PacketCable™ Network-Based Call Signaling Protocol Specification

PKT-SP-EC-MGCP-I10-040402

ISSUED

Superseded

Notice

Document Status Sheet

Document Control Number:	PKT-SP-EC-MGCP-I10-040402			
Document Title:	PacketCable™ Network-Based Call Signaling Protocol Specification			
Revision History:	I01 — Released March 12, 1999 I02 — Released December 1, 1999 I03 — Released June 20, 2001 I04 — Released December 21, 2001 I05 — Released October 18, 2002 I06 — Released November 27, 2002 I07 — Released April 15, 2003 I08 — Released July 28, 2003 I09 — Released January 13, 2004 I10 — Released April 2, 2004			
Date:	April 2, 2004			
Status:	~~Work in Progress~~	~~Draft~~	Issued	~~Released~~
Distribution Restrictions:	~~Author Only~~	~~CL/Member~~	~~CL/ PacketCable/Vendor~~	Public

Key to Document Status Codes:

Work in Progress	An incomplete document, designed to guide discussion and generate feedback, that may include several alternative requirements for consideration.
Draft	A document in specification format considered largely complete, but lacking review by Members and vendors. Drafts are susceptible to substantial change during the review process.
Issued	A stable document, which has undergone rigorous member and vendor review and is suitable for product design and development, cross-vendor interoperability, and for certification testing.
Closed	A static document, reviewed, tested, validated, and closed to further engineering change requests to the specification through CableLabs.

Contents

List of Figures

List of Tables

This page intentionally left blank.

1 STATUS OF THIS DOCUMENT

This document is considered part of the PacketCable™ specification. The document is based on MGCP 1.0 [1], which is an IETF Informational RFC.

1.1 Specification Language

Throughout this document, the words that are used to define the significance of particular requirements are capitalized. These words are:

"MUST"	This word or the adjective "REQUIRED" means that the item is an absolute requirement of this specification.
"MUST NOT"	This phrase means that the item is an absolute prohibition of this specification.
"SHOULD"	This word or the adjective "RECOMMENDED" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
"SHOULD NOT"	This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or event useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
"MAY"	This word or the adjective "OPTIONAL" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

The legal/regulatory classification of IP-based voice communications provided over cable networks and otherwise, and the legal/regulatory obligations, if any, borne by providers of such voice communications, are not yet fully defined by appropriate legal and regulatory authorities. Nothing in this specification is addressed to, or intended to affect, those issues. In particular, while this document uses standard terms such as "call," "call signaling," "telephony," etc., it will be evident from this document that while a PacketCable network performs activities analogous to these PSTN functions, the manner by which it does so differs considerably from the manner in which they are performed in the PSTN by telecommunications carriers. These differences may be significant for legal/regulatory purposes.

2 SCOPE

This specification describes a profile of the Media Gateway Control Protocol (MGCP) for PacketCable embedded clients, which we will refer to as the PacketCable Network-based Call Signaling (NCS) protocol. MGCP is a call signaling protocol for use in a centralized call control architecture, and assumes relatively simple client devices. The call signaling protocol is one layer of the overall PacketCable suite of specifications and relies upon companion protocol specifications to provide complete end-to-end PacketCable functionality. The scope of NCS is currently only embedded Voice-Over-IP client devices in a PacketCable environment and the NCS profile has therefore simplified and in some cases modified the base MGCP 1.0 protocol accordingly. Support for video will be added in a later version of this document.

This document describes a PacketCable profile of an application programming interface (MGCI), and a corresponding protocol (MGCP) for controlling voice-over-IP (VoIP) embedded clients from external call control elements. The MGCP assumes a call control architecture where the call control "intelligence" is outside the gateways and is handled by external call control elements.

This document is based on the Media Gateway Control Protocol (MGCP) 1.0 RFC [1], which was the result of a merge of the IETF draft of Simple Gateway Control Protocol, and the IETF draft of IP Device Control (IPDC) family of protocols, as well as input generated by the PacketCable embedded client signaling team that developed this profile.

This document, which defines the PacketCable NCS Protocol specification, constitutes a document that is independent of MGCP in order to provide a stable reference document, while meeting current time-to-market demands for such a reference. It is the intent of this document to be as closely aligned with MGCP as possible for the PacketCable environment, in order to avoid developing multiple protocols to solve the same problem. This goal has been, and continues to be, pursued through cooperation with the authors of the MGCP specification. The NCS profile of MGCP, however, is strictly and solely defined by the contents of this document.

This NCS profile of MGCP, referred to as the Network-based Call Signaling Protocol 1.0, NCS 1.0, the NCS profile, or simply NCS in this document, has been modified from the MGCP 1.0 RFC in the following ways:

- *The NCS protocol only aims at supporting PacketCable-embedded clients:* The NCS protocol supports embedded clients as defined by PacketCable. Functionality present in the MGCP 1.0 protocol, which was superfluous to NCS, has been removed.
- *The NCS protocol contains extensions and modifications to MGCP:* PacketCable-specific requirements have been addressed in NCS, which has resulted in a couple of minor extensions and modifications to MGCP. However, the MGCP architecture, and all of the MGCP constructs relevant to embedded clients, are preserved in NCS.
- *The NCS protocol contains minor simplifications from MGCP 1.0:* Where several choices were available, and not necessarily needed for an embedded client in the PacketCable environment, some simplifications have been made for embedded-client implementations.

Although MGCP is not NCS, and NCS is not MGCP, the names MGCP and NCS will be used interchangeably in this document since this document is based on MGCP. Unless otherwise stated or inferred by context, the word MGCP shall be taken to mean the NCS profile of MGCP in this document.

The document is structured in the following main sections:

- The introduction presents the basic assumptions and the relation to other protocols such as H.323, RTSP, SAP, and SIP.
- The interface section presents a conceptual overview of the NCS profile, presenting the naming conventions, the usage of the session description protocol (SDP), and the procedures that compose NCS:
  - Notification Request,
  - Notify,
  - Create Connection,
  - Modify Connection,
  - Delete Connection,
  - Audit Endpoint,
  - Audit Connection, and
  - Restart In Progress.
- Each of the commands is presented as an example API with the name of the command, the parameters it can take and return, as well as the semantics of each of these. The actual encoding of the command and its parameters is shown in the protocol description section. The section concludes with a description of how reliability and race conditions are handled.
- The protocol description section presents the actual NCS encodings of the commands and parameters, which are based on simple text formats. The transmission procedure over UDP is specified as well.
- The security section presents the security of NCS.
- The appendices contain definitions of event packages, mode interactions, example command encodings, and a complete example call flow.

3 INTRODUCTION

This document describes the NCS profile of an application programming interface (MGCI) and a corresponding protocol (MGCP) for controlling embedded clients from external call control elements. An embedded client is a network element that provides:

- Two or more traditional analog (RJ11) access lines to a voice-over-IP (VoIP) network.
- Optionally, one or more video lines to a VoIP network.

Embedded clients may not be confined to residential use only. For example, they may be used in a business as well. Embedded clients are used for line-side access and, as such, are expected to have line-side equipment, e.g., analog access lines for conventional telephones associated with them, as opposed to trunk gateways.

The MGCP assumes a call control architecture where the call control "intelligence" is outside the gateways and handled by external call-control elements referred to as Call Agents. The MGCP assumes that these call-control elements, or Call Agents (CAs), will synchronize with each other to send coherent commands to the gateways under their control. The MGCP defined in this document does not define a mechanism for synchronizing Call Agents, although future PacketCable specifications may specify such mechanisms.

The MGCP assumes a connection model where the basic constructs are endpoints and connections. A gateway contains a collection of endpoints, which are sources, or sinks, of data and could be physical or virtual.

An example of a physical endpoint is 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 embedded client or an MTA. Embedded clients may optionally support video as well.

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 accomplished by software. However, the NCS profile of MGCP only addresses physical endpoints.

Connections are point-to-point. 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. The association is established by creating the connection as two halves; one on the origination endpoint, and one on the terminating endpoint.

Call Agents instruct the gateways to create connections between endpoints and to detect certain events, e.g., off-hook, and generate certain signals, e.g., ringing. It is strictly up to the Call Agent to specify how and when connections are made, between which endpoints they are made, as well as what events and signals are to be detected and generated on the endpoints. The gateway, thereby, becomes a simple device, without any call state, that receives general instructions from the Call Agent without any need to know about or even understand the concept of calls, call states, features, or feature interactions. When new services are introduced, customer profiles changed, etc., the changes are transparent to the gateway. The Call Agents implement the changes and generate the appropriate new mix of instructions to the gateways for the changes made. Whenever the gateway reboots, it will come up in a clean state and simply carry out the Call Agent's instructions as they are received.

3.1 Relation With H.323 Standards

The 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, which may or may not be distributed over several computer platforms, and a set of gateways. In an H.323 configuration, this distributed gateway system may interface on one side with one or more POTS lines, and on the other side with H.323 conformant systems, as illustrated in Figure 1 below:

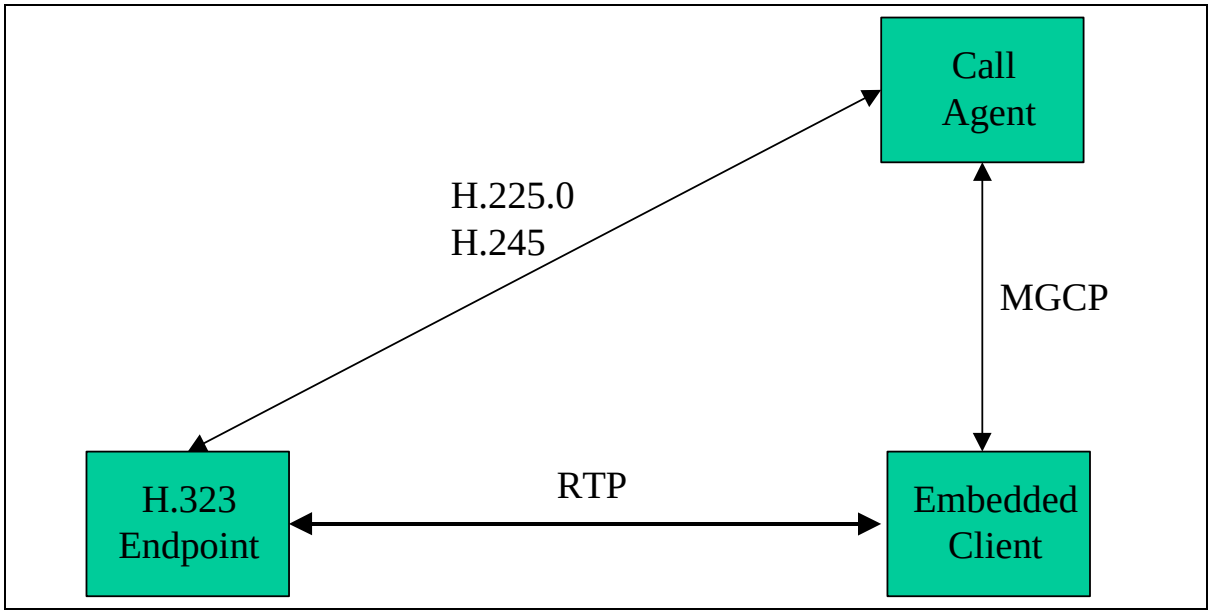

Figure 1 - Relation to H.323 Standards

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.

3.2 Relation With IETF Standards

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

- the session description protocol (SDP) [4],
- the session announcement protocol (SAP) [5],
- the session initiation protocol (SIP [6],
- the real-time streaming protocol (RTSP) [7].

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 and RTCP).

The distributed gateway systems and MGCP will enable PSTN voice communication and embedded client users to access sessions set up using SAP, SIP, or RTSP. The Call Agent provides for signaling conversion, as illustrated in Figure 2 below:

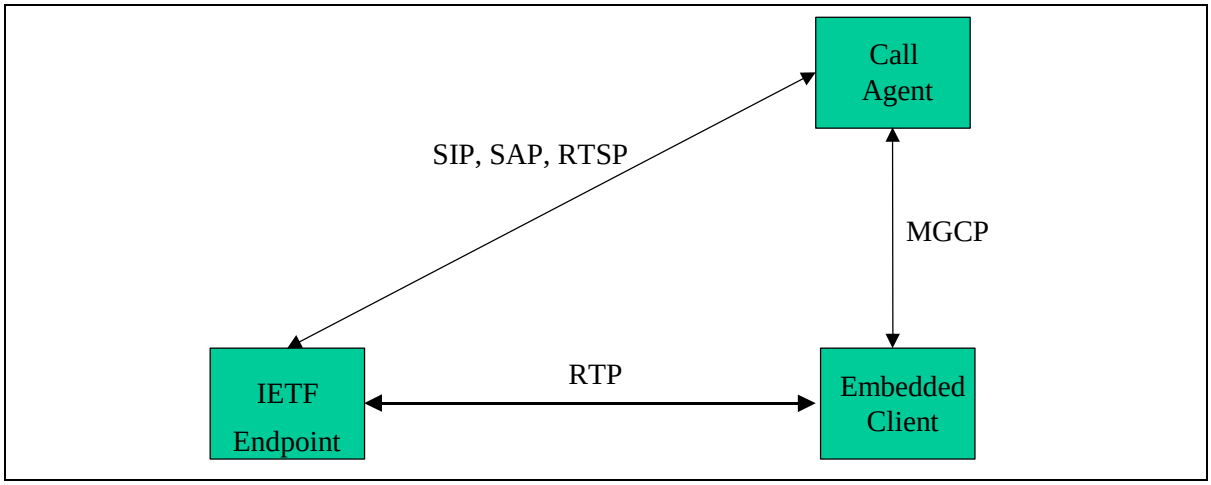

Figure 2 - Relation to IETF Standards

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.

3.3 Relation to RFC 3435 and ABNF Grammar

RFC 3435 includes a formal description of the MGCP protocol syntax following the "Augmented BNF for Syntax Specifications". This formal description is referenced by developers for the creation of interoperable devices. A copy of the MGCP protocol syntax, annotated and edited to indicate its applicability to PacketCable specifications, is provided in Appendix G. Adherence to these guidelines can improve interoperability by minimizing failures caused by different interpretations of syntax and grammar.

