;See RFC 2327
```

3.5. Encoding of the session description

The session description is encoded in conformance with the session description protocol, SDP. MGCP implementations are expected to be fully capable of parsing any conformant SDP message, and should send session descriptions that strictly conform to the SDP standard. The usage of SDP actually depends on the type of session that is being, as specified in the "mode" parameter:

* if the mode is set to "data", the session description describes the configuration of a data access service.

* if the mode is set to any other value, the session description is for an audio service.

For an audio service, the gateway will consider the information provided in SDP for the "audio" media. For a data service, the gateway will consider the information provided for the "network-access" media.

3.5.1. Usage of SDP for an audio service

In a telephony gateway, we only have to describe sessions that use exactly one media, audio. The parameters of SDP that are relevant for the telephony application are:

At the session description level:

* The IP address of the remote gateway (in commands) or of the local gateway (in responses), or multicast address of the audio conference, encoded as an SDP "connection data" parameter. This parameter specifies the IP address that will be used to exchange RTP packets.

For the audio media:

* Media description field (m) specifying the audio media, the transport port used for receiving RTP packets by the remote gateway (commands) or by the local gateway (responses), the

RTP/AVP transport, and the list of formats that the gateway will accept. This list should normally always include the code 0 (reserved for PCMU).

* Optionally, RTPMAP attributes that define the encoding of dynamic audio formats,

* Optionally, a packetization period (packet time) attribute (Ptime) defining the duration of the packet,

* Optionally, an attribute defining the type of connection (sendonly, recvonly, sendrecv, inactive). Note that this attribute does not have a direct relation with the "Mode" parameter of MGCP. In fact, the SDP type of connection will most of the time be set to "sendrecv", regardless of the value used by MGCP. Other values will only be used rarely, for example in the case of information or announcement servers that need to establish one way connections.

* The IP address of the remote gateway (in commands) or of the local gateway (in responses), if it is not present at the session level.

An example of SDP specification for an audio connection could be:

```
v=0
c=IN IP4 128.96.41.1
m=audio 3456 RTP/AVP 0 96
a=rtpmap:96 G726-32/8000
```

There is a request, in some environments, to use the MGCP to negotiate connections that will use other transmission channels than RTP over UDP and IP. This will be detailed in an extension to this document.

3.5.2. Usage of SDP in a network access service

The parameters of SDP that are relevant for a data network access application are:

For the data media:

* Media description field (m) specifying the network access media, identified by the code "m=nas/xxxx", where "xxxx" describes the access control method that should be used for parametrizing the network access, as specified below. The field may also specify the port that should be used for contacting the server, as specified in the SDP syntax.

* Connection address parameter (c=) specifying the address, or the domain name, of the server that implement the access control method. This parameter may also be specified at the session level.

* Optionally, a bearer type attribute (a=bearer:) describing the type of data connection to be used, including the modem type.

* Optionally, a framing type attribue (a=framing:) describing the type of framing that will be used on the channel.

* Optionally, attributes describing the called number (a=dialed:), the number to which the call was delivered (a=called:) and the calling number (a=dialing:).

* Optionally, attributes describing the range of addresses that could be used by the dialup client on its LAN (a=subnet:).

* Optionally, an encryption key, encoded as specified in the SDP protocol(k=).

The connection address shall be encoded as specified in the SDP standard. It will be used in conjunction with the port specified in the media line to access a server, whose type will one of:

Method name	Method description
radius	Authentication according to the Radius protocol.
tacacs	Authentication according to the TACACS+ protocol.
diameter	Authentication according to the Diameter protocol.
l2tp	Level 2 tunneling protocol. The address and port are those of the LNS.
login	Local login. (There is normally no server for that method.)
none	No authentication required. (The call was probably vetted by the Call Agent.)

If needed, the gateway may use the key specified in the announcement to access the service. That key, in particular, may be used for the establishment of an L2TP tunnel.

The bearer attribute is composed of a bearer name and an optional extension. The bearer type specifies the type of modulation (modem name) or, in the case of digital connections, the type of ISDN service (8 bits, 7 bits). When an extension is present, it is separated from the bearer name by a single slash (/). The valid values of the bearer attribute are defined in the following table:

Type of bearer description	Example of values
ITU modem standard	V.32, V.34, V.90.
ITU modem standard qualified by a manufacturer name	v.90/3com, v.90/rockwell, v.90/xxx
Well known modem types	X2, K56flex
ISDN transparent access, 64 kbps	ISDN64
ISDN64 + V.110	ISDN64/V.110
ISDN64 + V.120	ISDN64/V.120
ISDN transparent access, 56 kbps	ISDN56
Informal identification	(Requires coordination between the Call Agent and the gateway)

The valid values of the framing attribute are defined in the following table:

Type of framing description	Example of values
PPP, asynchronous framing	ppp-asynch
PPP, HDLC framing	ppp-hdlc
SLIP, asynchronous	slip
Asynchronous, no framing	asynch

The network access authentication parameter provides instructions on the access control that should be exercized for the data call. This optional attribute is encoded as:

```
"a=subnet:" <network type> <address type>
   <connection address> "/" <prefix length>
```

Where the parameters "network type", "address type", and "connection address" are formatted as defined for the connection address parameter (c=) in SDP, and where the "prefix length" is a decimal representation of the number of bits in the prefix.

Examples of SDP announcement for the network access service could be:

```
v=0
m=nas/radius
c=IN IP4 radius.example.net
a=bearer:v.34
a=framing:ppp-asynch
a=dialed:18001234567
a=called:12345678901
a=dialing:12340567890

v=0
m=nas/none
c=IN IP4 128.96.41.1
a=subnet:IN IP4 123.45.67.64/26
a=bearer:isdn64
a=framing:ppp-sync
a=dialed:18001234567
a=dialing:2345678901

v=0
c=IN IP4 access.example.net
m=nas/l2tp
k=clear:some-shared-secret
a=bearer:v.32
a=framing:ppp-asynch
a=dialed:18001234567
a=dialing:2345678901
```

3.5.3. Usage of SDP for ATM connections

The specification of the SDP payload for ATM connections will be described in a companion document, "Usage of MGCP to control Voice over ATM gateways." The following text is indicative.

The SDP payload will specify:

* That the connection is to be established over an ATM interface, using the "c=" parameter of SDP to specify an address in the ATM family, the ATM addressing variant (NSAP, UNI, E.164) and the ATM address.

* The "m=audio" parameter will specify the audio encoding and, if needed, the VPI and VCI.

* Additional attributes parameters (a=) will be used to specify the ATM coding variants, such as the type of adaptation layer and the error correction or loss compenmsation algorithms.

An example of SDP payload for an ATM connection could be:

```
v=0 c=ATM NSAP
47.0091.8100.0000.0060.3e64.fd01.0060.3e64.fd01.fe m=audio
5/1002 ATM/AVP PCMU a=connection_type:AAL2
```

3.5.4. Usage of SDP for local connections

When MGCP is used to set up internal connections within a single gateway, the SDP format is used to encode the parameters of that connection. The following parameters will be used:

* The connection parameter (C=) will specify that the connection is local, using the keyword "LOCAL" as network type space, the keyword "EPN" (endpoint name) as address type, and the name of the endpoint as the connection-address.

* The "m=audio" parameter will specify a port number, which will always be set to 0, the type of protocol, always set to the keyword LOCAL, and the type of encoding, using the same conventions used for RTP (RTP payload numbers.) The type of encoding should normally be set to 0 (PCMU).

An example of local SDP payload could be:

```
v=0
c=LOCAL EPN X35V3+A4/13
m=audio 0 LOCAL 0
```

3.6. Transmission over UDP

MGCP messages are transmitted over UDP. Commands are sent to one of the IP addresses defined in the DNS for the specified endpoint . The responses are sent back to the source address of the commands.

When no port is specified for the endpoint, the commands should be sent:

* by the Call Agents, to the default MGCP port for gateways, 2427.

* by the Gateways, to the default MGCP port for Call Agents, 2727.

3.6.1. Providing the At-Most-Once functionality

MGCP messages, being carried over UDP, may be subject to losses. In the absence of a timely response, commands are repeated. Most MGCP commands are not idempotent. The state of the gateway would become

unpredictable if, for example, CreateConnection commands were executed several times. The transmission procedures must thus provide an "At-Most-Once" functionality.

MGCP entities are expected to keep in memory a list of the responses that they sent to recent transactions and a list of the transactions that are currently being executed. The transaction identifiers of incoming commands are compared to the transaction identifiers of the recent responses. If a match is found, the MGCP entity does not execute the transaction, but simply repeats the response. The remaining commands will be compared to the list of current transaction. If a match is found, the MGCP entity does not execute the transaction, which is simply ignored.

The procedure use a long timer value, noted LONG-TIMER in the following. The timer should be set larger than the maximum duration of a transaction, which should take into account the maximum number of repetitions, the maximum value of the repetition timer and the maximum propagation delay of a packet in the network. A suggested value is 30 seconds.

The copy of the responses can be destroyed either LONG-TIMER seconds after the response is issued, or when the gateway (or the call agent) receives a confirmation that the response has been received, through the "Response Acknowledgement attribute". For transactions that are acknowledged through this attribute, the gateway shall keep a copy of the transaction-id for LONG-TIMER seconds after the response is issued, in order to detect and ignore duplicate copies of the transaction request that could be produced by the network.

3.6.2. Transaction identifiers and three ways handshake

Transaction identifiers are integer numbers in the range from 0 to 999,999,999. Call-agents may decide to use a specific number space for each of the gateways that they manage, or to use the same number space for all gateways that belong to some arbitrary group. Call agents may decide to share the load of managing a large gateway between several independent processes. These processes will share the same transaction number space. There are multiple possible implementations of this sharing, such as having a centralized allocation of transaction identifiers, or pre-allocating non-overlapping ranges of identifiers to different processes. The implementations must guarantee that unique transaction identifiers are allocated to all transactions that originate from a logical call agent, as defined in the "states, failover and race conditions" section. Gateways can simply detect duplicate transactions by looking at the transaction identifier only.

The Response Acknowledgement Attribute can be found in any command. It carries a set of "confirmed transaction-id ranges."

MGCP gateways may choose to delete the copies of the responses to transactions whose id is included in "confirmed transaction-id ranges" received in the Response Confirmation messages. They should silently discard further commands from that Call Agent when the transaction-id falls within these ranges.

The "confirmed transaction-id ranges" values shall not be used if more than LONG-TIMER seconds have elapsed since the gateway issued its last response to that call agent, or when a gateway resumes operation. In this situation, commands should be accepted and processed, without any test on the transaction-id.

Commands that carry the "Response Acknowledgement attribute" may be transmitted in disorder. The gateway shall retain the union of the "confirmed transaction-id ranges" received in recent commands.

3.6.3. Computing retransmission timers

It is the responsibility of the requesting entity to provide suitable time outs for all outstanding commands, and to retry commands when time outs have been exceeded. Furthermore, when repeated commands fail to be acknowledged, it is the responsibility of the requesting entity to seek redundant services and/or clear existing or pending connections.

The specification purposely avoids specifying any value for the retransmission timers. These values are typically network dependent. The retransmission timers should normally estimate the timer by measuring the time spent between the sending of a command and the return of a response. One possibility is to use the algorithm implemented in TCP-IP, which uses two variables:

* the average acknowledgement delay, AAD, estimated through an exponentially smoothed average of the observed delays,

* the average deviation, ADEV, estimated through an exponentially smoothed average of the absolute value of the difference between the observed delay and the current average

The retransmission timer, in TCP, is set to the sum of the average delay plus N times the average deviation. In MGCP, the maximum value of the timer should however be bounded, in order to guarantee that no repeated packet will be received by the gateways after LONG-TIMER seconds. A suggested maximum value is 4 seconds.

After any retransmission, the MGCP entity should do the following:

* It should double the estimated value of the average delay, AAD

* It should compute a random value, uniformly distributed between 0.5 AAD and AAD

* It should set the retransmission timer to the sum of that random value and N times the average deviation.

This procedure has two effects. Because it includes an exponentially increasing component, it will automatically slow down the stream of messages in case of congestion. Because it includes a random component, it will break the potential synchronization between notifications triggered by the same external event.

3.6.4. Piggy backing

There are cases when a Call Agent will want to send several messages at the same time to the same gateways. When several MGCP messages have to be sent in the same UDP packets, they should be separated by a line of text that contain a single dot, as in for example:

```
200 2005 OK
DLCX 1244 card23/21@trgw-7.example.net MGCP 1.0
C: A3C47F21456789F0
I: FDE234C8
```

The piggy-backed messages should be processed exactly has if they had been received in several simultaneous messages.

3.6.5. Provisional responses

Executing some transactions may require a long time. Long execution times may interact with the timer based retransmission procedure. This may result either in an inordinate number of retransmissions, or in timer values that become too long to be efficient.

Gateways that can predict that a transaction will require a long execution time may send a provisional response, with response code 100. They should send this response if they receive a repetition of a transaction that is still being executed.

MGCP entities that receive a provisional response shall switch to a longer repetition timer for that transaction.

4. States, failover and race conditions.

In order to implement proper call signalling, the Call Agent must keep track of the state of the endpoint, and the gateway must make sure that events are properly notified to the call agent. Special conditions exist when the gateway or the call agent are restarted: the gateway must be redirected to a new call agent during "failover" procedures, the call agent must take special action when the gateway is taken offline, or restarted.

4.1. Basic Asumptions

The support of "failover" is based on the following assumptions:

* Call Agents are identified by their domain name, not their network addresses, and several addresses can be associated with a domain name.

* An endpoint has one NotifiedEntity associated with it any given point in time.

* The NotifiedEntity is the last value of the "NotifiedEntity" parameter received for this endpoint (including wild-carded end-point-names). If no explicit "NotifiedEntity" parameter has been received, the "NotifiedEntity" defaults to the provisioned NotifiedEntity value, or if no value was provisioned to the source address of the last command received for the endpoint,

* Responses to commands are always sent to the source address of the command, regardless of the NotifiedEntity.

* When the "notified entity" refers to a domain name that resolves to multiple IP- address, endpoints are capable of switching between different interfaces on the same logical call agent, however they cannot switch to other (backup) call agent(s) on their own. A backup call agent can however instruct them to switch, either directly or indirectly.

* If an entire call agent becomes unavailable, the endpoints managed by that call agent will eventually become "disconnected". The only way for these endpoints to become connected again is either for the failed call agent to become available, or for a backup call agent to contact the affected endpoints.

* When a backup call agent has taken over control of a group of endpoints, it is assumed that the failed call agent will communicate and synchronize with the backup call agent in order to

transfer control of the affected endpoints back to the original call agent (if that's even desired - maybe the failed call agent should simply become the backup call agent now).

We should note that handover conflict resolution between separate CA's is not in place - we are relying strictly on the CA's knowing what they are doing and communicating with each other (although AuditEndpoint can be used to learn about the current NotifiedEntity).

4.2. Security, Retransmission, and Detection of Lost Associations:

The media gateway control protocol is organized as a set of transactions, each of which is composed of a command and a response, commonly referred to as an acknowledgement. The MGCP messages, being carried over UDP, may be subject to losses. In the absence of a timely response, commands are repeated. MGCP entities are expected to keep in memory a list of the responses that they sent to recent transactions, i.e. a list of all the responses they sent over the last LONG-TIMER seconds, and a list of the transactions that are currently being executed.

The transaction identifiers of incoming commands are compared to the transaction identifiers of the recent responses. If a match is found, the MGCP entity does not execute the transaction, but simply repeats the response. The remaining commands will be compared to the list of current transaction. If a match is found, the MGCP entity does not execute the transaction, which is simply ignored - a response will be provided when the execution of the command is complete.

The repetition mechanism is used to guard against four types of possible errors:

* transmission errors, when for example a packet is lost due to noise on a line or congestion in a queue,

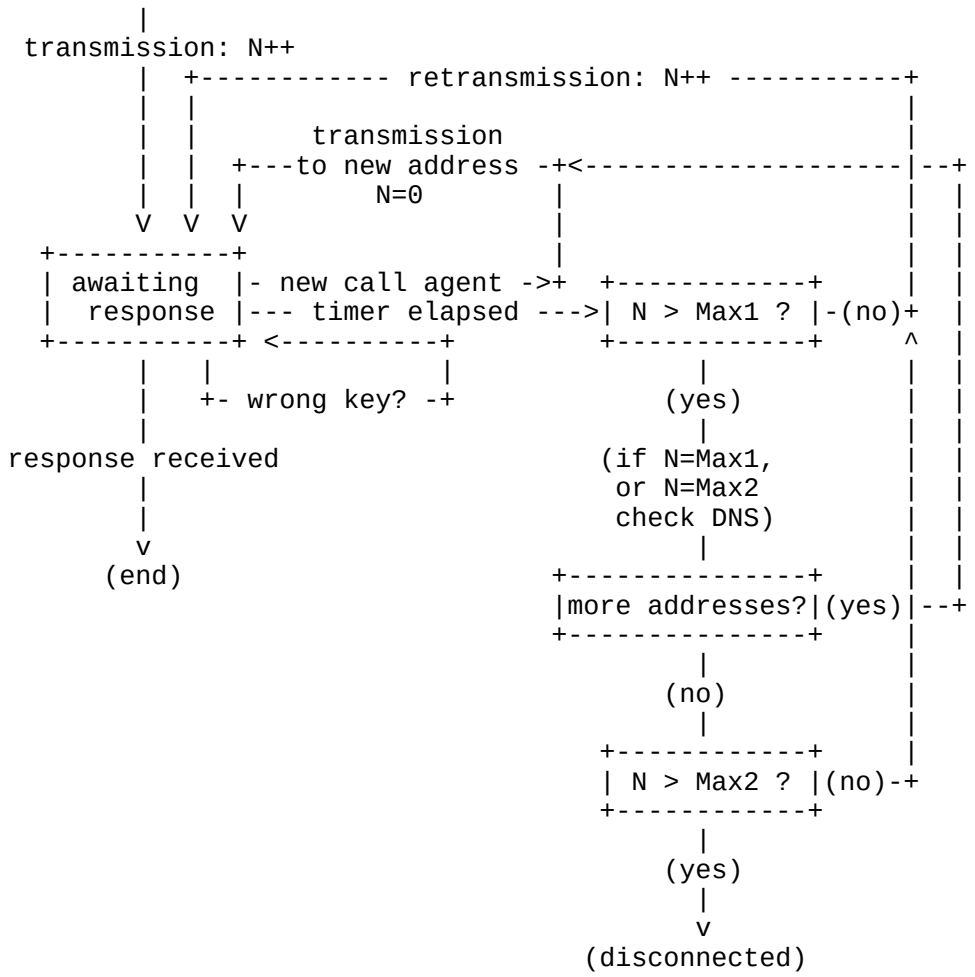
* component failure, when for example an interface to a call agent becomes unavailable,

* call agent failure, when for example an entire call agent becomes unavailable,

* failover, when a new call agent is "taking over" transparently.