4 MEDIA GATEWAY CONTROL INTERFACE (MGCI)

MGCI functions provide for connection control, endpoint control, auditing, and status reporting. They each use the same system model and the same naming conventions.

4.1 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.

4.1.1 Endpoint Names

Endpoint names, also known as (a.k.a.), endpoint identifiers, have two components, both of which are defined to be case insensitive here:

- the domain-name of the gateway managing the endpoint
- a local endpoint name within that gateway

Endpoint names will be of the form

```
local-endpoint-name@domain-name
```

where domain-name is an absolute domain-name as defined in [24] and includes a host portion, thus an example domain-name could be:

```
MyEmbeddedClient.cablelabs.com
```

Also, `domain-name` may be an IPv4 address in dotted decimal form represented as a text-string and surrounded by a left and a right square bracket ("[" and "]") as in "[128.96.41.1]" — please consult [22] for details. However, use of IP addresses is generally discouraged.

Embedded clients may have one or more endpoints (e.g., one for each RJ11 jack for black phones) associated with them, and each of the endpoints is identified by a separate local endpoint name. Just like the domain-name, the local endpoint name is case insensitive. Associated with the local endpoint name is an endpoint-type, which defines the type of the endpoint, such as analog phone or video phone. The endpoint-type can be derived from the local endpoint name. The local endpoint name is a hierarchical name, where the least specific component of the name is the leftmost term, and the most specific component is the rightmost term. More formally, the local endpoint name MUST adhere to the following naming rules:

- The individual terms of the local endpoint name must be separated by a single slash ("/", ASCII 2F hex).
- The individual terms are ASCII character strings composed of letters, digits or other printable characters, with the exception of characters used as delimiters in endpoint-names ("/", "@"), characters used for wildcarding ("*", "$"), and white space characters.
- Wild carding is represented either by an asterisk ("*") or a dollar sign ("$") for the terms of the naming path which are to be wild-carded. Thus, if the full local endpoint name looks like:

  term1/term2/term3

  and one of the terms of the local endpoint name is wild-carded, then the local endpoint name looks like this:

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

In each of the examples, a dollar sign could have appeared instead of the asterisk.

- Wild-carding is only allowed from the right, thus if a term is wild-carded, then all terms to the right of that term must be wild-carded as well.
- In cases where mixed dollar sign and asterisk wild-cards are used, dollar-signs are only allowed from the right, thus if a term had a dollar sign wild-card, all terms to the right of that term must also contain dollar sign wild-cards.
- A term represented by an asterisk is to be interpreted as:
  "use all values of this term known within the scope of the embedded client in question".
- 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 embedded client in question".
- Each endpoint-type may specify additional detail in the naming rules for that endpoint-type, however such rules must not be in conflict with the above.

It should be noted that different endpoint-types or even different sub-terms, e.g., "lines", within the same endpoint-type will result in two different local endpoint names. Consequently, each "line" will be treated as a separate endpoint. Note that since the domain name portion is part of the endpoint identifier, different forms or different values referring to the same entity are not freely interchangeable. Following a restart the most recently supplied form and value MUST always be used.

4.1.1.1 Embedded Client Endpoint Names

Endpoints in embedded clients MUST support the additional naming conventions specified in this section.

Embedded clients will support the following two endpoint-types:

- Analog telephone: The Analog Telephone is represented as an analog access line (aaln).This is basically the equivalent of an analog telephone line as known in the PSTN (see [18]).
- Video: The details of the video device-type will be provided in a future version of this document.

4.1.1.1.1 Analog Access Line Endpoints

In addition to the naming conventions specified above, local endpoint names for endpoints of type "analog access line" (aaln) for embedded clients MUST adhere to the following:

- Local endpoint names contain at least one and, at most, two terms.
- Term1 MUST be the term "aaln" or a wildcard character. It should be noted that the use of a wildcard character for term1 can refer to any or all endpoint-types in the embedded client regardless of their type. Use of this feature is generally expected to be for administrative purposes, e.g., auditing or restart.
- Term2 MUST be a number from one to the number of analog access lines supported by the embedded client in question. The number thus identifies a specific analog access line on the embedded client.
- If a local endpoint name is composed of only one term, that term will be term1:
  - If term1 is not a wildcard character, the wildcard character dollar sign (referring to "any one") is then assumed for term2, i.e., "aaln" is equivalent to "aaln/$".
  - If term1 is a wildcard character, the same wildcard character is then assumed for term2, i.e., "*" and "$" is equivalent to respectively "*/*" and "$/$".

Example analog access line local endpoint names could thus be:

`aaln/1`	The first analog access line on the embedded client in question.
`aaln/2`	The second analog access line on the embedded client in question.
`aaln/$`	Any analog access line on the embedded client in question.
`aaln/*`	All analog access lines on the embedded client in question.
`*`	All endpoints (regardless of endpoint-type) on the embedded client in question.

The provisioning/(auto)configuration process is responsible for obtaining and providing information about how many endpoints an embedded client has, as well as the endpoint-type of each endpoint. Although they are logically different, it should be noted that the *endpoint-type* can be derived from the local portion of the endpoint name.

4.1.1.1.2 Video Endpoints

Details on video endpoints will be provided in a future version of this document.

4.1.2 Call Names

Calls are identified by unique identifiers, independent of the underlying platforms or agents. Call identifiers are hexadecimal strings, which are created by the Call Agent. Call identifiers with a maximum length of 32 characters MUST be supported.

At a minimum, call identifiers MUST be unique within the collection of call agents that control the same gateways. However, the coordination of these call identifiers between Call Agents is outside the scope of this document. When a Call Agent builds several connections that pertain to the same call, either on the same gateway or in different gateways, these connections may all be linked to the same call through the call identifier. This identifier then can be used by accounting or management procedures, which are outside the scope of MGCP.

4.1.3 Connection Names

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. Connection identifiers are treated in MGCP as hexadecimal strings. The gateway MUST ensure that a proper waiting period, at least three minutes, elapses between the end of a connection that used this identifier and its use in a new connection for the same endpoint. Connection Names with a maximum length of 32 characters MUST be supported.

4.1.4 Names of Call Agents and Other Entities

The Media Gateway Control Protocol has been designed for enhanced network reliability to allow implementation of redundant Call Agents. This means that there is no fixed binding between entities and hardware platforms or network interfaces.

Call Agent names consist of two parts, similar to endpoint names. The local portion of the name does not exhibit any internal structure. An example Call Agent name is:

```
ca1@ca.whatever.net
```

Reliability is provided by the following precautions:

- Entities such as embedded clients 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 cannot be forwarded to one of the network addresses, implementations MUST 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 (DNS). Call Agents and gateways MUST keep track of the record's time-to-live read from the DNS. They MUST 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 failover 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" MUST 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[1], 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."

Section 4.4 contains a more detailed description of reliability and failover.

4.1.5 Digit Maps

The Call Agent can ask the gateway to collect digits dialed by the user. This facility is intended to be used for analog access lines with residential gateways to collect the numbers that a user dials; it may also be used to collect access codes, credit card numbers, and other numbers requested by call control services. Endpoints MUST support Digit Maps as defined in this section.

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

The problem with this accumulation approach, however, is that it is difficult 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 numbers illustrated in Figure 3 below:

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

Figure 3 - Dialed Digits - Examples

The solution to this problem is to load the gateway with a digit map that corresponds 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)
```

[1] This could happen as a result of specifying an empty NotifiedEntity parameter.

The formal syntax of the digit map is described by the following BNF notation:

```
Digit           ::= "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
Timer           ::= "T" | "t" -- matches the detection of a timer
Letter          ::= Digit | Timer | "#" | "*" | "A" | "a" | "B" | "b" | "C" | "c" | "D" | "d"
Range           ::= "X" | "x"           -- matches any digit
                  | "[" Letters "]"     -- matches any of the specified letters
Letters         ::= Subrange | Subrange Letters
Subrange        ::= Letter              -- matches the specified letter
                  | Digit "-" Digit     -- matches any digit between first and last
Position        ::= Letter | Range
StringElement   ::= Position            -- matches an occurrence of the position
                  | Position "."        -- matches an arbitrary number of occurrences
                                        -- of the position, including 0
String          ::= StringElement | StringElement String
StringList      ::= String | String "|" StringList
DigitMap        ::= String | "(" StringList ")"
```

A DigitMap, according to this syntax, is defined either by a (case insensitive) "string" or by a "list of strings" over which the gateway will attempt to find a shortest possible match. Regardless of the above syntax, a timer is currently only allowed if it appears in the last position in a string[2]. Each string in the list is an alternate numbering scheme. A gateway that detects digits, letters, or timers will:

1. Add the event parameter code for the digit, letter, or timer, as a token to the end of the "current dial string" internal state variable.
2. Apply the "current dial string" to the digit map table, attempting a match to all expressions in the Digit Map.
3. If the result is under-qualified (partially matches at least one entry in the digit map and doesn't completely match another entry), do nothing further.

If the result matches an entry, or is over-qualified (i.e., no further digits could possibly produce a match), send the current dial string to the Call Agent[3] and clear the "current dial string".." A match, in this specification, can be either a "perfect match," exactly matching one of the specified alternatives, or an "impossible match", which occurs when the dial string does not match any of the alternatives. Unexpected timers, for example, can cause impossible matches. Both perfect matches and impossible matches trigger notification of the accumulated digits (which may include other events).

Timer T is a digit input timer that can be used in two ways:

- When timer T is used with a digit map[4], 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 inter-digit timer when overlap sending is used.

[2] For instance, "123T" and "123[1-2T5]" satisfy that rule, but "12T3" does not.

[3] The list of digits may include other events as well. See Section 4.4.3.1

[4] Technically speaking with the "accumulate according to digit map" action.

When used with a digit map, timer T takes on one of two values, T_{par} or T_{crit}. 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_{par}, corresponding to partial dial timing. If a timer is all that is required to produce a match, timer T takes on the value T_{crit} corresponding to critical timing. When timer T is used without a digit map, timer T takes on the value T_{crit}. The default value for T_{par} is 16 seconds and the default value for T_{crit} is 4 seconds. The provisioning process may alter both of these.

Appendix A contains additional detail and an example on use of timer T.

The end-points MUST support at least 2048 Bytes of digit map on all of the telephony interfaces.

Digit maps can be provided to the gateway by the Call Agent, whenever the Call Agent instructs the gateway to listen for digits. Digit Maps, when provided by the Call Agent, MUST be as defined in this section.

4.1.6 Events and Signals

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. A Call Agent also 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. A package is a collection of events and signals supported by a particular endpoint-type. 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, and each endpoint-type has a default package with which it is associated.

Event names consist of a package name and an event code and, since each package defines a separate namespace, the same event codes may be used in different packages. Package names and event codes are case insensitive strings of letters, digits, and hyphens, with the restriction that hyphens MUST NOT be the first or last character in a name. Some event codes may need to be parameterized with additional data, which is accomplished by adding the parameters between a set of parentheses. The package name is separated from the event code by a slash ("/"). The package name may be excluded from the event name, in which case the default package name for the endpoint-type in question is assumed. For example, for an analog access line with the line package (package name "L") being the default package, the following two event names are considered 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 the packages for embedded client types listed in Table 1.

Table 1 - Packages for Embedded Client End-point Types

Endpoint-Type	Package	Package Name	Default Package
Analog Access Line	Line	L	Yes
Video	For further study	For further study	For further study

Additional package names and event codes may be defined by and/or registered with PacketCable. Any change to the packages defined in this document MUST result in a change of the package name, or a change in the NCS profile version number, or possibly both.

Each package MUST have a package definition, which MUST define the name of the package, and the definition of each event belonging to the package. The event definition MUST include the precise name of the event, i.e., the event code, a plain text definition of the event and, when appropriate, the precise definition of the corresponding signals, for example the exact frequencies of audio signals such as dial-tone or DTMF tones. Events must further specify if they are persistent (e.g., off-hook, see Section 4.3.1) and if they contain auditable event-states (e.g., off-hook, see Section 4.3.8.1). Signals MUST also have their type defined (On/Off, Time-Out, or Brief), and Time-Out signals MUST have a default time-out value defined – see Section 4.3.1.

In addition to PacketCable packages, implementers MAY gain experience by defining experimental packages. The package name of experimental packages MUST begin with the two characters "x-" or "X-"; PacketCable MUST NOT register package names that start with these two characters. An embedded client that receives a command referring to an unsupported package MUST return an error (error code 518 – unsupported package).

Package names and event codes support one wild-card notation each. The wildcard character "*" (asterisk) can be used to refer to all packages supported by the endpoint in question, and the event code "all" to refer to all events in the package in question. For example:

L/all refers to all events in the line package for an analog access line.

*/all for an analog access line; refers to all packages and all events in those packages supported by the endpoint in question.

Consequently, the package name "*" MUST NOT be assigned to a package, and the event code "all" MUST NOT be used in any package.

Events and signals are by default detected and generated on endpoints, however some events and signals may be detected and generated on connections in addition to or instead of on an endpoint. For example, endpoints may be asked to provide a ringback tone on a connection. In order for an event or signal to be able to be detected or generated on a connection, the definition of the event/signal MUST explicitly define that the event/signal can be detected or generated on a connection.

When a signal 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:

```
L/rt@0A3F58
```

Should the connection be deleted while an event or signal is being detected or applied on it, that particular event detection or signal generation simply MUST stop. Depending on the signal, the endpoint SHOULD generate a failure, i.e., if the signal type is TO, the operation failure event will be generated, since the connection associated with the signal was deleted prior to the signal timing out. Notification action associated with reporting the failure must conform to the notify operations as defined for Notification Request handling (section 4.3.1).

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

```
L/ma@*
```

However, when the event is actually observed, the gateway MUST include the name of the specific connection on which the event occurred. The wildcard character "$" (dollar sign) can be used to denote "the current connection". This convention MUST NOT be used unless the event notification request is "encapsulated" within a CreateConnection or ModifyConnection command. When the convention is used, the gateway MUST generate or detect the event on the connection that is currently being created or modified. An example of this convention is:

```
L/rt@$
```

When processing a command using the "current connection" wildcard, the "$" wildcard character MUST be expanded by the gateway to the value of the current connection. If a subsequent command either explicitly (e.g., by auditing) or implicitly (e.g., by persistence) refers to such an event, the expanded value MUST be

used by the gateway. In other words, the "current connection" wildcard is expanded once, which is at the initial processing of the command in which it was explicitly included.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 for the affected endpoint(s). However, the use of the "all packages" and "all events" wildcards is strongly discouraged.

4.2 SDP Use

The Call Agent uses the MGCP to provide the gateways with the description of connection parameters such as IP addresses, UDP port, and RTP profiles. Except where otherwise noted or implied in this specification, SDP descriptions MUST follow the conventions delineated in the session description protocol (SDP), which is now an IETF-Proposed Standard documented in [4]. In addition, all call agents and gateways MUST ignore any SDP parameters, attributes, or fields that are not understood by the call agent or gateway.

SDP allows for description of multimedia conferences. The NCS profile will only support the setting of audio and video connections using the media types "audio" and "video". Currently, only "audio" connections have been specified.

4.3 Gateway Control Functions

This section describes the commands of the MGCP in the form of a remote procedure call (RPC) like API, which we will refer to as the media gateway control interface (MGCI). An MGCI function is defined for each MGCP command, where the MGCI function takes and returns the same parameters as the corresponding MGCP command. The functions shown in this section provide a high-level description of the operation of MGCP and describe an example of an RPC-like API that MAY be used for an implementation of MGCP. Although the MGCI API is merely an example API, the semantic behavior defined by MGCI is an integral part of the specification, and all implementations MUST conform to the semantics specified for MGCI. The actual MGCP messages exchanged, including the message formats and encodings used are defined in the protocol section (Section 5). Embedded clients MUST implement those exactly as specified.

The MGCI service consists of connection handling and endpoint handling commands. The following is an overview of the commands:

- 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 on the specified endpoint.
- 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. In some circumstances, the DeleteConnection command also can 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 is generally desirable, e.g., information about the status of the embedded client. 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 endpoint, 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. Currently, an endpoint is limited to a specific analog access line within an embedded client.

- Connections are grouped into "calls". Several connections, that may or may not belong to the same call, can terminate in the same endpoint. Flow of media on each connection is controlled by a "mode" parameter, which can be set to "send only" (sendonly), "receive only" (recvonly), "send/receive" (sendrecv), "conference" (confrnce), "inactive" (inactive), "replicate" (replcate), "network loopback" (netwloop) or "network continuity test" (netwtest). The "mode" parameter determines if media packets can be sent and/or received on the connection. RTCP is independent of the connection mode; for more details see the Codec specification [19].

Handling of media received from the endpoint is determined by the mode parameter:

- Media originating from the endpoint will be sent on all the connections for that endpoint whose mode is either "send only", "send/receive", "conference", or "replicate".

Handling of media received on these connections is also determined by the mode parameters

- Media received in data packets through connections in "inactive", "sendonly", or "replicate" mode is discarded.
- Media received in data packets through connections in "receive only", "conference", or "send/receive" mode is mixed together and then sent to the endpoint.
- In addition to being sent to the endpoint, media received in data packets through connections in "conference" mode is replicated to all the other connections for the endpoint whose mode is "conference". The details of this forwarding, e.g., RTP translator or mixer, etc., are outside the scope of this document.
- In addition to the media received from the endpoint, media sent to the endpoint is mixed and transmitted over all the other connections for the endpoint whose mode is "replicate". This SHOULD include media generated by signals applied to the endpoint.
- Media received in data packets through connections in "network loopback" or "network continuity test" mode will be sent back on the connection as described below.

If the mode is set to "network loopback," the audio signals received from the connection will be echoed back on the same connection. The "network loopback" mode SHOULD simply operate as an RTP packet reflector.

The "network continuity test" mode is used for continuity checking across the IP network. An endpoint-type specific signal is sent to the endpoints over the IP network, and the endpoint is then supposed to echo the signal over the IP network after passing it through the gateway's internal equipment to verify proper operation. The signal MUST go through internal decoding and re-encoding prior to being passed back. For analog access lines, the signal will be an audio signal, and the signal MUST NOT be passed on to a telephone connected to the analog access line, regardless of the current hook-state of that handset, i.e., on-hook or off-hook.

New and existing connections for the endpoint MUST NOT be affected by connections placed in "network loopback" or "network continuity test" mode. However, local resource constraints may limit the number of new connections that can be made.

The "replicate" mode MUST at a minimum support replicating the stream from the endpoint and one other connection regardless of the encoding method used for that other connection. The "replicate" connection is

however only REQUIRED to support a resulting media stream in G.711 encoding[5]. Support of the "conference" mode is optional; all other connection modes must be supported. Please refer to Appendix B for illustrations of mode interactions.

4.3.1 NotificationRequest

The NotificationRequest command is used to request the gateway to send a notification upon the occurrence of specified events in an endpoint. For example, a notification may be requested when tones associated with fax communication are detected on the endpoint. The entity receiving this notification, usually the Call Agent, may then decide that a different type of encoding should be used on the connections bound to this endpoint and instruct the gateway accordingly[6].

ReturnCode
← NotificationRequest(EndpointId
[, NotifiedEntity]
[, RequestedEvents]
, RequestIdentifier
[, DigitMap]
[, SignalRequests]
[, QuarantineHandling]
[, DetectEvents])

EndpointId is the identifier for the endpoint(s) in the gateway where NotificationRequest executes. The EndpointId MUST follow the rules for endpoint names specified in Section 4.1.1. The "any of" wildcard MUST NOT be used. An embedded client that receives a NotificationRequest with the "any of" wildcard convention MUST return an error (the error returned SHOULD be error code 500 - the transaction could not be executed because the endpoint is unknown) in response. The "all of" wildcard MUST be supported for NotificationRequests with each of RequestedEvents, SignalsRequest, DigitMap, and DetectEvents being either empty or omitted. For simplicity, some gateways may choose to not support the "all-of" wildcard for NotificationRequests where one or more of these parameters is neither empty nor omitted. Such gateways shall respond with error code 503 if they receive an "all-of" wildcarded NotificationRequest which they are unable to process for this reason.

NotifiedEntity is an optional parameter that specifies a new "notified entity" for the endpoint. When used, the entire Call Agent name MUST be specified which includes both the local name and domain name - even if a bracketed IP address[7] is used for the domain name. See Sections 4.1.1 and 4.1.4 for more information. If, however, only the domain name is provided, the MTA SHOULD use the domain name as the Call Agent ID.

RequestIdentifier is used to correlate this request with the notification it may trigger. It will be repeated in the corresponding Notify command.

SignalRequests is a parameter that contains the set of signals that the gateway is asked to apply. Unless otherwise specified, signals are applied to the endpoint, however some signals can be applied to a connection. The following are examples of signals[8]:

- Ringing,
- Busy tone,

[5] The "replicate" connection can, e.g., be used to support "busy line verification" with minimal resource impact on the embedded client.

[6] The new instruction would be a ModifyConnection command.

[7] The use of an IP address in the NotifiedEntity is not permitted by the security specification [21]. When implementing the security specification [21], an absolute domain name (including the hostname) must be used.

[8] Please refer to Appendix A for a complete list of signals.

- Call waiting tone,
- Off hook warning tone,
- Ringback tones on a connection.

Signals are divided into different types depending upon their behavior:

- On/off (OO) Once applied, these signals last until they are turned off. This can only happen as the result of a new SignalRequests where the signal is turned off (see later). Signals of type OO are defined to be idempotent, thus multiple requests to turn a given OO signal on (or off) are perfectly valid and MUST NOT result in any errors. An On/Off signal could be a visual message waiting indicator (VMWI). Once turned on, an OO signal MUST NOT be turned off until explicitly instructed to by the Call Agent; OO signals will be off following restart of the endpoint. A missing or empty SignalRequest parameter or a SignalRequest parameter that omits a particular signal is not an explicit instruction for an OO signal; it will not change the MTA state for an OO signal.
- Time-out (TO) Once applied, these signals last until they are either cancelled (by the occurrence of an event or by not being included in a subsequent [possibly empty] list of signals), or a signal-specific period of time has elapsed. A signal that times out will generate an "operation complete" event (please see Appendix A for further definition of this event). A TO signal could be "ringback" timing out after 180 seconds. If an event occurs prior to the 180 seconds, the signal will, by default, be stopped[9]. If the signal is not stopped, the signal will time out, stop and generate an "operation complete" event, about which the Call Agent may or may not have requested to be notified. If the Call Agent has asked for the "operation complete" event to be notified, the "operation complete" event sent to the Call Agent will include the name(s) of the signal(s) that timed out[10]. Signal(s) generated on a connection will include the name of that connection. Time-out signals have a default time-out value defined for them, which may be altered by the provisioning process. Also, the time-out period may be provided as a parameter to the signal. A value of zero indicates that the time-out period is infinite. A TO signal that fails after being started, but before having generated on "operation complete" event will generate an "operation failure" event, which will include the name(s) of the signal(s), that time out[10].
- Brief (BR) The duration of these signals is so short that they stop on their own. If a signal stopping event occurs, or a new SignalRequests is applied, a currently active BR signal will not stop. However, any pending BR signals not yet applied will be cancelled. A brief tone could be a DTMF digit. If the DTMF digit "1" is currently being played, and a signal stopping event occurs, the "1" would finish playing.

Signals are, by default, applied to endpoints. If a signal applied to an endpoint results in the generation of a media stream (audio, video, etc.), the media stream MUST NOT be forwarded on any connection associated with that endpoint, regardless of the mode of the connection. For example, if a call-waiting tone is applied to an endpoint involved in an active call, only the party using the endpoint in question will hear the call-waiting tone. However, individual signals may define a different behavior.

When a signal is applied to a connection that has received a RemoteConnectionDescriptor (see Section 4.3.3), the media stream generated by that signal MUST be forwarded on the connection regardless of the current mode of the connection. If a RemoteConnectionDescriptor has not been received, the gateway MUST return an error (error code 527 – missing RemoteConnectionDescriptor).

When a (possibly empty) list of signal(s) is supplied, this list completely replaces the current list of active time-out signals. Currently active time-out signals that are not provided in the new list MUST be stopped and the new signal(s) provided will now become active. Currently active time-out signals that are provided in the new list of signals MUST remain active without interruption, thus the timer for such time-out signals will not be affected. Consequently, there is currently no way to restart the timer for a currently active time-out signal without turning the signal off first. If the time-out signal is parameterized, the original set of parameters MUST remain in effect, regardless of what values are provided subsequently. A given signal MUST NOT

[9] The "Keep signal(s) active" action may override this behavior.

[10] If parameters were passed to the signal, the parameters will not be reported.

appear more than once in a SignalRequests. The omission of the SignalRequests parameter is interpreted as an empty SignalRequests list.

The currently defined signals can be found in Appendix A.

RequestedEvents is a list of events that the gateway is requested to detect on the endpoint. Unless otherwise specified, events are detected on the endpoint, however some events can be detected on a connection. Examples of events are:

- fax tones,
- modem tones,
- on-hook transition (occurring in classic telephone sets when the user hangs up the handset),
- off-hook transition (occurring in classic telephone sets when the user lifts the handset),
- flash hook (occurring in classic telephone sets when the user briefly presses the hook that holds the handset),
- DTMF digits (or pulse digits).

The currently defined events can be found in Appendix A.

To each event is associated one or more **actions** that define the action that the gateway MUST take when the event in question occurs. The possible actions are:

- Notify the event immediately, together with the accumulated list of observed events,
- Accumulate the event,
- Accumulate according to Digit Map,
- Ignore the event, ,
- Keep Signal(s) active,
- Embedded NotificationRequest,
- Embedded ModifyConnection.

Two sets of requested events will be detected by the endpoint: persistent and non-persistent.

Persistent events are always detected on an endpoint. 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[11]. 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[12]. Persistent events are identified as such through their definition – see Appendix A.

Non-persistent events are those events that have to be 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 omission of the RequestedEvents parameter is interpreted as an empty RequestedEvents list.

[11] Thus the RequestIdentifier will be the RequestIdentifier of the current NotificationRequest.

[12] Normally, if a request to look for, e.g., off-hook, is made, the request is only successful if the phone is not already off-hook.

More than one action can be specified for an event, although a given action can not appear more than once for a given event. Table 2 below specifies the legal combinations of actions:

Table 2 - Legal Combinations of Events and Actions

	Notify	Accumulate	Accumulate according to digit map	Ignore	Keep Signal(s) Active	Embedded NotificationRequest	Embedded ModifyConnection
Notify	-	-	-	-	✓	✓	✓
Accumulate	-	-	-	-	✓	✓	✓
Accumulate according to digit map	-	-	-	-	✓	-	✓
Ignore	-	-	-	-	✓	-	✓
Keep Signal(s) active	✓	✓	✓	✓	-	✓	✓
Embedded NotificationRequest	✓	✓	-	-	✓	-	✓
Embedded ModifyConnection	✓	✓	✓	✓	✓	✓	-

Note: The “Embedded Notification Request” action can only be combined with the “Notify” action, if the gateway is allowed to issue more than one Notify command per Notification request.

If a client receives a request with an invalid action or illegal combination of actions, it MUST return an error to the Call Agent (error code 523–unknown or illegal combination of actions).

When multiple actions are specified, e.g., “Keep signal(s) active” and “Notify”, the individual actions are assumed to occur simultaneously.

The Call Agent can send a NotificationRequest with an empty RequestedEvents list to the gateway. The Call Agent can do so, for example, to an embedded client when it does not want to collect any more DTMF digits. However, persistent events will still be detected and notified.

DigitMap is an optional parameter that allows the Call Agent to provision the endpoint with a digit map according to which digits will be accumulated when the Call Agent provides a RequestedEvents parameter with the action “accumulate according to digit map” for that endpoint. The digit map provided is persistent and, therefore, need not be provided whenever a request to “accumulate according to digit map” is made, however Call Agents can provide a digit map at any time. A digit map MUST be provided for the endpoint no later than with the first request to “accumulate according to digit map”. If the gateway is requested to “accumulate according to digit map” and the gateway currently does not have a digit map for the endpoint in question, the gateway MUST return an error (error code 519 – endpoint does not have a digit map).

Each endpoint has a variable called the “current dial string” in which digits are collected for matching with the digit map, as specified in Section 4.1.5. Whenever a Notify is sent or a NotificationRequest is to be processed, the “current dial string” is initialized to a null string. The digits to be processed may now either be detected as input, or they may be retrieved from an event input holding area known as the “quarantine buffer”. Please see Section 4.4.3.1 for further details.

The signals being applied by the SignalRequests is synchronized with the collection of events specified or implied in the RequestedEvents parameter, except if overridden by the "Keep signal(s) active" action. For example, if the NotificationRequest mandates a "ringing" signal and the event request asks to look for an "off-hook" event, the ringing shall, by default, stop as soon as the gateway detects an off-hook event. If the event request did not ask to look for an "off-hook" event, the ringing would stop anyway since off-hook is a persistent event and therefore implied in the RequestedEvents parameter. The formal definition is that the generation of all "Time Out" signals MUST stop as soon as one of the requested events is detected, unless the "Keep signal(s) active" action is associated to the specified event. In the case of the action "accumulate according to digit map", the default behavior would be to stop all active time-out signals when the first digit[13] is accumulated—it is irrelevant to this synchronization if the accumulated digit results in a match, mismatch, or partial matching to the digit map.

If it is desired that time-out signal(s) continue when a looked-for event occurs, the "Keep Signal(s) Active" action can be used. This action has the effect of keeping all currently active time-out signal(s) active, thereby negating the default stopping of time-out signals upon the event's occurrence.

If signal(s) are desired to start when a looked-for event 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, RequestIdentifier, QuarantineHandling and DetectEvents. When the "Embedded NotificationRequest" is activated, the "current dial string" will be cleared; however the list of observed events and the quarantine buffer will be unaffected (if combined with a Notify, the Notify will clear the ObservedEvents list though – see Section 4.4.3.1). Note, that the Embedded NotificationRequest action does not accumulate the triggering event, however it can be combined with the Accumulate action to achieve that. NCS implementations MUST be able to support at least one level of embedding. An embedded NotificationRequest that respects this limitation MUST NOT contain another Embedded NotificationRequest.

The embedded NotificationRequest action allows the Call Agent to set up a "mini-script" to be processed by the gateway immediately following the detection of the associated event. Any SignalRequests specified in the embedded NotificationRequest will start immediately. Considerable care must be taken to prevent discrepancies between the Call Agent and the gateway. However, long-term discrepancies should not occur as new SignalRequests completely replaces the old list of active time-out signals, and BR-type signals always stop on their own. Limiting the number of On/Off-type signals is encouraged. It is considered good practice for a Call Agent to occasionally turn on all On/Off signals that should be on, and turn off all On/Off signals that should be off.

If connection modes are desired to be changed when a looked-for event occurs, the "Embedded ModifyConnection" action can be used. The embedded ModifyConnection may include a list of connection mode changes each consisting of the mode change and the affected connection-id. The wildcard "$" can be used to denote "the current connection", however this notation MUST NOT be used outside a connection handling command – the wildcard refers to the connection in question for the connection handling command.