The elements should be able to derive from the past history an estimate of the packet loss rate due to transmission errors. In a properly configured system, this loss rate should be kept very low, typically less than 1%. If a call agent or a gateway has to repeat a message more than a few times, it is very legitimate to assume that

something else than a transmission error is occurring. For example, given a loss rate of 1%, the probability that 5 consecutive transmission attempts fail is 1 in 100 billion, an event that should occur less than once every 10 days for a call agent that processes 1,000 transactions per second. (Indeed, the number of repetition that is considered excessive should be a function of the prevailing packet loss rate.) We should note that the "suspicion threshold", which we will call "Max1", is normally lower than the "disconnection threshold", which should be set to a larger value.



A classic retransmission algorithm would simply count the number of successive repetitions, and conclude that the association is broken after re-transmitting the packet an excessive number of times

(typically between 7 and 11 times.) In order to account for the possibility of an undetected or in-progress "failover", we modify the classic algorithm as follows:

* We request that the gateway always checks for the presence of a new call agent. It can be noticed either by

  - receiving a valid multicast message announcing a failover, or

  - receiving a command where the NotifiedEntity points to the new call agent, or

  - receiving a redirection response pointing to a new Call Agent.

  If a new call agent is detected, the gateway starts transmitting outstanding commands to that new agent. Responses to commands are still transmitted to the source address of the command.

* we request that if the number of repetitions for this Call Agent is larger than "Max1", that the gateway actively queries the name server in order to detect the possible change of the call agent interfaces.

* The gateway may have learned several IP addresses for the call agent. If the number of repetitions is larger than "Max1" and lower than "Max2", and there are more interfaces that have not been tried, then the gateway should direct the retransmissions to alternate addresses.

* If there are no more interfaces to try, and the number of repetitions is Max2, then the gateway contacts the DNS one more time to see if any other interface should have become available. If not, the gateway is now disconnected.

The procedure will maximize the chances of detecting an ongoing failover. It poses indeed two very specific problems, the potentially long delays of a timer based procedure and the risk of confusion caused by the use of cryptographic protections.

In order to automatically adapt to network load, MGCP specifies exponentially increasing timers. If the initial timer is set to 200 milliseconds, the loss of a fifth retransmission will be detected after about 6 seconds. This is probably an acceptable waiting delay to detect a failover. The repetitions should continue after that delay not only in order to perhaps overcome a transient connectivity problem, but also in order to allow some more time for the execution of a failover - waiting a total delay of 30 seconds is probably acceptable.

It is however important that the maximum delay of retransmissions be bounded. Prior to any retransmission, it is checked that the time elapsed since the sending of the initial datagram is no greater than T- MAX. If more than T-MAX time has elapsed, the endpoint becomes disconnected. The value T-MAX is related to the LONG-TIMER value: the LONG-TIMER value is obtained by adding to T-MAX the maximum propagation delay in the network.

Another potential cause of connection failure would be the reception of a "wrong key" message, sent by a call agent that could not authenticate the command, presumably because it had lost the security parameters of the association. Such messages are actually not authorized in IPSEC, and they should in fact not be taken at face value: an attacker could easily forge "wrong key" messages in order to precipitate the loss of a control connection. The current algorithm ignores these messages, which translates into a strict reliance on timers. The algorithm could in fact be improved, maybe by executing a check with the key server of the call agent after "Max1" repetitions.

4.3. Race conditions

MGCP deals with race conditions through the notion of a "quarantine list" and through explicit detection of desynchronization.

MGCP does not assume that the transport mechanism will maintain the order of command and responses. This may cause race conditions, that may be obviated through a proper behavior of the call agent. (Note that some race conditions are inherent to distributed systems; they would still occur, even if the commands were transmitted in strict order.)

In some cases, many gateways may decide to restart operation at the same time. This may occur, for example, if an area loses power or transmission capability during an earthquake or an ice storm. When power and transmission are reestablished, many gateways may decide to send "RestartInProgress" commands simultaneously, leading to very unstable operation.

4.3.1. Quarantine list

MGCP controlled gateways will receive "notification requests" that ask them to watch for a list of "events." The protocol elements that determine the handling of these events are the "Requested Events" list, the "Digit Map" and the "Detect Events" list.

When the endpoint is initialized, the requested events list and the digit map are empty. After reception of a command, the gateway starts observing the endpoint for occurrences of the events mentioned in the list.

The events are examined as they occur. The action that follows is determined by the "action" parameter associated to the event in the list of requested events, and also by the digit map. The events that are defined as "accumulate" or "treat according to digit map" are accumulated in a list of events, the events that are marked as "treated according to the digit map" will additionally be accumulated in the dialed string. This will go on until one event is encountered that triggers a Notification to the "notified entity."

The gateway, at this point, will transmit the notification command and will place the endpoint in a "notification" state. As long as the endpoint is in this notification state, the events that are to be detected on the endpoint are stored in a "quarantine" buffer for later processing. The events are, in a sense, "quarantined." All events that are specified by the union of the RequestedEvents parameter and the most recently received DetectEvent parameter or, in the absence of the latter, all events that are referred to in the RequestedEvents, should be detected and quarantined, regardless of the action associated to the event.

The endpoint exits the "notification state" when the acknowledgement of the Notify command is received. The Notify command may be retransmitted in the "notification state", as specified in section 3.5. When the endpoint exits the "notification state" it resets the list of observed events and the "current dial string" of the endpoint to a null value.

Following that point, the behavior of the gateway depends on the value of The QuarantineHandling parameter in the notification request. If the Call Agent specified that it expected at most one notification in response to the notification request command, then the gateway should simply keep on accumulating events in the quarantine list until it receives the next notification request command.

If the gateway is authorized to send multiple successive Notify commands, it will proceed as follows. When the gateway exits the "notification state", it resets the list of observed events and the "current dial string" of the endpoint to a null value and starts processing the list of quarantined events, using the already received list of requested events and digit map. When processing these events,

the gateway may encounter an event which requires a Notify command to be sent. If that is the case, the gateway can adopt one of the two following behaviors:

* it can immediately transmit a Notify command that will report all events that were accumulated in the list of observed events until the triggering event, included, leaving the unprocessed events in the quarantine list,

* or it can attempt to empty the quarantined list and transmit a single Notify command reporting several sets of events and possibly several dial strings. The dial string is reset to a null value after each triggering event. The events that follow the last triggering event are left in the quarantine list.

If the gateway transmits a Notify command, the end point will remain in the "notification state" until the acknowledgement is received. If the gateway does not find a quarantined event that requests a Notify command, it places the end point in a normal state. Events are then processed as they come, in exactly the same way as if a Notification Request command had just been received.

A gateway may receive at any time a new Notification Request command for the end point. When a new notification request is received in the notification state, the gateway shall ensure that the pending notification is received by the Call Agent prior to a successful response to the new NotificationRequest. It does so by using the "piggy-backing" functionality of the protocol. The messages will then be sent in a single packetto the source of the new NotificationRequest, regardless of respectively the source and "notified entity" for the old and new command. The steps involved are the following:

a) the gateway builds a message that carries in a single packet a repetition of the old pending Notify command and the acknowledgement of the new notification request.

b) the endpoint is then taken out of the "notification state" without waiting for the acknowledgement of the notification command.

c) a copy of the unacknowledged Notify command command is kept until an acknowledgement is received. If a timer elapses, the notification will be repeated, in a packet that will also carry a repetition of the acknowledgement of the notification request.

d) if the acknowledgement is lost, the Call Agent will retransmit the Notification Request. The gateway will reply to this repetition by retransmitting in a single packet the unacknowledged Notify and the acknowledgement of the notification request.