The embedded ModifyConnection action allows the Call Agent to instruct the endpoint to change the connection mode of one or more connections immediately following the detection of the associated event. Each of connection mode changes work similarly to a corresponding ModifyConnection command[14]. When a list of connection mode changes is supplied, the connection mode changes MUST be applied one at a time in left-to-right order. When all the connection mode changes have finished, an "operation complete" event parameterized with the name of the completed action will be generated (see Appendix A for details). Should any of the connection mode changes fail, an "operation failure" event parameterized with the name of the failed action and connection mode change will be generated (see Appendix A for details); the rest of the connection mode changes MUST NOT be attempted, and the previous successful connection mode changes in the list MUST remain effective.

[13] Digit as defined in digit maps, i.e., including asterisk, timer, etc.

[14] Thus, if, e.g., D-QoS is used on the connection, the default D-QoS action will still be taken when the embedded ModifyConnection action is carried out.

Finally, the Ignore action can be used to ignore an event, e.g., to prevent a persistent event from being notified. However, the synchronization between the event and an active signal will still occur by default.

Section 4.4.3.1 contains additional details on the semantics of event detection and reporting. The reader is encouraged to study it carefully.

The specific definition of actions that are requested via these SignalRequests (e.g., the duration of and frequency of a DTMF digit) is outside the scope of the core NCS Specification. This definition may vary from location to location and, hence, from gateway to gateway. Consequently, the definitions are provided in event packages, which may be provided outside of the core specification. An initial list of event packages can be found in Appendix A.

The RequestedEvents and SignalRequests generally refer to the same events. 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.. There are a few exceptions to this rule, e.g., fax and modem tones, which can be detected but can not be signaled. However, we necessarily cannot expect all endpoints to detect all events. 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 endpoint. Each package specifies a list of events and signals that can be detected or applied. A gateway that is requested to detect or to apply an event that is not supported by the specified endpoint MUST return an error (error code 512 or 513 – not equipped to detect event or generate signal). When the event name is not qualified by a package name, the default package name for the endpoint is assumed. If the event name is not registered in this default package, the gateway MUST return an error (error code 522 – no such event or signal).

The Call Agent can send a NotificationRequest whose requested signal list is empty. This has the effect of stopping all active time-out signals. It can do so, for example, when tone generation, e.g., ringback, should stop.

QuarantineHandling is an optional parameter that specifies the handling options for events in the quarantine buffer (see Section 4.4.3.1), 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 (lockstep), or multiple notifications (loop), in response to this request (the default is at most one).

When the parameter is absent, the quarantined events MUST be processed. Support for the “lockstep” mode (via default) and “loop” mode is mandatory. An endpoint that receives a NotificationRequest with an unsupported QuarantineHandling parameter value SHOULD respond with error code 508 (unsupported QuarantineHandling.

Note that the quarantine-handling parameter also governs the handling of events that were detected and processed but not yet notified when the command is received.

DetectEvents is an optional parameter that specifies a minimum list of events that the gateway is requested to detect in the “notification” and “lockstep” state. When this parameter is absent, the events that MUST be detected in the quarantine period are those listed in the last received DetectEvents list. In addition, the gateway MUST also detect persistent events and the events specified in the RequestedEvents list, including those for which the "ignore" action is specified. Further explanation of this parameter may be found in Section 4.4.3.1.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 4.5) optionally followed by commentary.

4.3.2 Notifications

Notifications are sent via the Notify command by the gateway when an observed event is to be notified:

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 4.1.1. The identifier MUST be a fully qualified endpoint name, including the domain name of the gateway. The local part of the name MUST NOT use the wildcard convention. A Call Agent that receives a Notification with wildcard convention MUST return an error (the error returned SHOULD be error code 500 - the transaction could not be executed because the endpoint is unknown) in response.

NotifiedEntity is an optional parameter that identifies the entity to which the notification is sent. This parameter is equal to the NotifiedEntity parameter of the NotificationRequest that triggered this notification. Note that the MTA MAY include only the absolute domain name (including the hostname) of its NotifiedEntity if only the absolute domain name was received in the triggering NotificationRequest. The CMS SHOULD accept the value in this case. The parameter is absent if there was no such parameter in the triggering request. Regardless of the value of the **NotifiedEntity** parameter, the notification MUST be sent to the current "notified entity" for the endpoint.

RequestIdentifier is a parameter that repeats the RequestIdentifier parameter of the NotificationRequest that triggered this notification. It is used to correlate this notification with the notification request that triggered it. Persistent events will be viewed here as if they had been included in the last NotificationRequest. When no NotificationRequest has been received, the RequestIdentifier used will be zero ("0").

ObservedEvents is a list of events that the gateway detected and accumulated, either by the "accumulate", "accumulate according to digit map", or "notify" action. A single notification can report a list of events that will be reported in the order in which they were detected. The list can only contain persistent events and events that were requested in the RequestedEvents parameter of the triggering NotificationRequest. Events that were detected on a connection will include the name of that connection. The list will contain the events that were either accumulated (but not notified) or accumulated according to digit map (but no match yet), and the final event that triggered the notification or provided a final match in the digit map. It should be noted that digits are added to the list of observed events as they are accumulated, irrespective of whether they are accumulated according to the digit map or not. For example, if a user enters the digits "1234" and some event E is accumulated between the digits "3" and "4" being entered, the list of observed events would be "1, 2, 3, E, 4".

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 4.5) optionally followed by commentary.

4.3.3 CreateConnection

This command is used to create a connection.

ReturnCode
[, ConnectionId]
[, SpecificEndPointId]
[, LocalConnectionDescriptor]
[, ResourceID]
← CreateConnection(CallId
, EndpointId
[, NotifiedEntity]
[, LocalConnectionOptions]
, Mode
[, RemoteConnectionDescriptor]
[, RequestedEvents]
[, RequestIdentifier]
[, DigitMap]
[, SignalRequests]
[, QuarantineHandling]
[, DetectEvents])

This function is used when setting up a connection between two endpoints. A connection is defined by its attributes and the endpoints it associates. The input parameters in CreateConnection provide the data necessary to build one of the two endpoints' "view" of a connection.

CallId is a parameter that identifies the call (or session) to which this connection belongs. This parameter is, at a minimum, unique within the collection of Call Agents that control the same gateways; 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.

EndpointId is the identifier for the endpoint in the gateway where CreateConnection executes. The EndpointId can be specified fully by assigning a non-wildcarded value to the parameter EndpointId in the function call or it can be under-specified by using the "anyone" wildcard convention. If the endpoint is under-specified, the endpoint identifier will be assigned by the gateway and its complete value MUST be returned in the Specific EndpointId parameter of the response only if the command is successful. In this case, the endpoint assigned MUST be in service and MUST NOT already have any connections on it. The "all" wildcard convention MUST NOT be used. An embedded client that receives a CreateConnection with the "all" wildcard convention MUST return an error (the error returned SHOULD be error code 500 - the transaction could not be executed because the endpoint is unknown) in response.

NotifiedEntity is an optional parameter that specifies a new "notified entity" for the endpoint.

LocalConnectionOptions is a structure that describes the characteristics of the media data connection from the point-of-view of the gateway executing CreateConnection. It instructs the endpoint on send and receive characteristics of the media connection. The basic fields contained in LocalConnectionOptions are:

- **Encoding Method**: A list of literal names for the compression algorithm (encoding/decoding method) used to send and receive media on the connection MUST be specified with at least one value. The entries in the list are ordered by preference. The endpoint MUST choose at least one of the codecs, and the codec(s) SHOULD be chosen according to the preference indicated. If the endpoint receives any media on the connection encoded with a different encoding method, it MAY discard it. See Section 4.7 for details on the codec selection process.
    - The endpoint MUST additionally indicate which of the remaining compression algorithms it is willing to support as alternatives - see Section 5.4.1 for details.
    - A list of permissible encoding methods is specified in the PacketCable Audio/Video Codecs Specification [19]. The literal names defined in Section 4.5 (Table 3) of that specification MUST be used. Unknown compression algorithms SHOULD be ignored if they are received.

- **Packetization Period**: The packetization period in milliseconds, as defined in the SDP standard [4], MAY be specified and with exactly one decimal value. If this specifier is used, then the same packetization period MUST be used for all codecs allowed by the LocalConnectionOptions. Note that the MTA MUST NOT choose a codec with a packetization period that differs from that specified here. If different packetization periods for different codecs is desired, then this field MUST NOT be used. The value pertains to both media sent and received. Note that only the valid packetization period in conjunction with the associated encoding method are to be used by the MTA. A list of permissible packetization periods is specified in the PacketCable Audio/Video Codecs Specification [19]. This specifier MUST NOT be supplied in the same LCO as the Multiple Packetization Period field. An MTA MUST return an error (error code 524 – inconsistency in LocalConnectionOptions) when it receives an LCO with both the Packetization Period and Multiple Packetization Period fields.

- **Multiple Packetization Period:** A list of packetization periods in milliseconds, as defined in the SDP standard [4] MAY be specified if, and only if, the Encoding Method field is included. When specified, the multiple packetization period in milliseconds MUST contain exactly one decimal value or a hyphen for each entry in the encoding method field included in the LocalConnectionOptions. This applies even if several of the encoding methods have the same value. The first entry in the list MUST be a decimal number. When a hyphen is used, the codec in question MUST use the same packetization period as one of the other entries in the list that actually contains a decimal number, and furthermore the codec MUST NOT consume any more bandwidth that that other entry. This can for example be used for non-voice codecs (e.g., telephone-event or comfort noise) that use the same packetization period as the voice codec with which they are being used. Successive entries in the list of packetization periods MUST be ordered identically to the corresponding encoding methods. The values pertain to both media sent and received. Note that the MTA MUST NOT choose a codec with a packetization period that differs from that specified here. Note that only the valid packetization period in conjunction with the associated encoding method are to be used by the MTA. A list of permissible packetization periods is specified in the PacketCable Audio/Video Codecs Specification [19]. This specifier MUST NOT be supplied in the same LCO as the Packetization Period field. An MTA MUST return an error (error code 524 – inconsistency in LocalConnectionOptions) under the following conditions:

    - When it receives an LCO with both the Packetization Period and Multiple Packetization Period fields.

    - When it receives an LCO where the number of codecs specified in the Encoding Method field is different from the number of elements in the Multiple Packetization Period field.

- **Echo Cancellation**: Whether echo cancellation should be used initially on the line side or not[15]. The parameter can have the value “on” (when the echo cancellation is requested) or “off” (when it is turned off). The parameter is optional. When the parameter is omitted, the embedded client MUST apply echo cancellation initially. The embedded client SHOULD subsequently enable or disable echo cancellation in accordance with ITU-T recommendation V.8 and V.25 when voiceband data is detected. For re-enabling echo cancellation see e.g., G.168. Following termination of voiceband data, the handling of echo cancellation MUST revert to the current value of the echo cancellation parameter. It is RECOMMENDED that echo cancellation handling is left to the embedded client rather than having this parameter specified by the Call Agent.

- **Type of Service**: Specifies the class of service that will be used for sending media on the connection by encoding the 8-bit type of service value parameter of the IP header as two hexadecimal digits.[16] The parameter is optional. When the parameter is omitted, a default value of 0x00 (unless provisioned otherwise) will be used. The left-most "bit" in the parameter corresponds to the most significant bit in the IP header.

[15] Echo cancellation on the packet side is not supported.

[16] For RTP media streams, this type of service value applies only to the RTP media packets, not the RTCP packets.

- **Silence Suppression**: The embedded client may perform voice activity detection, and avoid sending packets during periods of silence. However, it is necessary, for certain calls (e.g., modem calls) to disable silence suppression. The parameter can have the value “on” (when silence is to be suppressed) or “off” (when silence is not to be suppressed). The parameter is optional. When the parameter is omitted, the default is "off". If the value is "on", upon detecting voiceband data, the endpoint SHOULD disable silence suppression. Following termination of voiceband data, the handling of silence suppression MUST revert to the current value of the silence suppression parameter.

The following LocalConnectionOptions fields are used to support Dynamic Quality of Service (D-QoS) – please refer to Appendix C for further details:

- **D-QoS GateID:** The GateID for the gate that has been setup at the CMTS. The GateID is a 32 bit identifier encoded as a string of up to 8 hex characters. This parameter is optional in general, but mandatory when D-QoS resource reservation and/or committal is to be performed. The presence of this parameter implies that D-QoS MUST be performed for this command. The absence of this parameter indicates that D-QoS MUST NOT be performed.
- **D-QoS Resource Reservation**: Allows explicit control over whether D-QoS resource reservation and/or committal should be performed in the send and/or receive direction or not. The parameter is optional and can have one or more of the following values:

  Reserve values:

“Send Reserve”	Resources are reserved in the send direction only.
“Receive Reserve”	Resources are reserved in the receive direction only.
“SendReceiveReserve”	Resources are reserved in the send and receive direction.

  Commit values:

“SendCommit”	Resources are committed in the send direction only.
“ReceiveCommit”	Resources are committed in the receive direction.
“SendReceiveCommit”	Resources are committed in the send and receive direction.

The parameter is optional, and multiple values are separated by commas. When D-QoS is to be performed, and the parameter is omitted, resource reservation MUST be performed for both the send and receive direction. The resources reserved are determined by the coding parameters applied to the connection, i.e., encoding method, packetization period, silence suppression, ciphersuite, etc. External parameters, such as the use of payload header suppression, may affect the amount of resources reserved as well. Please see the PacketCable Dynamic Quality of Service Specification [20] for details.

Receive resources can be reserved and committed without having obtained a RemoteConnectionDescriptor, whereas send resources can be reserved, but not committed, until a RemoteConnectionDescriptor is supplied. Note that, as long as a RemoteConnectionDescriptor has not been received, the resources reserved and committed must be based on the codec(s) selected locally. Once a RemoteConnectionDescriptor is received, the list of codec(s) that can actually be used for sending may contain a subset of these. The list of codecs that can be used for receiving is however unchanged until the endpoint issues a new LocalConnectionDescriptor. When D-QoS reservation is to be performed, and the parameter is omitted, resources MUST by default be committed based on the connection mode as specified in Table 3 below:

Table 3 - Default Resource Reservation Values

Connection Mode	D-QoS
"inactive"	Do not commit
"send only", "replicate"	Commit send
"receive only"	Commit receive
"send/receive", "conference", "network loopback", "network continuity test"	Commit send and receive

If a different commit operation is desired, the appropriate commit value is supplied and will be used instead. If a commit operation is to be performed, but no reservation has been made, or an existing reservation does not fully satisfy the resources to be committed[17], a reservation will be made automatically. If a reserve value is specified, but no commit value is specified, a commit operation will not be performed.

- **ResourceID**: An existing ResourceID for resources already reserved at the edge router. The use of the ResourceID allows separate reservations to reserve the same resource, however only one of the reservations can be active at a given point in time. The ResourceID is a 32-bit identifier encoded as a string of up to 8 hex characters. The parameter is optional. However, this parameter MUST be used for resource reservation by the embedded client if provided by the Call Agent.
- **ReserveDestination**: This optional parameter may specify an IPv4 address, optionally followed by a colon and a UDP port number, that is the destination for the resource reservation. When a UDP port number is not specified, a default value of 9 applies. The ReserveDestination is typically used when resource reservation is to be performed, and a RemoteConnectionDescriptor has not yet been provided for the connection. This enables reservations and downstream commits to be sent to the edge router when the source of a media stream is not yet known[18]. When a RemoteConnectionDescriptor has been provided, the parameter is ignored.

The following LocalConnectionOptions fields are used to support the PacketCable security services:

- **RTP ciphersuite:** A list of allowable ciphersuites for RTP security in order of preference. The entries in the list are ordered by preference where the first ciphersuite is the preferred choice**.** The endpoint MUST choose exactly one of the ciphersuites according to the rules described in the PacketCable Security Specification [21]. The endpoint SHOULD additionally indicate which of the remaining ciphersuites it is willing to support as alternatives (see Section 5.4.1 for details). Each ciphersuite is represented as ASCII strings consisting of two substrings separated by a slash ("/"), where the first substring identifies the authentication algorithm, and the second substring identifies the encryption algorithm. A list of permissible ciphersuites are specified in the PacketCable Security Specification [21].
- **RTCP ciphersuite:** A list of ciphersuites for RTCP security in order of preference. The entries in the list are ordered by preference where the first ciphersuite is the preferred choice. The endpoint MUST choose exactly one of the ciphersuites according to the rules described in the PacketCable Security Specification [21]. The endpoint SHOULD additionally indicate which of the remaining ciphersuites it is willing to support as alternatives. (See Section 5.4.1 for details). Each ciphersuite is represented as an ASCII string consisting of two substrings separated by a slash ("/"), where the first substring identifies the authentication algorithm, and the second substring identifies the encryption algorithm. A list of permissible ciphersuites is specified in the PacketCable Security Specification [21].

[17] This is not possible for the CreateConnection command but is noted here for completeness. It is possible for the ModifyConnection command however (see Section 4.3.4).

[18] Note that this will enable certain theft-of-service scenarios. See PacketCable Dynamic Quality of Service Specification [20] for details

The embedded client MUST respond with an error (error code 524 – **LocalConnectionOptions** inconsistency) if any of the above rules are violated. All of the above mentioned default values can be altered by the provisioning process.

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 the LocalConnectionDescriptor (not to be confused with LocalConnectionOptions), i.e., the fields that describe a session according to the SDP standard. Section 5.4 details the supported use of SDP in the NCS profile. This parameter may have a null value when the information for the remote end is not known. This occurs because the entity that builds a connection starts by sending a CreateConnection to one of the two gateways involved. 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.

When codecs are changed during a call, small periods of time may exist where the endpoints use different codes. As stated above, embedded clients MAY discard any media received that is encoded with a different codec than what is specified in the LocalConnectionOptions for a connection.

Mode indicates the mode of operation for this side of the connection. The options are "send only," "receive only", "send/receive", "conference", "inactive", "replicate", "network loopback" or "network continuity test". The handling of these modes is specified in the beginning of Section 4.3. Some endpoints may not be capable of supporting all modes. If the command specifies a mode that the endpoint does not support, an error MUST be returned (error code 517 – unsupported mode). Also, if a connection has not yet received a **RemoteConnectionDescriptor**, an error MUST be returned if the connection is attempted to be placed in any of the modes "send only", "send/receive", "replicate", or "conference", "netwloop" or "netwtest (error code 527 – missing **RemoteConnectionDescriptor**).

ConnectionId is a parameter returned by the gateway that uniquely identifies the connection within the context of the endpoint in question. The ConnectionId MUST be included with any provisional or successful response to a CreateConnection command. The ConncectionId MUST NOT be included when any error response is returned and the connection was not created.

LocalConnectionDescriptor is a parameter returned by the gateway, which is a session description that contains information about, e.g., addresses and RTP ports for "IN" connections as defined in SDP. It is similar to the RemoteConnectionDescriptor, except that it specifies this side of the connection. Section 5.4 details the supported use of SDP in the NCS profile. The LocalConnectionDescriptor MUST be included with any provisional or successful response to a CreateConnection command. The LocalConnectionDescriptor MUST NOT be included when any error response is returned and the connection was not created.

After receiving a "CreateConnection" command that does not include a RemoteConnectionDescriptor parameter, a gateway is in an ambiguous situation for the connection in question. Because it has exported a LocalConnectionDescriptor parameter, it potentially can receive packets on that connection. Because it has not yet received the other gateway's RemoteConnectionDescriptor parameter, it does not know whether the packets it receives have been authorized by the Call Agent. Thus, it must 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 (subject to security):

- If the mode was set to "receive only", the gateway MUST accept the voice signals received on the connection and transmit them through to the endpoint.
- If the mode was set to "inactive", the gateway MUST (as always) discard the voice signals received on the connection.
- Note, that when the endpoint does not have a RemoteConnectionDescriptor for the connection, the connection can by definition not be in any of the modes "send only", "send/receive", "replicate", or "conference", "netwloop" or "netwtest".

The **RequestedEvents, RequestIdentifier, DigitMap, SignalRequests, QuarantineHandling**, and **DetectEvents** parameters are all optional. They can be used by the Call Agent to effectively include a notification request that is executed simultaneously with the creation of the connection. If one or more of

these parameters is present, the RequestIdentifier MUST be one of them. Thus, the inclusion of a notification request can be recognized by the presence of a RequestIdentifier. The rest of the parameters may or may not be present. If one of the parameters is not present, it MUST be treated as if it was a normal **NotificationRequest** with the parameter in question being omitted. This may have the effect of canceling signals and of stop looking for events. Note that if the RequestedEvents and SignalRequests parameters are omitted, then the corresponding lists are considered empty only if a RequestIdentifier parameter is included.

As an example of use, consider a Call Agent that wants to place a call to an embedded client. The Call Agent should:

- ask the embedded client to create a connection, in order to be sure that the user can start speaking as soon as the phone goes off hook,
- ask the embedded client to start ringing,
- ask the embedded client to notify the Call Agent when the phone goes off-hook.

All of the above can be accomplished in a single CreateConnection command by including a notification request with the RequestedEvents parameters for the off-hook event and the SignalRequests parameter for the ringing signal.

When these parameters are present, the creation of the connection and the notification request MUST be synchronized, which means that they are both either accepted or refused. In our example, the CreateConnection must be refused if the gateway does not have sufficient resources or cannot get adequate resources from the local network access. The off-hook notification request must be refused in the glare condition if the user is already off-hook. In this example, the phone must not ring if the connection cannot be established, and the connection must not be established if the user is already off-hook. An error would be returned instead (error code 401 – phone off hook), which informs the Call Agent of the glare condition.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 4.5) optionally followed by commentary.

ResourceID is a D-QoS parameter that MAY be returned by the gateway. Note that this parameter MUST be returned by the MTA when D-QoS is to be performed as instructed by the Call Agent. When a successful D-QoS resource reservation is made, the ResourceID provides a handle for the resources reserved. The ResourceID MUST NOT be included when any error response is returned and the connection was not created.

4.3.4 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 descriptor, as well as the remote connection descriptor.

ReturnCode
[, LocalConnectionDescriptor]
[, ResourceID]
← ModifyConnection(CallId
, EndpointId
, ConnectionId
[, NotifiedEntity]
[, LocalConnectionOptions]
[, Mode]
[, RemoteConnectionDescriptor]
[, RequestedEvents]
[, RequestIdentifier]
[, DigitMap]
[, SignalRequests]
[, QuarantineHandling]
[, DetectEvents])

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

The **EndpointId** MUST be a fully qualified endpoint name. The local name MUST NOT use the wildcard convention. An embedded client that receives a ModifyConnection with a wildcard convention MUST return an error (the error returned SHOULD be error code 500 - the transaction could not be executed because the endpoint is unknown) in response.

The ModifyConnection command can be used to affect connection parameters, subject to the same rules and constraints as specified for CreateConnection:

- Provide information on the other end of the connection through the **RemoteConnectionDescriptor**.
- Activate or deactivate the connection by changing the **mode** parameter's value. This can occur at any time during the connection, with arbitrary parameter values. An activation can, for example, be set to the "receive only" mode.
- Change the parameters of the connection through the **LocalConnectionOptions**, for example, by switching to a different coding scheme, changing the packetization period, or modifying the handling of echo cancellation.

The details of D-QoS operation were specified in the CreateConnection command and generally the same rules apply here, except as noted below:

- **D-QoS GateID** A D-QoS GateID is mandatory when D-QoS operation is required, unless D-QoS operation has previously been done for the connection in question. In the latter case, the previously supplied D-QoS GateID MUST then be used by the MTA.
- **D-QoS Resource Reservation** Allows explicit control over whether D-QoS resource reservation and/or committal should be performed in the send and/or receive direction or not. The parameter is optional and multiple values can be specified. When the parameter is omitted and D-QoS reservation is to be performed, the default is to reserve in both the send and receive direction, unless a suitable reservation for the connection has already been made (see Appendix C). In that case, a new reservation will not be made. Resources are committed the same way as for **CreateConnection**, except when changing to "inactive" mode. In that case, the committed resources MUST be lowered to zero. An existing resource reservation is still maintained though.

- **ResourceID** The parameter is optional. When supplied, it MUST be used by the embedded client for resource reservation and replaces the ResourceID kept for the connection.
- **ReserveDestination** The parameter is optional. When supplied, it replaces the ReserveDestination kept by the embedded client for the connection. If a RemoteConnectionDescriptor has been supplied for the connection, the parameter is ignored.

The command will only return a **LocalConnectionDescriptor** if the local connection parameters, such as, e.g., RTP ports, etc. are modified. Thus, if, e.g., only the mode of the connection is changed, a LocalConnectionDescriptor will not be returned. The LocalConnectionDescriptor MUST NOT be included when any error response is returned and the connection was not modified. If a connection parameter is omitted, e.g., mode or silence suppression, the old value of that parameter will be retained if possible. If a parameter change necessitates a change in one or more *unspecified* parameters, the gateway is free to choose suitable values for the unspecified parameters that must change[19].

The RTP address information provided in the RemoteConnectionDescriptor specifies the remote RTP address of the receiver of media for the connection. This RTP address information may have been changed by the Call Agent[20]. When RTP address information is given to an embedded client for a connection, the embedded client SHOULD only accept media streams (and RTCP) from the IP address specified as well. Any media streams received from any other addresses SHOULD be discarded. The PacketCable Security Specification [21] should be consulted for additional security requirements.

The **RequestedEvents, RequestIdentifier, DigitMap, SignalRequests, QuarantineHandling**, and **DetectEvents** parameters are optional. The parameters can be used by the Call Agent to include a notification request that is tied to and executed simultaneously with the connection modification. If one or more of these parameters is supplied, then RequestIdentifier MUST be one of them. For example, when a call is accepted, the calling gateway should be instructed to place the connection in "send/receive" mode and to stop providing ringback tones. This can be accomplished in a single ModifyConnection command by including a notification request with the RequestedEvents parameters for the on-hook event, and an empty SignalRequests parameter, to stop the provision of ringback tones. Note that absence of the RequestedEvents and SignalRequests parameters is interpreted as an empty list only if a RequestIdentifier parameter is included.

When these parameters are present, the connection modification and the notification request MUST be synchronized, which means that they are both either accepted or refused.

NotifiedEntity is an optional parameter that specifies a new "notified entity" for the endpoint.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 4.5) optionally followed by commentary.

ResourceID is a D-QoS parameter that MUST be returned by the gateway if it performs a resource reservation and obtains a new ResourceID from the edge router. When a successful D-QoS resource reservation is made, the ResourceID provides a handle for the resources reserved. The Resource ID MUST NOT be included when any error response is returned and the connection was not modified.

[19] This can for instance happen if a codec change is specified, and the old codec used silence suppression, but the new one does not support it.

[20] For instance if media needs to traverse a firewall.

4.3.5 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
[, NotifiedEntity]
[, RequestedEvents]
[, RequestIdentifier]
[, DigitMap]
[, SignalRequests]
[, QuarantineHandling]
[, DetectEvents])

The endpoint identifier, in this form of the **DeleteConnection** command, MUST be fully qualified. Wildcard conventions MUST NOT be used. An embedded client that receives a Delete Connection with a wildcard convention MUST return an error (the error returned SHOULD be error code 500 - the transaction could not be executed because the endpoint is unknown) in response.

In the general case where a connection has two ends, this command has to be sent to both gateways involved in the connection. After the connection has been deleted, packet network media streams previously supported by the connection are no longer available. Any media packets received for the old connection are simply discarded and no new media packets for the stream are sent. When one or more D-QoS reservations and/or committals have been made for the connection, the DeleteConnection command will release the resources reserved.

In response to the DeleteConnection command, the gateway returns a list of parameters that describe the status of the connection. The connection parameters MUST only be returned if the command is successful and the connection is deleted. These parameters are:

- **Number of packets sent**. The total number of RTP data packets transmitted by the sender since starting transmission on the 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.
- **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 the connection. The count is not reset if the sender changes its SSRC identifier—for example, as a result of a ModifyConnection command.
- **Number of packets received**. The total number of RTP data packets received by the sender since starting reception on the connection. The count includes packets received from different SSRC if the sender used several values. The value is zero if, e.g., the connection was always 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 the connection. The count includes packets received from different SSRC if the sender used several values. The value is zero if, e.g., the connection was always 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 are 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, 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, e.g., the connection was always set in "send only" 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 [2]. The count includes packets received from different SSRC if the sender used several values. The value is zero if, e.g., the connection was always set in "send only" 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 the messages are received. The average is obtained by summing all the estimates and then dividing by the number of RTCP messages that have been received. It should be noted that the correct calculation of this parameter relies on synchronized clocks. Embedded client devices MAY alternatively estimate the average transmission delay by dividing the measured roundtrip time by two.

For a more detailed definition of these variables, please refer to [2].

In addition to the parameters above, an endpoint that has received one or more RTCP sender or receiver reports from its peer MUST return the following parameters:

- Remote Packets Sent: The number of packets that were sent on the connection from the perspective of the remote endpoint.
- Remote Octets Sent: The number of octets that were sent on the connection from the perspective of the remote endpoint.
- Remote Packets Lost: The number of packets that were not received on the connection, as deduced from gaps in the sequence number from the perspective of the remote endpoint.
- Remote Jitter: The average inter-packet arrival jitter, in milliseconds, expressed as an integer number from the perspective of the remote endpoint.

The **RequestedEvents, RequestIdentifier, DigitMap, SignalRequests, QuarantineHandling**, and **DetectEvents** parameters are optional. They can be used by the Call Agent to transmit a notification request that is tied to and executed simultaneously with the deletion of the connection. However, if one or more of these parameters are present, RequestIdentifier MUST be one of them. For example, when a user hangs up the phone, the gateway might be instructed to delete the connection and to start looking for an off-hook event. This can be accomplished in a single DeleteConnection command also by transmitting the RequestedEvents parameter for the off-hook event and an empty SignalRequests parameter. Note that absence of the RequestedEvents and SignalRequests parameters is interpreted as an empty list only if a RequestIdentifier parameter is included.

When these parameters are present, the delete connection and the notification request MUST be synchronized, which means that they are both either accepted or refused.

NotifiedEntity is an optional parameter that specifies a new "notified entity" for the endpoint.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 4.5) optionally followed by commentary.

4.3.6 DeleteConnection (From the Embedded Client)

In some circumstances, a gateway may have to clear a connection, for example, because it has lost the resource associated with the connection. The gateway can terminate the connection by using a variant of the DeleteConnection command:

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

The **EndpointId,** in this form of the **DeleteConnection** command, MUST be fully qualified. Wildcard conventions MUST NOT be used. A Call Agent that receives a DeleteConnection with a wildcard convention MUST return an error (the error returned SHOULD be error code 500 - the transaction could not be executed because the endpoint is unknown) in response.

The **Reason-code** is a text string starting with a numeric reason-code and optionally followed by a descriptive text string. A list of reason-codes can be found in Section 4.6.

In addition to the **CallId**, **EndpointId**, and **ConnectionId,** the embedded client will also send the connection's parameters, which would have been returned to the Call Agent in response to a DeleteConnection command from the Call Agent. The reason code indicates the cause of the DeleteConnection. When one or more D-QoS reservations and/or committals have been made for the connection, the embedded client will release the resources reserved.

ReturnCode is a parameter returned by the Call Agent. It indicates the outcome of the command and consists of an integer number (see Section 4.5) optionally followed by commentary.

4.3.7 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)

The **EndpointId**, in this form of the **DeleteConnection** command, MUST NOT use the "any of" wildcard. All connections for the endpoint(s) with the CallId specified will be deleted. The command does not return any individual statistics or call parameters. An embedded client that receives a **DeleteConnection** (Multiple Connections From the Call Agent) with the "any of" wildcard convention MUST return an error (the error returned SHOULD be error code 500 - the transaction could not be executed because the endpoint is unknown) in response.

DeleteConnection can also be used by the Call Agent to delete all connections that terminate in a given endpoint:

ReturnCode
← DeleteConnection(EndpointId)

In this form of the **DeleteConnection** command, Call Agents can take advantage of the hierarchical naming structure of endpoints to delete all the connections that belong to a group of endpoints. In this case, part of the "local endpoint name" component of the **EndpointId** can be specified using the "all" wildcarding convention, as specified in Section 4.1.1. The "any of" wildcarding convention MUST NOT be used. The command does not return any individual statistics or call parameters.

After the connection has been deleted, packet network media streams previously supported by the connection are no longer available. Any media packets received for the old connection are simply discarded and no new

media packets for the stream are sent. When one or more D-QoS reservations and/or committals have been made for the connection, the embedded client will release the resources reserved.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 4.5) optionally followed by commentary.

4.3.8 Auditing

The MGCP is based upon a centralized call control architecture where a Call Agent acts as the remote controller of client devices that provide voice interfaces to users and networks. In order to achieve the same or higher levels of availability as the current PSTN, some protocols have implemented mechanisms to periodically "ping" subscribers in order to minimize the time before an individual outage is detected. In this interest, an MGCP-specific auditing mechanism between the embedded clients and the Call Agents in a PacketCable system is provided to allow the Call Agent to audit endpoint and connection state and to retrieve protocol-specific capabilities of an endpoint.

Two commands for auditing are defined for the embedded clients:

- **AuditEndPoint**: Used by the Call Agent to determine the status of an endpoint.
- **AuditConnection**: Used by the Call Agent to obtain information about a connection.

Network management beyond the capabilities provided by these commands is generally desirable, e.g., information about the status of the embedded client as opposed to individual endpoints. Such capabilities are expected to be supported by the use of the Simple Network Management Protocol (SNMP) and by definition of a MIB for the embedded client, both of which are outside the scope of this specification.

4.3.8.1 AuditEndPoint

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