e) if the gateway has to transmit a Notify before the previous Notify is acknowledged, it should construct a packet that piggybacks a repetition of the old Notify, a repetition of the acknowledgement of the last notification request and the new Notify.

f) Gateways that cannot piggyback several packets in the same message should elect to leave the endpoint in the "notification" state as long as the last notification is not acknowledged.

After receiving the Notification Request command, the requested events list and digit map (if a new one was provided) are replaced by the newly received parameters, and the list of observed events and accumulated dial string are reset to a null value. The behavior is conditioned by the value of the QuarantineHandling parameter. The parameter may specify that quarantined events, or previously observed events, should be discarded, in which case they will be. If the parameter specifies that the quarantined events should be processed, the gateway will start processing the list of quarantined events or previously observed events, using the newly received list of requested events and digit map. When processing these events, the gateway may encounter an event which requires a Notify command to be sent. If that is the case, the gateway will immediately transmit a Notify command that will report all events that were accumulated in the list of observed events until the triggering event, included, leaving the unprocessed events in the quarantine buffer, and will enter the "notification state".

A new notification request may be received while the gateway has accumulated events according to the previous notification requests, but has not yet detected a notification-triggering events. The handling of not-yet-notified events is determined, as with the quarantined events, by the quarantine handling parameters:

* If the quarantine-handling parameter specifies that quarantined events shall be ignored, the observed event list is simply reset.

* If the quarantine-handling parameter specifies that quarantined events shall be processed, the observed event list is transferred to the quarantined event list. The observed event list is then reset, and the quarantined event list is processed.

Call Agents SHOULD provide the response to a successful Notify message and the new NotificationRequest in the same datagram using the piggy-backing mechanism.

4.3.2. Explicit detection

A key element of the state of several endpoints is the position of the hook. A race condition may occur when the user decides to go off-hook before the Call Agent has the time to ask the gateway to notify an off hook event (the "glare" condition well known in telephony), or if the user goes on-hook before the Call Agent has the time to request the event's notification.

To avoid this race condition, the gateway should check the condition of the endpoint before acknowledging a NotificationRequest. It should return an error:

1- If the gateway is requested to notify an "off hook" transition while the phone is already off hook,

2- If the gateway is requested to notify an "on hook" or "flash hook" condition while the phone is already on hook.

It should be noted, that the condition check is performed at the time the notification request is received, where as the actual event that caused the current condition may have either been reported, or ignored earlier, or it may currently be quarantined.

The other state variables of the gateway, such as the list of RequestedEvent or list of requested signals, are entirely replaced after each successful NotificationRequest, which prevents any long term discrepancy between the Call Agent and the gateway.

When a NotificationRequest is unsuccessful, whether it is included in a connection-handling command or not, the gateway will simply continue as if the command had never been received. As all other transactions, the NotificationRequest should operate as an atomic transaction, thus any changes initiated as a result of the command should be reverted.

Another race condition may occur when a Notify is issued shortly before the reception by the gateway of a NotificationRequest. The RequestIdentifier is used to correlate Notify commands with NotificationRequest commands.

4.3.3. Ordering of commands, and treatment of disorder

MGCP does not mandate that the underlying transport protocol guarantees the sequencing of commands sent to a gateway or an endpoint. This property tends to maximize the timeliness of actions, but it has a few draw backs. For example:

* Notify commands may be delayed and arrive to the call agent after the transmission of a new Notification Request command,

* If a new NotificationRequest is transmitted before a previous one is acknowledged, there is no guarantee that the previous one will not be received in second position.

Call Agents that want to guarantee consistent operation of the end points can use the following rules:

1) When a gateway handles several endpoints, commands pertaining to the different endpoints can be sent in parallel, for example following a model where each endpoint is controlled by its own process or its own thread.

2) When several connections are created on the same endpoint, commands pertaining to different connections can be sent in parallel.

3) On a given connection, there should normally be only one outstanding command (create or modify). However, a DeleteConnection command can be issued at any time. In consequence, a gateway may sometimes receive a ModifyConnection command that applies to a previously deleted connection. Such commands should be ignored, and an error code should be returned.

4) On a given endpoint, there should normally be only one outstanding NotificationRequest command at any time. The RequestId parameter should be used to correlate Notify commands with the triggering notification request.

5) In some cases, an implicitly or explicitly wildcarded DeleteConnection command that applies to a group of endpoints can step in front of a pending CreateConnection command. The Call Agent should individually delete all connections whose completion was pending at the time of the global DeleteConnection command. Also, new CreateConnection commands for endpoints named by the wild-carding cannot be sent until the wild-carded DeleteConnection command is acknowledged.

6) When commands are embedded within each other, sequencing requirements for all commands must be adhered to. For example a Create Connection command with a Notification Request in it must adhere to the sequencing for CreateConnection and NotificationRequest at the same time.

7) AuditEndpoint and AuditConnection is not subject to any sequencing.

8) RestartInProgress must always be the first command sent by an endpoint as defined by the restart procedure. Any other command or response must be delivered after this RestartInProgress command (piggy-backing allowed).

9) When multiple messages are piggy-backed in a single packet, the messages are always processed in order.

These rules do not affect the gateway, which should always respond to commands.

4.3.4. Fighting the restart avalanche

Let's suppose that a large number of gateways are powered on simultaneously. If they were to all initiate a RestartInProgress transaction, the call agent would very likely be swamped, leading to message losses and network congestion during the critical period of service restoration. In order to prevent such avalanches, the following behavior is suggested:

1) When a gateway is powered on, it should initiate a restart timer to a random value, uniformly distributed between 0 and a maximum waiting delay (MWD). Care should be taken to avoid synchronicity of the random number generation between multiple gateways that would use the same algorithm.

2) The gateway should then wait for either the end of this timer, the reception of a command from the call agent, or the detection of a local user activity, such as for example an off-hook transition on a residential gateway.

3) When the timer elapses, when a command is received, or when an activity is detected, the gateway should initiate the restart procedure.

The restart procedure simply requires the endpoint to guarantee that the first message (command or response) that the Call Agent sees from this endpoint is a RestartInProgress message informing the Call Agent about the restart. The endpoint is free to take full advantage of piggy-backing to achieve this.

It is expected that each endpoint in a gateway will have a provisionable Call Agent, i.e., "notified entity", to direct the initial restart message towards. When the collection of endpoints in a gateway is managed by more than one Call Agent, the above procedure must be performed for each collection of endpoints managed by a given Call Agent. The gateway MUST take full advantage of wild-carding to minimize the number of RestartInProgress messages generated when multiple endpoints in a gateway restart and the endpoints are managed by the same Call Agent.

The value of MWD is a configuration parameter that depends on the type of the gateway. The following]reasoning can be used to determine the value of this delay on residential gateways.

Call agents are typically dimensioned to handle the peak hour traffic load, during which, in average, 10% of the lines will be busy, placing calls whose average duration is typically 3 minutes. The processing of a call typically involves 5 to 6 MGCP transactions between each end point and the call agent. This simple calculation shows that the call agent is expected to handle 5 to 6 transactions for each end point, every 30 minutes on average, or, to put it otherwise, about one transaction per end point every 5 to 6 minutes on average. This suggest that a reasonable value of MWD for a residential gateway would be 10 to 12 minutes. In the absence of explicit configuration, residential gateways should adopt a value of 600 seconds for MWD.

The same reasoning suggests that the value of MWD should be much shorter for trunking gateways or for business gateways, because they handle a large number of endpoints, and also because the usage rate of these endpoints is much higher than 10% during the peak busy hour, a typical value being 60%. These endpoints, during the peak hour, are this expected to contribute about one transaction per minute to the call agent load. A reasonable algorithm is to make the value of MWD per "trunk" endpoint six times shorter than the MWD per residential gateway, and also inversely proportional to the number of endpoints that are being restarted. for example MWD should be set to 2.5 seconds for a gateway that handles a T1 line, or to 60 milliseconds for a gateway that handles a T3 line.

4.3.5. Disconnected Endpoints

In addition to the restart procedure, gateways also have a "disconnected" procedure, which is initiated when an endpoint becomes "disconnected" as described in Section 3.4.2. It should here be noted, that endpoints can only become disconnected when they attempt to communicate with the Call Agent. The following steps are followed by an endpoint that becomes "disconnected":

1. A "disconnected" timer is initialized to a random value, uniformly distributed between 0 and a provisionable "disconnected" initial waiting delay (Tdinit), e.g., 15 seconds. Care MUST be taken to avoid synchronicity of the random number generation between multiple gateways and endpoints that would use the same algorithm.

2. The gateway then waits for either the end of this timer, the reception of a command from the call agent, or the detection of a local user activity for the endpoint, such as for example an off-hook transition.