```
{ ReturnCode
  [, EndPointIdList]
  [, NumEndPoints] } |
 { ReturnCode
  [, RequestedEvents]
  [, DigitMap]
  [, SignalRequests]
  [, RequestIdentifier]
  [, NotifiedEntity]
  [, ConnectionIdentifiers]
  [, DetectEvents]
  [, ObservedEvents]
  [, EventStates]
  [, VersionSupported]
  [, ReasonCode
  [, MaxMGCPDatagram]
  [, Capabilities] }
  ← AuditEndPoint(EndpointId
                              [, RequestedInfo] |
                              { [, SpecificEndPointID]
                                    [, MaxEndPointIDs] })
```

The **EndpointId** identifies the endpoint that is being audited. The "any of" wildcard convention MUST NOT be used. An embedded client that receives an AuditEndPoint with the "any of" wildcard convention MUST return an error (the error returned SHOULD be error code 500 - the transaction could not be executed because the endpoint is unknown) in response.

The "all of" wildcard convention can be used to audit a group of endpoints. If this convention is used, the gateway MUST return the list of endpoint identifiers that match the wildcard in the **EndPointIdList** parameter, which is simply a list of **SpecificEndpointIds** – **RequestedInfo** MUST NOT be included in this case. **MaxEndPointIDs** is a numerical value that indicates the maximum number of **EndpointIds** to return. If additional endpoints exist, the **NumEndPoints** return parameter MUST be present and indicate the total number of endpoints that match the **EndpointID** specified. In order to retrieve the next block of **EndpointIDs**, the **SpecificEndPointID** is set to the value of the last endpoint returned in the previous **EndPointIDList**, and the command is issued.

When the wildcard convention is not used, the (possibly empty) **RequestedInfo** describes the information that is requested for the **EndpointId** specified – the **SpecificEndpointID** and **MaxEndpointID** parameters MUST NOT be used then. The following endpoint-specific information can then be audited with this command:

> RequestedEvents, DigitMap, SignalRequests, RequestIdentifier, NotifiedEntity, ConnectionIdentifiers, DetectEvents, ObservedEvents, EventStates, VersionSupported, ReasonCode, MaxMGCPDatagram, and Capabilities

If an endpoint is queried about a parameter it does not understand, the endpoint MUST NOT generate an error; instead the parameter MUST be omitted from the response.

If an endpoint is queried about a parameter it does support, but has no value for, the endpoint MUST NOT generate an error; instead the parameter MUST be included in the response with an empty parameter value.

Only when successful, the response MUST, in turn, include information about each of the items for which auditing information was requested. Excluding parameters that are explicitly marked "optional", endpoints MUST support all of the following parameters:

- **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 using currently.
- **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[21]. Time-Out signals that have timed-out, and currently playing Brief signals are not included. Parameterized signals are reported with the parameters they were applied with.
- **RequestIdentifier:** The RequestIdentifier for the last NotificationRequest received by the endpoint (includes notification request embedded in connection handling primitives). If no notification request has been received, the value zero will be returned.
- **NotifiedEntity:** The current "notified entity" for the endpoint. Note that the MTA MAY include only the absolute domain name (including the hostname) of its NotifiedEntity if only the absolute domain name (including the hostname) was provided to it via the NotifiedEntity parameter of an NCS message or acknowledgement. The CMS SHOULD accept the value in this case.
- **ConnectionIdentifiers**: A comma-separated list of ConnectionIdentifiers for all connections that currently exist for the specified endpoint.
- **DetectEvents:** The current value of DetectEvents the endpoint is using. Persistent events are included in the list.
- **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.

[21] Currently, there should be no pending brief signals.

- **VersionSupported:** A list of protocol versions supported by the endpoint.
- **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 normal.
- **MaxMGCPDatagram:** The maximum size of an MGCP datagram in bytes supported by the endpoint (see Section 5.5.3). The value excludes any lower layer overhead. Support for this parameter is optional. The default maximum MGCP datagram size is assumed if a value is not returned.
- **Capabilities:** The capabilities for the endpoint similar to the LocalConnectionOptions parameter and including event packages and connection modes. If any unknown capabilities are reported, they MUST simply be ignored. 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 literal names defined in the PacketCable Audio/Video Codecs Specification [19] MUST be used. Unknown compression algorithms SHOULD be ignored if they are received. 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.
    - **Dynamic Quality of Service:** Whether Dynamic Quality of Service is supported or not.
    - **Security:** Whether PacketCable Security services are supported or not. If supported, the following parameters may be present as well:
        - RTP Ciphersuites: A list of authentication and encryption algorithms supported for RTP.
        - RTCP Ciphersuites: A list of authentication and encryption algorithms supported for RTCP.

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

ReturnCode: A parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 4.5) optionally followed by commentary.

If no info was requested and the EndpointId refers to a valid fully-specified EndpointId, the gateway simply returns a successful response (return code 200 – transaction executed normally).

It should be noted, that all of the information returned is merely a snapshot. New commands received, local activity, etc. may alter most of the above. For example the hook-state may change before the Call Agent receives the above information.

4.3.8.2 AuditConnection

Auditing of individual connections on an endpoint can be achieved using the AuditConnection command.

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

The **EndpointId** identifies the endpoint that is being audited–wildcards MUST NOT be used. An embedded client that receives a AuditConnection with a wildcard convention MUST return an error (the error returned SHOULD be error code 500 - the transaction could not be executed because the endpoint is unknown) in response. 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,
ConnectionParameters, RemoteConnectionDescriptor,
LocalConnectionDescriptor.
```