3. When the "disconnected" timer elapses, when a command is received, or when a local user activity is detected, the gateway initiates the "disconnected" procedure for the endpoint. In the case of local user activity, a provisionable "disconnected" minimum waiting delay (Tdmin) must furthermore have elapsed since the gateway became disconnected or the last time it initiated the "disconnected" procedure in order to limit the rate at which the procedure is performed.

4. If the "disconnected" procedure still left the endpoint disconnected, the "disconnected" timer is then doubled, subject to a provisionable "disconnected" maximum waiting delay (Tdmax), e.g., 600 seconds, and the gateway proceeds with step 2 again.

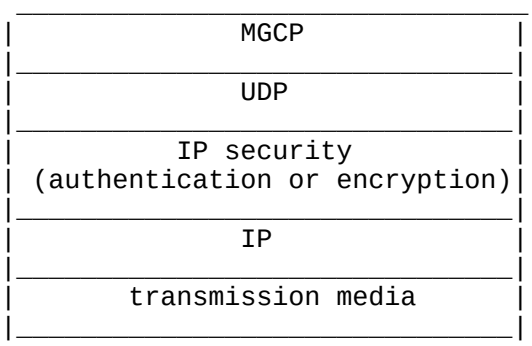
The "disconnected" procedure is similar to the restart procedure in that it now simply states that the endpoint MUST send a RestartInProgress command to the Call Agent informing it that the endpoint was disconnected and furthermore guarantee that the first message (command or response) that the Call Agent now sees from this endpoint MUST be this RestartInProgress command. The endpoint MUST take full advantage of piggy-backing in achieving this. The Call Agent may then for instance decide to audit the endpoint, or simply clear all connections for the endpoint.

This specification purposely does not specify any additional behavior for a disconnected endpoint. Vendors MAY for instance choose to provide silence, play reorder tone, or even enable a downloaded wav file to be played.

The default value for Tdinit is 15 seconds, the default value for Tdmin, is 15 seconds, and the default value for Tdmax is 600 seconds.

5. Security requirements

If unauthorized entities could use the MGCP, they would be able to set-up unauthorized calls, or to interfere with authorized calls. We expect that MGCP messages will always be carried over secure Internet connections, as defined in the IP security architecture as defined in RFC 2401, using either the IP Authentication Header, defined in RFC 2402, or the IP Encapsulating Security Payload, defined in RFC 2406. The complete MGCP protocol stack would thus include the following layers:



Adequate protection of the connections will be achieved if the gateways and the Call Agents only accept messages for which IP security provided an authentication service. An encryption service will provide additional protection against eavesdropping, thus forbidding third parties from monitoring the connections set up by a given endpoint

The encryption service will also be requested if the session descriptions are used to carry session keys, as defined in SDP.

These procedures do not necessarily protect against denial of service attacks by misbehaving gateways or misbehaving call agents. However, they will provide an identification of these misbehaving entities, which should then be deprived of their authorization through maintenance procedures.

5.1. Protection of media connections

MGCP allows call agent to provide gateways with "session keys" that can be used to encrypt the audio messages, protecting against eavesdropping.

A specific problem of packet networks is "uncontrolled barge-in." This attack can be performed by directing media packets to the IP address and UDP port used by a connection. If no protection is implemented, the packets will be decompressed and the signals will be played on the "line side".

A basic protection against this attack is to only accept packets from known sources, checking for example that the IP source address and UDP source port match the values announced in the "remote session description." But this has two inconveniences: it slows down connection establishment and it can be fooled by source spoofing:

* To enable the address-based protection, the call agent must obtain the remote session description of the e-gress gateway and pass it to the in-gress gateway. This requires at least one network round trip, and leaves us with a dilemma: either allow the call to proceed without waiting for the round trip to complete, and risk for example "clipping" a remote announcement, or wait for the full round trip and settle for slower call-set-up procedures.

* Source spoofing is only effective if the attacker can obtain valid pairs of source destination addresses and ports, for example by listening to a fraction of the traffic. To fight source spoofing, one could try to control all access points to the network. But this is in practice very hard to achieve.

An alternative to checking the source address is to encrypt and authenticate the packets, using a secret key that is conveyed during the call set-up procedure. This will no slow down the call set-up, and provides strong protection against address spoofing.

6. Event packages and end point types

This section provides an initial definition of packages and event names. More packages can be defined in additional documents.

6.1. Basic packages

The list of basic packages includes the following:

Package	name
Generic Media Package	G
DTMF package	D
MF Package	M
Trunk Package	T
Line Package	L
Handset Package	H
RTP Package	R
Network Access Server Package	N
Announcement Server Package	A
Script Package	Script

In the tables of events for each package, there are five columns:

Symbol: the unique symbol used for the event
Definition: a short description of the event

R: an x appears in this column is the event can be Requested by the call agent.

S: if nothing appears in this column for an event, then the event cannot be signaled on command by the call agent. Otherwise, the following symbols identify the type of event:

OO On/Off signal. The signal is turned on until commanded by the call agent to turn it off, and vice versa.

TO Timeout signal. The signal lasts for a given duration unless it is superseded by a new signal.

BR Brief signal. The event has a short, known duration.

Duration: specifies the duration of TO signals.

6.1.1. Generic Media Package

Package Name: G

The generic media package group the events and signals that can be observed on several types of endpoints, such as trunking gateways, access gateways or residential gateways.

Symbol	Definition	R	S	Duration
mt	Modem detected	x		
ft	Fax tone detected	x		
ld	Long duration connection	x		
pat(###)	Pattern ### detected	x	OO	
rt	Ringback tone		TO	
rbk(###)	ring back on connection		TO	180 seconds
cf	Confirm tone		BR	
cg	Network Congestion tone		TO	
it	Intercept tone		OO	
pt	Preemption tone		OO	
of	report failure	x		

The signals are defined as follows:

The pattern definition can be used for specific algorithms such as answering machine detection, tone detection, and the like.

Ring back tone (rt)
an Audible Ring Tone, a combination of two AC tones with frequencies of 440 and 480 Hertz and levels of -19 dBm each, to give a combined level of -16 dBm. The cadence for Audible Ring Tone is 2 seconds on followed by 4 seconds off. See GR- 506-CORE - LSSGR: SIGNALING, Section 17.2.5.

Ring back on connection
A ring back tone, applied to the connection whose identifier is passed as a parameter.

The "long duration connection" is detected when a connection has been established for more than 1 hour.

6.1.2. DTMF package

Package name: D

Symbol	Definition	R	S	Duration
0	DTMF 0	x	BR	
1	DTMF 1	x	BR	
2	DTMF 2	x	BR	
3	DTMF 3	x	BR	
4	DTMF 4	x	BR	
5	DTMF 5	x	BR	
6	DTMF 6	x	BR	
7	DTMF 7	x	BR	
8	DTMF 8	x	BR	
9	DTMF 9	x	BR	
#	DTMF #	x	BR	
*	DTMF *	x	BR	
A	DTMF A	x	BR	
B	DTMF B	x	BR	
C	DTMF C	x	BR	
D	DTMF D	x	BR	
L	long duration indicator	x		2 seconds
X	Wildcard, match any digit 0-9	x		
T	Interdigit timer	x		4 seconds
of	report failure	x		

The "interdigit timer" T is a digit input timer that can be used in two ways:

* When timer T is used with a digit map, the timer is not started until the first digit is entered, and the timer is restarted after each new digit is entered until either a digit map match or mismatch occurs. In this case, timer T functions as an inter-digit timer.

* When timer T is used without a digit map, the timer is started immediately and simply cancelled (but not restarted) as soon as a digit is entered. In this case, timer T can be used as an interdigit timer when overlap sending is used.

  When used with a digit map, timer T takes on one of two values, T(partial) or T(critical). When at least one more digit is required for the digit string to match any of the patterns in the digit map, timer T takes on the value T(partial), corresponding to

partial dial timing. If a timer is all that is required to produce a match, timer T takes on the value T(critical) corresponding to critical timing. When timer T is used without a digit map, timer T takes on the value T(critical). The default value for T(partial) is 16 seconds and the default value for T(critical) is 4 seconds. The provisioning process may alter both of these.

The "long duration indicator" is observed when a DTMF signal is produced for a duration larger than two seconds. In this case, the gateway will detect two successive events: first, when the signal has been recognized, the DTMF signal, and then, 2 seconds later, the long duration signal.

6.1.3. MF Package

Package Name: M

Symbol	Definition	R	S	Duration
0	MF 0	x	BR	
1	MF 1	x	BR	
2	MF 2	x	BR	
3	MF 3	x	BR	
4	MF 4	x	BR	
5	MF 5	x	BR	
6	MF 6	x	BR	
7	MF 7	x	BR	
8	MF 8	x	BR	
9	MF 9	x	BR	
X	Wildcard, match any digit 0-9	x		
T	Interdigit timer	x		4 seconds
K0	MF K0 or KP	x	BR	
K1	MF K1	x	BR	
K2	MF K2	x	BR	
S0	MF S0 or ST	x	BR	
S1	MF S1	x	BR	
S2	MF S2	x	BR	
S3	MF S3	x	BR	
wk	Wink	x	BR	
wko	Wink off	x	BR	
is	Incoming seizure	x	OO	
rs	Return seizure	x	OO	
us	Unseize circuit	x	OO	
of	report failure	x		

The definition of the MF package events is as follows:

Wink
A transition from unseized to seized to unseized trunk states within a specified period. Typical seizure period is 100-350 msec.)