If an endpoint is queried about a connection parameter it does not support, the endpoint MUST NOT generate an error; instead the parameter MUST be omitted from the response.

If an endpoint is queried about a connection parameter it does support, but has no value for, the endpoint MUST NOT generate an error; instead the parameter MUST be included in the response with an empty parameter value. Only when successful, the response MUST, in turn, include information about each of the items for which auditing information was requested. Excluding parameters that are explicitly marked "optional", endpoints MUST support all of the following parameters:

- **CallId:** The CallId for the call to which the connection belongs.
- **NotifiedEntity:** The current “notified entity” for the endpoint.
- **LocalConnectionOptions:** The LocalConnectionOptions supplied for the connection.
- **Mode:** The current connection mode.
- **ConnectionParameters:** Current connection parameters for the connection.
- **LocalConnectionDescriptor:** The LocalConnectionDescriptor that the gateway supplied for the connection.
- **RemoteConnectionDescriptor:** The most recent RemoteConnectionDescriptor that was supplied in a previous CreateConnection or ModifyConnection command to the gateway for this connection.

ReturnCode is a parameter returned by the gateway. It indicates the outcome of the command and consists of an integer number (see Section 4.5) optionally followed by commentary.

If no information was requested, and the EndpointId refers to a valid endpoint, the gateway simply checks that the connection specified exists and, if so, returns a positive response (return code 200 – transaction executed).

4.3.9 Restart in Progress

The RestartInProgress command is used by the gateway to signal that an endpoint, or a group of endpoints, is taken out of service or is being placed back in service.

ReturnCode
[, NotifiedEntity]
[, VersionSupported]
← RestartInProgress(EndpointId
, RestartMethod
[, RestartDelay]
[, ReasonCode])

The EndpointId identifies the endpoints that are taken in or out of service. The “all of” wildcard convention can be used to apply the command to a group of endpoints, 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 MUST NOT be used. ACall Agent that receives a Restart in Progress with the “any of” wildcard convention MUST return an error (the error returned SHOULD be error code 500 - the transaction could not be executed because the endpoint is unknown) in response. The RestartMethod parameter specifies the type of restart:

- A “graceful” restart method indicates that the specified endpoint(s) will be taken out of service after the specified “restart delay”. The established connections are not yet affected, but the Call Agent should refrain from establishing new connections, and should try to gracefully tear down any existing connections.
- A “cancel-graceful” restart method indicates that a gateway is canceling a previously issued "graceful" restart method for the same endpoints. When this command is sent, the gateway will immediately begin to allow the establishment of new connections on these endpoints.
- A “forced” restart method indicates that the specified endpoints are taken out of service abruptly. 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.

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” and “cancel-graceful” methods. A restart delay of null for the “restart” method indicates that service has already been restored. This typically will occur after gateway startup/reboot. To mitigate the effects of a client IP address change, the Call Agent MAY wish to resolve the embedded client’s domain name by querying the DNS regardless of the TTL of a current resource record for the restarted embedded client.

Embedded clients 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. However, an embedded client MUST send a “forced” RestartInProgress message when it is shutdown through the provisioning system. The Call Agent cannot rely on receiving such messages.

Embedded clients 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 4.4.3.5 – Call Agents can rely on receiving this message. Also, embedded clients MUST send a “disconnected” **RestartInProgress** message to their current “notified entity” according to the “disconnected” procedure

specified in Section 4.4.3.6. The “restart delay” parameter MUST NOT be used with the “forced” and “cancel graceful” restart method.

The optional ReasonCode parameter may be used to indicate the cause of the restart. 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. Embedded clients 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.

ReturnCode is a parameter returned by the Call Agent. It indicates the outcome of the command and consists of an integer number (see Section 4.5) optionally followed by commentary.

A NotifiedEntity MAY additionally be returned with the response to the RestartInProgress from the Call Agent - this should normally only be done in response to "restart" or "disconnected" (see also Section 4.4.3.5 and 4.4.3.6). If a NotifiedEntity parameter was included in the response returned, it specifies a new "notified entity" for the endpoint(s) – this operation SHOULD only be done with the response error code 521 (endpoint redirected). Note that the behavior for returning a NotifiedEntity in the response is only defined for RestartInProgress responses and SHOULD NOT be done for responses to other commands. Any other behavior is undefined.

- If the response indicated success (return code 200 – transaction executed), the restart in question completed successfully, and the NotifiedEntity returned is the new “notified entity” for the endpoint(s).
- If the response from the Call Agent indicated an error code, the restart in question is not yet complete. If the response was 521 (endpoint redirected), then the response MUST include a NotifiedEntity parameter, which specifies the new “notified entity” for the endpoint(s) and MUST be used when retrying the restart in question (as a new transaction).

In the case of "restart" and "disconnected", the restart in question MUST be retried whenever the Call Agent returns a transient (4xx) error code, whereas it SHOULD be retried for any other restartMethod. It is RECOMMENDED that any type of restart is terminated if a permanent (5xx) error code is returned, except for 521, as specified above.

Finally, a **VersionSupported** parameter with a list of supported versions may be returned if the response indicated version incompatibility (error code 528).

4.4 States, Failover and Race Conditions

In order to implement proper call signaling, 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 may exist when the gateway or the call agent are restarted: the gateway may need to be redirected to a new call agent during “failover” procedures. Similarly, the call agent may need to take special action when the gateway is taken offline, or restarted.

4.4.1 Recaps and Highlights

As mentioned in Section 4.1.4, Call Agents are identified by their domain name, and each endpoint has one, and only one, “notified entity” associated with it at any given point in time. In this section we recap and highlight the areas that are of special importance to reliability and failover in MGCP:

- A Call Agent is identified by its domain name, not its network addresses, and several network addresses can be associated with a domain name.
- An endpoint has one, and only one, Call Agent associated with it at any given point in time. The Call Agent associated with an endpoint is the current value of the “notified entity”.

- The “notified entity” is initially set to a provisioned value. When commands with a NotifiedEntity parameter are received for the endpoint, including wild-carded endpoint-names, the “notified entity” is set to the value specified. If the “notified entity” for an endpoint is empty or has not been set explicitly[22], the “notified entity” defaults to the source address of the last connection handling command or notification request received for the endpoint. In this case, the Call Agent will thus be identified by its network address, which SHOULD only be done on exceptional basis.
- Responses to commands are always sent to the source address of the command, regardless of the current “notified entity”. When a Notify message needs to be piggy-backed with the response, the datagram is still sent to the source address of the new command received, regardless of the NotifiedEntity for any of the commands.
- When the “notified entity” refers to a domain name that resolves to multiple IP-addresses, endpoints are capable of switching between each of these addresses, however they cannot change the “notified entity” to another domain name on their own. A call agent can however instruct them to switch by providing them with a new “notified entity”.
- If a 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 again, or for another (backup) call agent to contact the affected endpoints with a new “notified entity”.
- When another (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 so desired. Alternatively, the failed call agent could simply become the backup call agent now.

We should note that handover conflict resolution between separate Call Agent’s is not provided. We are relying strictly on the Call Agent's knowing what they are doing and communicating with each other (although AuditEndpoint can be used to learn about the current “notified entity”).

4.4.2 Retransmission, and Detection of Lost Associations

The MGCP protocol is organized as a set of transactions, each of which is composed of a command and a response. The MGCP messages, being carried over UDP, may be subject to losses. In the absence of a timely response (see Section 5.5), commands are repeated. Gateways MUST 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 Tt_{hist} seconds), and a list of the transactions that have not yet finished executing. The default value for Tt_{hist} is 30 seconds.

The transaction identifiers of incoming commands are first compared to the transaction identifiers of the recent responses. If a match is found, the gateway does not execute the transaction, but simply repeats the old response. If a match to a previously responded to transaction is not found, the transaction identifier of the incoming command is compared to the list of transactions that have not yet finished executing. If a match is found, the gateway does not execute the transaction; subsequent handling depends on the command in question. If it is a CreateConnection or ModifyConnection command, the gateway MUST send a provisional response. If it is any other command, it is simply ignored. In either case, a final response will be provided when the execution of the command is complete.

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

- transmission errors, when, e.g., a packet is lost due to noise on a line or congestion in a queue,
- component failure, when, e.g., an interface for a call agent becomes unavailable,
- call agent failure, when, e.g., all interfaces for a call agent becomes unavailable,
- failover, when a new call agent is “taking over” transparently.

[22] This could for instance happen by specifying an empty NotifiedEntity parameter.

The elements should be able to derive from the past history an estimate of the packet loss rate. In a properly configured system, this loss rate should be very low, typically less than 1% on average. 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 uniformly distributed 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 repetitions that is considered excessive should be a function of the prevailing packet loss rate.) When errors are non-uniformly distributed, the consecutive failure probability can become somewhat higher. We should note that the "suspicion threshold", which we will call "Max1", is normally lower than the "disconnection threshold", which we will call "Max2". Max2 MUST be set to a larger value than Max1.

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: (A retransmission algorithm including these modifications is illustrated in Figure 4 below.)

- The gateway MUST always check for the presence of a new call agent. It can be noticed by:
  - receiving a command where the NotifiedEntity points to a new call agent, or
  - receiving a redirection response pointing to a new call agent.
- If a new Call Agent is detected, the gateway MUST direct retransmissions of any outstanding commands for the endpoint(s) redirected to that new Call Agent. Responses to new or old commands are still transmitted to the source address of the command.
- Prior to any retransmission, it is checked that the time elapsed since the sending of the initial datagram is no greater than Ts_{max}. If more than Ts_{max} time has elapsed, then retransmissions MUST cease. If more than 2*Tthist has elapsed, then the endpoint becomes disconnected.
- If the number of retransmissions to this Call Agent equals "Max1", the gateway MAY actively query the name server in order to detect the possible change of call agent interfaces, regardless of the Time To Live (TTL) associated with the DNS record.
- The gateway may have learned several IP addresses for the Call Agent. If the number of retransmissions for this IP address is greater than or equal to "Max1" and lower than "Max2", and there are more IP addresses that have not been tried, then the gateway MUST direct the retransmissions to the remaining alternate addresses in its local list. Also, receipt of explicit network notifications such as, e.g., ICMP network, host, protocol, or port unreachable SHOULD lead the gateway to try alternate addresses (with due consideration to possible security issues).
- If there are no more interfaces to try, and the number of retransmissions is Max2, then the gateway SHOULD contact the DNS one more time to see if any other interfaces have become available. If there still are no more interfaces to try, then retransmissions MUST cease. If more than 2*Tthist has elapsed, then the endpoint becomes disconnected and MUST initiate the "disconnected" procedure as specified in Section 4.4.3.6.

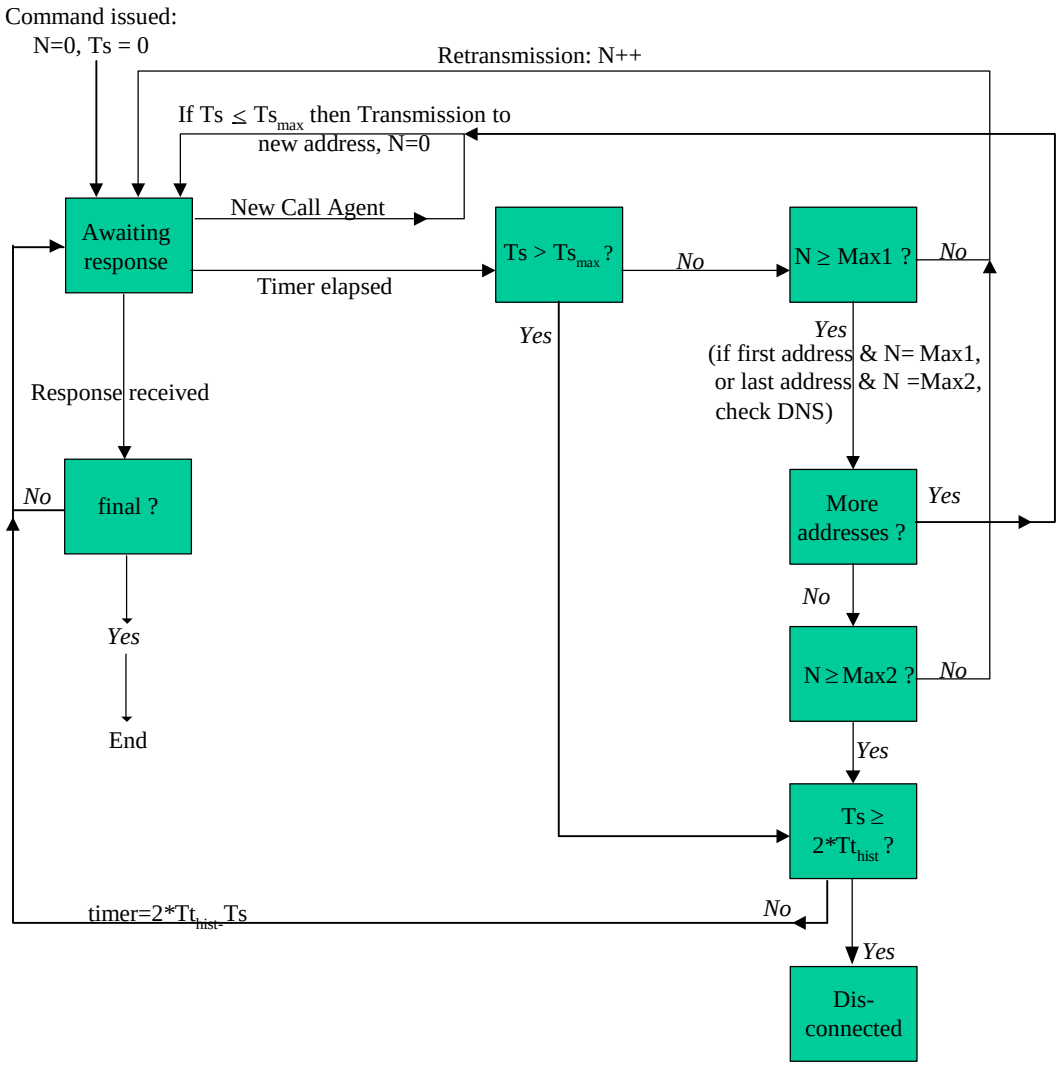

Figure 4 - Retransmission Algorithm

In order to adapt to network load automatically, MGCP specifies exponentially increasing timers (see Section 5.5.2). If the initial time-out 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 retransmissions 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 (Ts) elapsed since the sending of the initial datagram is no greater than Ts_{max}. If more than Ts_{max} time has elapsed, retransmissions MUST cease. When Ts_{max} has expired, or all of the retransmissions to all known IP addresses have been sent, there is a pause before declaring the endpoint disconnected. This pause represents a period of time where the only action is to wait for a response from any of the recent retransmissions. The quiescent period lasts for what remains of twice the life expectancy of the original transaction (2*Tthist). This settling time allows all active transactions to complete or time out before the endpoint is declared disconnected. This helps ensure that each restart of the endpoint is from a clean and initial state. If more than 2* Tt_{hist} time has elapsed, the endpoint becomes disconnected. The value Ts_{max} is related to the Tt_{hist} value: the Tt_{hist} value MUST be greater than or equal to Ts_{max} plus the maximum propagation delay in the network, Tp_{max}.