Incoming seizure
Incoming indication of call attempt.

Return seizure:
Seizure in response to outgoing seizure.

Unseize circuit:
Unseizure of a circuit at the end of a call.

Wink off:
A signal used in operator services trunks. A transition from seized to unseized to seized trunk states within a specified period of 100-350 ms. (To be checked)

6.1.4. Trunk Package

Package Name: T

Symbol	Definition	R	S	Duration
co1	Continuity tone (single tone, or return tone)	x	OO	
co2	Continuity test (go tone, in dual tone procedures)	x	OO	
lb	Loopback		OO	
om	Old Milliwatt Tone (1000 Hz)	x	OO	
nm	New Milliwatt Tone (1004 Hz)	x	OO	
tl	Test Line	x	OO	
zz	No circuit	x	OO	
as	Answer Supervision	x	OO	
ro	Reorder Tone	x	TO	30 seconds
of	report failure	x		
bl	Blocking		OO	

The definition of the trunk package signal events is as follows:

Continuity Tone (co1):
A tone at 2010 + or - 30 Hz.

Continuity Test (co2):
A tone at the 1780 + or - 30 Hz.

Milliwatt Tones:
Old Milliwatt Tone (1000 Hz), New Milliwatt Tone (1004 Hz)

Line Test:
105 Test Line test progress tone (2225 Hz + or - 25 Hz at -10 dBm0 + or -- 0.5dB).

No circuit:
(that annoying tri-tone, low to high)

Answer Supervision:

Reorder Tone:
Reorder tone is a combination of two AC tones with frequencies of 480 and 620 Hertz and levels of -24 dBm each, to give a combined level of -21 dBm. The cadence for Station Busy Tone is 0.25 seconds on followed by 0.25 seconds off, repeating continuously. See GR-506-CORE - LSSGR: SIGNALING, Section 17.2.7.

Blocking:
The call agent can place the circuit in a blocked state by applying the "bl(+)" signal to the endpoint. It can unblock it by applying the "bl(-)" signal.

The continuity tones are used when the call agent wants to initiate a continuity test. There are two types of tests, single tone and dual tone. The Call agent is expected to know, through provisioning information, which test should be applied to a given endpoint. For example, the call agent that wants to initiate a single frequency test will send to the gateway a command of the form:

```
RQNT 1234 epx-t1/17@tgw2.example.net
X: AB123FE0
S: co1
R: co1
```

If it wanted instead to initiate a dual-tone test, it would send the command:

```
RQNT 1234 epx-t1/17@tgw2.example.net
X: AB123FE0
S: co2
R: co1
```

The gateway would send the requested signal, and in both cases would look for the return of the 2010 Hz tone (co1). When it detects that tone, it will send the corresponding notification.

The tones are of type OO: the gateway will keep sending them until it receives a new notification request.

6.1.5. Line Package

Package Name: L

Symbol	Definition	R	S	Duration
adsi(string)	adsi display		BR	
vmwi	visual message waiting indicator		OO	
hd	Off hook transition	x		
hu	On hook transition	x		
hf	Flash hook	x		
aw	Answer tone	x	OO	
bz	Busy tone		TO	30 seconds
ci(ti,nu,na)	Caller-id		BR	
wt	Call Waiting tone		TO	30 seconds
wt1, wt2, wt3, wt4	Alternative call waiting tones			
dl	Dial tone		TO	16 seconds
mwi	Message waiting ind.		TO	16 seconds
nbz	Network busy (fast cycle busy)	x	OO	
ro	Reorder tone		TO	30 seconds
rg	Ringing		TO	180 seconds
r0, r1, r2, r3, r4, r5, r6 or r7	Distinctive ringing		TO	180 seconds
rs	Ringsplash		BR	
p	Prompt tone	x	BR	
e	Error tone	x	BR	
sl	Stutter dialtone		TO	16 seconds
v	Alerting Tone		OO	
y	Recorder Warning Tone		OO	
sit	SIT tone			
z	Calling Card Service Tone		OO	
oc	Report on completion	x		
ot	Off hook warning tone		TO	indefinite
s(###)	Distinctive tone pattern	x	BR	
of	report failure	x		

The definition of the tones is as follows:

Dial tone:
A combined 350 + 440 Hz tone.

Visual Message Waiting Indicator
The transmission of the VMWI messages will conform to the requirements in Section 2.3.2, "On-hook Data Transmission Not Associated with Ringing" in TR-H-000030 and the CPE guidelines in SR-TSV-002476. VMWI messages will only be sent from the SPCS when the line is idle. If new messages arrive while the line is busy, the VMWI indicator message will be delayed until the line goes back to the idle state. The CA should periodically refresh the CPE's visual indicator. See TR-NWT-001401 - Visual Message Waiting Indicator Generic Requirements; and GR- 30-CORE - Voiceband Data Transmission Interface.

Message waiting Indicator
See GR-506-CORE, 17.2.3.

Alerting Tone:
a 440 Hz Tone of 2 second duration followed by 1/2 second of tone every 10 seconds.

Ring splash
Ringsplash, also known as "Reminder ring" is a burst of ringing that may be applied to the physical forwarding line (when idle) to indicate that a call has been forwarded and to remind the user that a CF subfeature is active. In the US, it is defined to be a 0.5(-0,+0.1) second burst of power ringing. See TR-TSY-000586 - Call Forwarding Subfeatures.

Call waiting tone
Call Waiting tone is defined in GR-506-CORE, 14.2. Call Waiting feature is defined in TR-TSY-000571. By defining "wt" as a TO signal you are really defining the feature which seems wrong to me (given the spirit of MGCP), hence the definition of "wt" as a BR signal in ECS, per GR-506-CORE. Also, it turns out that there is actually four different call waiting tone patterns (see GR-506-CORE, 14.2) so we have wt1, wt2, wt3, wt4.

Caller Id (ci(time, number, name)):
The caller-id event carries three parameters, the time of the call, the calling number and the calling name. Each of the three fields are optional, however each of the commas will always be included. See TR-NWT-001188, GR-30-CORE, and TR-NWT-000031.

Recorder Warning Tone:
1400 Hz of Tone of 0.5 second duration every 15 seconds.

SIT tone:
used for indicating a line is out of service.

Calling Card Service Tone:
60 ms of 941 + 1477 Hz and 940 ms of 350 + 440 Hz (dial tone), decaying exponentially with a time constant of 200 ms.

Distinctive tone pattern:
where ### is any number between 000 and 999, inclusive. Can be used for distinctive ringing, customized dial tone, etc.

Report on completion
The report on completion event is detected when the gateway was asked to perform one or several signals of type TO on the endpoint, and when these signals were completed without being stopped by the detection of a requested event such as off-hook transition or dialed digit. The completion report may carry as parameter the name of the signal that came to the end of its live time, as in:

O: L/oc(L/dl)

Ring back on connection
A ring back tone, applied to the connection wghose identifier is passed as a parameter.

We should note that many of these definitions vary from country to country. The frequencies listed above are the one in use in North America. There is a need to accommodate different tone sets in different countries, and there is still an ongoing debate on the best way to meet that requirement:

* One solution is to define different event packages specifying for example the German dialtone as "L-DE/DL".

* Another solution is to use a management interface to specify on an endpoint basis which frequency shall be associated to what tone.

6.1.6. Handset emulation package

Package Name: H

Symbol	Definition	R	S	Duration
adsi(string)	adsi display	x	BR	
tdd				
vmwi				
hd	Off hook transition	x	OO	
hu	On hook transition	x	OO	
hf	Flash hook	x	BR	
aw	Answer tone	x	OO	
bz	Busy tone	x	OO	
wt	Call Waiting tone	x	TO	30 seconds
dl	Dial tone (350 + 440 Hz)	x	TO	120 seconds
nbz	Network busy	x	OO	
	(fast cycle busy)			
rg	Ringing	x	TO	30 seconds
r0, r1, r2,	Distinctive ringing	x	TO	30 seconds
r3, r4, r5,				
r6 or r7				
p	Prompt tone	x	BR	
e	Error tone	x	BR	
sdl	Stutter dialtone	x	TO	16 seconds
v	Alerting Tone	x	OO	
y	Recorder Warning Tone	x	OO	
t	SIT tone	x		
z	Calling Card Service Tone	x	OO	
oc	Report on completion	x		
ot	Off hook warning tone	x	OO	
s(###)	Distinctive tone pattern	x	BR	
of	report failure	x		

The handset emulation package is an extension of the line package, to be used when the gateway is capable of emulating a handset. The difference with the line package is that events such as "off hook" can be signalled as well as detected.

6.1.7. RTP Package

Package Name: R

Symbol	Definition	R	S Duration
UC	Used codec changed	x	
SR(###)	Sampling rate changed	x	
JI(###)	Jitter buffer size changed	x	
PL(###)	Packet loss exceeded	x	
qa	Quality alert	x	
co1	Continuity tone (single tone, or return tone)	x	OO
co2	Continuity test (go tone, in dual tone procedures)	x	OO
of	report failure	x	

Codec Changed:
Codec changed to hexadecimal codec number enclosed in parenthesis, as in UC(15), to indicate the codec was changed to PCM mu-law. Codec Numbers are specified in RFC 1890, or in a new definition of the audio profiles for RTP that replaces this RFC. Some implementations of media gateways may not allow the codec to be changed upon command from the call agent. codec changed to codec hexadecimal ##.

Sampling Rate Changed:
Sampling rate changed to decimal number in milliseconds enclosed in parenthesis, as in SR(20), to indicate the sampling rate was changed to 20 milliseconds. Some implementations of media gateways may not allow the sampling rate to be changed upon command from a call agent.

Jitter Buffer Size Changed:
When the media gateway has the ability to automatically adjust the depth of the jitter buffer for received RTP streams, it is useful for the media gateway controller to receive notification that the media gateway has automatically increased its jitter buffer size to accomodate increased or decreased variability in network latency. The syntax for requesting notification is "JI", which tells the media gateway that the controller wants notification of any jitter buffer size changes. The syntax for notification from the media gateway to the controller is "JI(####)", where the #### is the new size of the jitter buffer, in milliseconds.

Packet Loss Exceeded:
Packet loss rate exceed the threshold of the specified decimal number of packets per 100,000 packets, where the packet loss number is contained in parenthesis. For example, PL(10) indicates packets are being dropped at a rate of 1 in 10,000 packets.

Quality alert
The packet loss rate or the combination of delay and jitter exceed a specified quality threshold.

The continuity tones are the same as those defined in the Trunk package. They can be use in conjunction with the Network LoopBack or Network Continuity Test modes to test the continuity of an RTP circuit.

The "operation failure" code can be used to report problems such as the loss of underlying connectivity. The observed event can include as parameter the reason code of the failure.

6.1.8. Network Access Server Package

Package Name: N

Symbol	Definition	R	S	Duration
pa	Packet arrival	x		
cbk	Call back request	x		
cl	Carrier lost	x		
au	Authorization succeeded	x		
ax	Authorization denied	x		
of	Report failure	x		

The packet arrival event is used to notify that at least one packet was recently sent to an Internet address that is observed by an endpoint. The event report includes the Internet address, in standard ASCII encoding, between parenthesis:

O: pa(192.96.41.1)

The call back event is used to notify that a call back has been requested during the initial phase of a data connection. The event report includes the identification of the user that should be called back, between parenthesis:

O: cbk(user25)

6.1.9. Announcement Server Package

Package Name: A

Symbol	Definition	R	S	Duration
ann(url,parms)	Play an announcement		TO	variable
oc	Report on completion	x		
of	Report failure	x		

The announcement action is qualified by an URL name and by a set of initial parameters as in for example:

S: ann(http://scripts.example.net/all-lines-busy.au)

The "operation complete" event will be detected when the announcement is played out. If the announcement cannot be played out, an operation failure event can be returned. The failure may be explained by a commentary, as in:

O: A/of(file not found)

6.1.10. Script Package

Package Name: Script

Symbol	Definition	R	S	Duration
java(url)	Load a java script		TO	variable
perl(url)	Load a perl script		TO	variable
tcl(url)	Load a TCL script		TO	variable
xml(url)	Load an XML script		TO	variable
oc	Report on completion	x		
of	Report failure	x		

The "language" action define is qualified by an URL name and by a set of initial parameters as in for example:

S: script/java(http://scripts.example.net/credit-
card.java,long,1234)

The current definition defines keywords for the most common languages. More languages may be defined in further version of this documents. For each language, an API specification will describe how the scripts can issue local "notificationRequest" commands, and receive the corresponding notifications.

The script produces an output which consists of one or several text string, separated by commas. The text string are reported as a commentary in the report on completion, as in for example:

O: script/oc(21223456794567,9738234567)

The failure report may also return a string, as in:

O: script/oc(21223456794567,9738234567)

The definition of the script environment and the specific actions in that environment are for further study.

6.2. Basic endpoint types and profiles

We define the following basic endpoint types and profiles:

* Trunk gateway (ISUP)
* Trunk gateway (MF)
* Network Access Server (NAS)
* Combined NAS/VOIP gateway
* Access Gateway
* Residential Gateway
* Announcement servers

These gateways are supposed to implement the following packages

Gateway	Supported packages
Trunk gateway (ISUP)	GM, DTMF, TK, RTP
Trunk gateway (MF)	GM, MF, DTMF, TK, RTP
Network Access Server (NAS)	GM, MF, TK, NAS
Combined NAS/VOIP gateway	GM, MF, DTMF, TK, NAS, RTP
Access Gateway (VOIP)	GM, DTMF, MF, RTP
Access Gateway (VOIP+NAS)	GM, DTMF, MF, NAS, RTP
Residential Gateway	GM, DTMF, Line, RTP
Announcement Server	ANN, RTP

Advanced announcement servers may also support the Script package.

Advanced trunking servers may support the ANN package, the Script package, and in some cases the Line and Handset package as well.

7. Versions and compatibility

7.1. Differences between version 1.0 and draft 0.5

Draft 0-5 was issued in February 1999, as the last update of draft version 0.1. Version 1.0 benefits from implementation experience, and also aligns as much as possible with the CableLabs' NCS project. The main differences between the February draft and version 1.0 are:

* Specified more clearly that the encoding of three LocalConnectionOptions parameters, Encoding Method, Packetization Period and Bandwidth, shall follow the conventions laid out in SDP.

* Specified how the quarantine handling parameter governs the handling of detected but not yet specified events.

* Specified that unexpected timers or digits should trigger transmission of the dialed string.

* Removed the digit map syntax description from section 2.1.5 (it was redundant with section 3.4.)

* Corrected miscellaneous bugs in the formal syntax description.

* Aligned specification of commands with the CableLabs NCS specification. This mostly affects the AuditEndpoint and

RestartInProgress commands.

* Aligned the handling of retransmission with the CableLabs NCS specification.

* Added the provisional response return code and corresponding behavior description.

* Added an optional reason code parameter to restart in progress.

* Added the possibility to audit the restart method, restart delay and reason code.

7.2. Differences between draft-04 and draft-05

Differences are minor: corrected the copyright statement, and corrected a bug in the formal description.

7.3. Differences between draft-03 and draft-04

Draft 04 corrects a number of minor editing mistakes that were pointed out during the review of draft 03, issued on February 1.

7.4. Differences between draft-02 and draft-03

The main differences between draft-02, issued in January 22 1998, and draft 03 are:

* Introduced a discussion on endpoint types,

* Introduced a discussion of the connection set-up procedure, and of the role of connection parameters,

* Introduced a notation of the connection identifier within event names,

* Documented the extension procedure for the LocalConnectionOptions parameter and for the ConnectionParameters parameter,

* Introduced a three-way handshake procedure, using a ResponseAck parameter, in order to allow gateways to delete copies of old responses without waiting for a 30 seconds timer,

* Expanded the security section to include a discussion of "uncontrolled barge-in."

* Propsed a "create two connections" command, as an appendix.

7.5. Differences between draft-01 and draft-02

The main differences between draft-01, issued in November 1998, and draft 02 are:

* Added an ABNF description of the protocol.
* Specification of an EndpointConfiguration command,
* Addition of a "two endpoints" mode in the create connection command,
* Modification of the package wildcards from "$/$" to "*/all" at the Request of early implementors,
* Revision of some package definitions to better align with external specifications.
* Addition of a specification for the handling of "failover."
* Revision of the section on race conditions.

7.6. The making of MGCP from IPDC and SGCP

MGCP version 0.1 results from the fusion of the SGCP and IPDC proposals.

7.7. Changes between MGCP and initial versions of SGCP

MGCP version 0.1 (which subsumes SGCP version 1.2) introduces the following changes from SGCP version 1.1:

* Protocol name changed to MGCP.
* Introduce a formal wildcarding structure in the name of endpoints, inspired from IPDC, and detailed the usage of wildcard names in each operation.
* Naming scheme for events, introducing a package structure inspired from IPDC.
* New operations for audit endpoint, audit connection (requested by the Cablelabs) and restart (inspired from IPDC).
* New parameter to control the behavior of the notification request.
* Improved text on the detection and handling of race conditions.

* Syntax modification for event reporting, to incorporate package names.

* Definition of basic event packages (inspired from IPDC).

* Incorporation of mandatory and optional extension parameters, inspired by IPDC.

SGCP version 1.1 introduces the following changes from version SGCP 1.0:

* Extension parameters (X-??:)

* Error Code 511 (Unrecognized extension).

* All event codes can be used in RequestEvent, SignalRequest and ObservedEvent parameters.

* Error Code 512 (Not equipped to detect requested event).

* Error Code 513 (Not equipped to generate requested signal).

* Error Code 514 (Unrecognized announcement).

* Specific Endpoint-ID can be returned in creation commands.

* Changed the code for the ASDI display from "ad" to "asdi" to avoid conflict with the digits A and D.

* Changed the code for the answer tone from "at" to "aw" to avoid conflict with the digit A and the timer mark T

* Changed the code for the busy tone from "bt" to "bz" to avoid conflict with the digit B and the timer mark T

* Specified that the continuity tone value is "co" (CT was incorrectly used in several instances; CT conflicts with .)

* Changed the code for the dial tone from "dt" to "dl" to avoid conflict with the digit D and the timer mark T

* Added a code point for announcement requests.

* Added a code point for the "wink" event.

* Set the "octet received" code in the "Connection Parameters" to "OR" (was set to RO, but then "OR" was used throughout all examples.)

* Added a "data" mode.

* Added a description of SDP parameters for the network access mode (NAS).

* Added four flow diagrams for the network access mode.

* Incorporated numerous editing suggestions to make the description easier to understand. In particular, cleared the confusion between requests, queries, functions and commands.

* Defined the continuity test mode as specifying a dual-tone transponder, while the loopback mode can be used for a single tone test.

* Added event code "OC", operation completed.

* Added the specification of the "quarantine list", which clarifies the expected handling of events and notifications.

* Added the specification of a "wildcard delete" operation.

8. Security Considerations

Security issues are discussed in section 5.

9. Acknowledgements

We want to thank here the many reviewers who provided us with advice on the design of SGCP and then MGCP, notably Flemming Andreasen, Sankar Ardhanari, Francois Berard, David Auerbach, Bob Biskner, David Bukovinsky, Jerry Kamitses, Oren Kudevitzki, Barry Hoffner, Troy Morley, Dave Oran, Jeff Orwick, John Pickens, Lou Rubin, Chip Sharp, Paul Sijben, Kurt Steinbrenner, Joe Stone and Stuart Wray.

The version 0.1 of MGCP is heavily inspired by the "Internet Protocol Device Control" (IPDC) designed by the Technical Advisory Committee set up by Level 3 Communications. Whole sets of text have been retrieved from the IP Connection Control protocol, IP Media Control protocol, and IP Device Management. The authors wish to acknowledge the contribution to these protocols made by Ilya Akramovich, Bob Bell, Dan Brendes, Peter Chung, John Clark, Russ Dehlinger, Andrew Dugan, Isaac Elliott, Cary FitzGerald, Jan Gronski, Tom Hess, Geoff Jordan, Tony Lam, Shawn Lewis, Dave Mazik, Alan Mikhak, Pete O'Connell, Scott Pickett, Shyamal Prasad, Eric Presworsky, Paul Richards, Dale Skran, Louise Spergel, David Sprague, Raj Srinivasan, Tom Taylor and Michael Thomas.

10. References

* Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

* Schulzrinne, H., "RTP Profile for Audio and Video Conferences with Minimal Control", RFC 1890, January 1996.

* Handley, M and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

* Handley, M., "SAP - Session Announcement Protocol", Work in Progress.

* Handley, M., Schulzrinne, H. and E. Schooler, "Session Initiation Protocol (SIP)", RFC 2543, March 1999.

* Schulzrinne, H., Rao, A. and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

* ITU-T, Recommendation Q.761, "FUNCTIONAL DESCRIPTION OF THE ISDN USER PART OF SIGNALLING SYSTEM No. 7", (Malaga-Torremolinos, 1984; modified at Helsinki, 1993)

* ITU-T, Recommendation Q.762, "GENERAL FUNCTION OF MESSAGES AND SIGNALS OF THE ISDN USER PART OF SIGNALLING SYSTEM No. 7", (MalagaTorremolinos, 1984; modified at Helsinki, 1993)

* ITU-T, Recommendation H.323 (02/98), "PACKET-BASED MULTIMEDIA COMMUNICATIONS SYSTEMS."

* ITU-T, Recommendation H.225, "Call Signaling Protocols and Media Stream Packetization for Packet Based Multimedia Communications Systems."

* ITU-T, Recommendation H.245 (02/98), "CONTROL PROTOCOL FOR MULTIMEDIA COMMUNICATION."

* Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.

* Kent, S. and R. Atkinson, "IP Authentication Header", RFC 2402, November 1998.

* Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998.

* Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

11. Authors' Addresses

Mauricio Arango
RSL COM Latin America
6300 N.W. 5th Way, Suite 100
Ft. Lauderdale, FL 33309

Phone: (954) 492-0913
EMail: marango@rslcom.com

Andrew Dugan
Level3 Communications
1450 Infinite Drive
Louisville, CO 80027

Phone: (303)926 3123
EMail: andrew.dugan@l3.com

Isaac Elliott
Level3 Communications
1450 Infinite Drive
Louisville, CO 80027

Phone: (303)926 3123
EMail: ike.elliott@l3.com

Christian Huitema
Telcordia Technologies
MCC 1J236B
445 South Street
Morristown, NJ 07960
U.S.A.

Phone: +1 973-829-4266
EMail: huitema@research.telcordia.com

Scott Pickett
Vertical Networks
1148 East Arques Ave
Sunnyvale, CA 94086

Phone: (408) 523-9700 extension 200
EMail: ScottP@vertical.com

Further information is available on the SGCP web site:

http://www.argreenhouse.com/SGCP/

12. Appendix A: Proposed "MoveConnection" command

It has been proposed to create a new command, that would move an existing connection from one endpoint to another, on the same gateway. This command would be specially useful for handling certain call services, such as call forwarding between endpoints served by the same gateway.

```
      [SecondEndPointId,]
      [ConnectionId,]
      [LocalConnectionDescriptor]
       <--- ModifyConnection(CallId,
                             EndpointId,
                             ConnectionId,
                             SecondEndPointId,
                             [NotifiedEntity,]
                             [LocalConnectionOptions,]
                             [Mode,]
                             [RemoteConnectionDescriptor,]
                             [Encapsulated NotificationRequest,]
                             [Encapsulated EndpointConfiguration])
```

The parameters used are the same as in the ModifyConnection command, with the addition of a SecondEndpointId that identifies the endpoint towards which the connection is moved.

The EndpointId should be the fully qualified endpoint identifier of the endpoint on which the connection has been created. The local name shall not use the wildcard convention.

The SecondEndpointId shall be the endpoint identifier of the endpoint towards which the connection has been created. The "any of" wildcard convention can be used, but not the "all of" convention. If the SecondEndpointId parameter is unqualified, the gateway will choose a value, that will be returned to the call agent as a response parameter.

The command will result in the "move" of the existing connection to the second endpoint. Depending on gateway implementations, the connection identifier of the connection after the move may or may not be the same as the connection identifier before the move. If it is not the same, the new value is returned as a response parameter.

The intent of the command is to effect a local relocation of the connection, without having to modify such transmission parameters as IP addresses and port, and thus without forcing the call agent to signal the change of parameters to the remote gateway, at the other

end of the connection. However, gateway architectures may not always allow such transparent moves. For example, some architectures could allow specific IP addresses to different boards that handles specific group of endpoints. If for any reason the transmission parameters have to be changed as a result of the move, the new LocalConnectionDescriptor is returned as a response parameter.

The LocalConnectionOptions, Mode, and RemoteConnectionDescriptor, when present, are applied after the move.

The RequestedEvents, RequestIdentifier, DigitMap, SignalRequests, QuarantineHandling and DetectEvents parameters are optional. They can be used by the Call Agent to transmit a NotificationRequest that is executed simultaneously with the move of the connection. When these parameters are present, the NotificationRequest applies to the second endpoint.

When these parameters are present, the move and the NotificationRequests should be synchronized, which means that both should be accepted, or both refused. The NotifiedEntity parameter, if present, applies to both the ModifyConnection and the NotificationRequest command.

The command may carry an encapsulated EndpointConfiguration command, that will also apply to the second endpoint. When this command is present, the parameters of the EndpointConfiguration command are inserted after the normal parameters of the MoveConnection with the exception of the SecondEndpointId, which is not replicated. The EndpointConfiguration command may be encapsulated together with an encapsulated NotificationRequest command.

The encapsulated EndpointConfiguration command shares the fate of the MoveConnection command. If the MoveConnection is rejected, the EndpointConfiguration is not executed.

12.1. Proposed syntax modification

The only syntax modification necessary for the addition of the moveConnection command is the addition of the keyword MOVE to the authorized values in the MGCPVerb clause of the formal syntax.

13. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.