In other words, the following relation MUST be satisfied to prevent retransmitted commands from being executed more than once: $Tt_{hist} \geq Ts_{max} + Tp_{max}$

The default value for Ts_{max} is 20 seconds. Thus, if the assumed maximum propagation delay is 10 seconds, then responses to old transactions must be kept for a period of at least 30 seconds. The importance of having the sender and receiver agree on these values cannot be overstated.

The default value for Max1 is 5 retransmissions and the default value for Max2 is 7 retransmissions. Both of these values may be altered by the provisioning process.

Furthermore, the provisioning process MUST be able to disable one or both of the Max1 and Max2 DNS queries.

4.4.3 Race Conditions

In this section we describe how MGCP deals with race conditions.

First of all, MGCP deals with race conditions through the notion of a “quarantine list” that quarantines events and through explicit detection of desynchronization, e.g., for mismatched hook-state due to glare for an endpoint.

Secondly, MGCP does not assume that the transport mechanism will maintain the order of commands and responses. This may cause race conditions that may be obviated through a proper behavior of the call agent by a proper ordering of commands.

Finally, 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 capability are reestablished, many gateways may decide to send RestartInProgress commands simultaneously, which could lead to very unstable operation if not carefully controlled.

4.4.3.1 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", the “Quarantine Handling”, and the “Detect Events” list.

When the endpoint is initialized, the requested events list only consists of persistent events for the endpoint, and the digit map is assumed empty. After reception of a NotificationRequest command, the gateway starts observing the endpoint for occurrences of the events mentioned in the list, including persistent events.

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 “accumulate according to digit map” are accumulated in a list of observed events. The events that are marked as “accumulate according to the digit map” will additionally be accumulated in the “current dial string”. This will go on until one event is encountered that triggers a Notify command which will be sent to the current “notified entity”.

The gateway, at this point, will transmit the Notify command and will place the endpoint in a “notification state”. As long as the endpoint is in this “notification state”, the events that are detected on the endpoint are stored in a “quarantine” buffer for later processing. The events are, in a sense, “quarantined". The events detected are the events specified by the union of the RequestedEvents parameter and the most recently received DetectEvents parameter or, in case no DetectEvents parameter has been received, the events that are referred to in the RequestedEvents. Persistent events are detected as well.

The endpoint exits the “notification state” when the response (whether success or failure) to the Notify command is received. The Notify command may be retransmitted in the “notification state”, as specified in Section 4.4.2. If the endpoint is or becomes disconnected (see Section 4.4.2) during this, a response to the Notify command will never be received. The Notify command is then lost and hence no longer considered pending, yet the endpoint is still in the "notification state". Should that occur, completion of the disconnected procedure specified in Section 4.4.3.6 shall then lead the endpoint to exit the "notification state".

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 triggering NotificationRequest command:

If the Call Agent had specified that it expected, at most, one notification in response to the notification request command (“lockstep” mode), then the gateway MUST simply keep on accumulating events in the quarantine buffer until it receives the next notification request command. Until this happens, the endpoint is in a “lockstep state”, and events that occur and are to be detected, are simply stored in the quarantine buffer. The events to be quarantined are the same as in the “notification state”. Once the new NotificationRequest is received and executed successfully, the endpoint exits the “lockstep state”.

If, however, the gateway is authorized to send multiple successive Notify commands (“loop” mode), 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 triggers 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 buffer,
- it can attempt to empty the quarantine buffer and transmit a single Notify command reporting several sets of events. The “current dial string” MUST then be reset to a null value after each triggering event. The events that follow the last triggering event MUST be left in the quarantine buffer.

If the gateway transmits a Notify command, the endpoint will reenter and remain in the "notification state" until the acknowledgement is received (as described above). If the gateway does not find a quarantined event that triggers a Notify command, it places the endpoint 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 can receive at any time a new NotificationRequest command for the endpoint, including the case where the endpoint is disconnected, which will also have the effect of taking the endpoint out of the “notification state” assuming the NotificationRequest executes successfully. Activating an embedded NotificationRequest is here viewed as receiving a new NotificationRequest as well, except that the current list of ObservedEvents remains unmodified rather than being processed again.

When a new NotificationRequest is received in the “notification state”, the gateway SHOULD attempt to deliver the pending Notify (note that a Notify that was lost due to being disconnected, is no longer considered pending) prior to a successful response to the new NotificationRequest. It does so by using the “piggy-backing” functionality of the protocol and placing the messages (commands and responses) to be sent in order with the oldest message first. The messages will then be sent in a single packet to the source of the new NotificationRequest, regardless of the source and “notified entity” for the old and new command. The steps involved are the following:

1. the gateway builds a message that carries in a single packet a repetition of the old outstanding Notify command and the response to the new NotificationRequest command.
2. the endpoint is then taken out of the “notification state” without waiting for the response to the Notify command.
3. a copy of the outstanding Notify command is kept until a response is received. If a time-out occurs, the Notify will be repeated, in a packet that will also carry a repetition of the response to the NotificationRequest.
    - If the packet carrying the response to the NotificationRequest is lost, the Call Agent will retransmit the NotificationRequest. The gateway will reply to this repetition by retransmitting in a single packet the outstanding Notify command and the response to the NotificationRequest – this datagram will be sent to the source of the NotificationRequest.

- Notify(s) for a given endpoint MUST be delivered in-order. If the gateway has to transmit a new Notify before a response to the previous Notify is received, it constructs a packet that piggy-backs a repetition of the old Notify, a repetition of the response to the last NotificationRequest, and the new Notify – this datagram will be sent to the current "notified entity".

After receiving a NotificationRequest command, the "requested events" list, and "digit map" (if a new one was provided) are replaced by the newly received parameters, and the "current dial string" is reset to a null value. Furthermore, when the NotificationRequest was received in the "notification state", the list of observed event is reset to a null value. The subsequent behavior is then conditioned by the value of the QuarantineHandling parameter. The parameter may specify that quarantined events, and observed events (which in this case is an empty list), are to be discarded, in which case all quarantined and observed events are discarded. If the parameter specifies that the quarantined and observed events should be processed, the gateway will start processing the list of quarantined and observed events, using the newly received list of "requested events" and "digit map" if provided. When processing these events, the gateway may encounter an event, which triggers 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" up until and including the triggering event, leaving the unprocessed events in the quarantine buffer. The endpoint then enters the "notification state" again.

A new notification request may be received while the gateway has accumulated events according to the previous notification requests, but has not yet detected any 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 events 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. The only exception is the activation of an embedded Notification Request. In this case the observed event list remains unmodified rather than being processed again.

The above procedure applies to all forms of notification requests, regardless of whether they are part of a connection handling command or provided as a NotificationRequest command. Connection handling commands that do not include a notification request are neither affected by nor do they affect the above procedure.

Figure 5 below illustrates the procedure specified above assuming all transactions execute successfully.

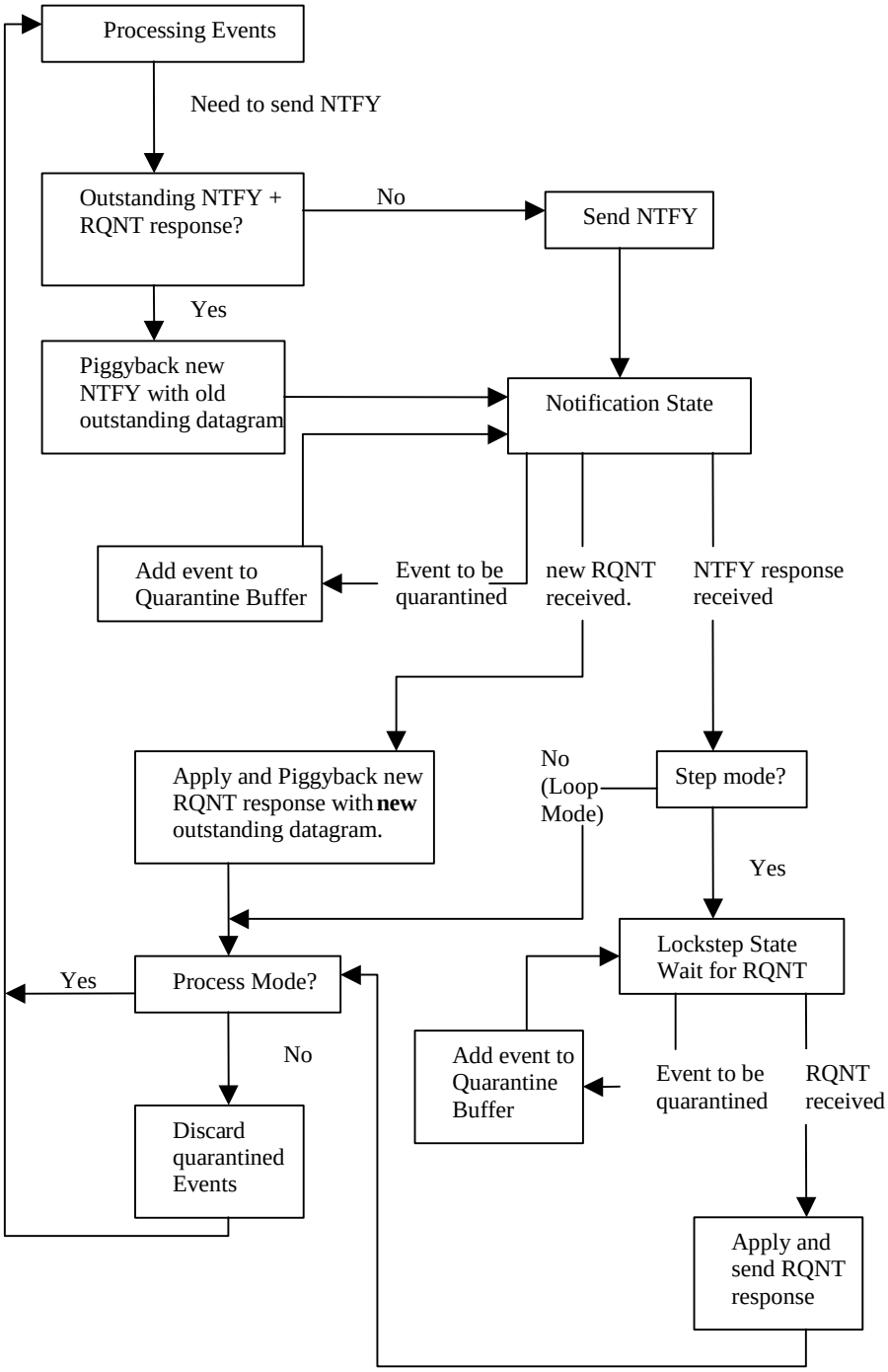

Figure 5 - Quarantine List Procedures

Call Agents SHOULD provide the response to a successful Notify message and the new NotificationRequest in the same datagram using the piggy-backing mechanism.[23]

4.4.3.2 Explicit Detection

A key element of the state of several endpoints is the position of the hook. Although hook-state changing events are persistent in NCS, race conditions and state mismatch may still occur, for example when the user decides to go off-hook while the Call Agent is in the process of requesting the gateway to look for off-hook events and perhaps apply a ringing signal (the "glare" condition well known in voice-based capabilities).

To avoid this race condition, the gateway MUST check the condition of the endpoint before responding to a **NotificationRequest**. Specifically, it MUST return an error:

- If the gateway is requested to notify an "off hook" transition while the phone is already off hook (error code 401 – phone off hook),
- If the gateway is requested to notify an "on hook" or "flash hook" condition while the phone is already on hook (error code 402 – phone on hook).

Additionally, individual signal definitions can specify that a signal will only operate under certain conditions, e.g., ringing may only be possible if the phone is already off hook. If such prerequisites exist for a given signal, the gateway MUST return the error specified in the signal definition if the prerequisite is not met.

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 requested events 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. although an error is returned. As all other transactions, the **NotificationRequest** MUST operate as an atomic transaction. Thus any changes initiated as a result of the command MUST be reverted.

Another race condition can 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 thereby enabling the Call Agent to determine if the Notify command was generated before or after the gateway received the new **NotificationRequest**.

4.4.3.3 Transactional Semantics

As the potential transaction completion times increases, e.g., due to external resource reservations, a careful definition of the transactional semantics becomes increasingly important. In particular the issue of race conditions, specifically as it relates to hook-state must be defined carefully.

An important point to consider is, that the hook-state may in fact change between the time a transaction is initiated and the time it completes. More generally, we may say that the successful completion of a transaction depends on one or more pre-conditions where one or more of the pre-conditions may change dynamically during the execution of the transaction.

The simplest semantics for this is simply to require that all pre-conditions MUST be met from the time the transaction is initiated until the transaction completes. Thus, if any of the preconditions change during the execution of the transaction, the transaction MUST fail. Furthermore, as soon as the transaction is initiated, all new events are quarantined. When the outcome of the transaction is known, all quarantined events are then processed.

[23] Vendors who choose not to follow this recommendation should examine call agent failure scenarios carefully.

As an example, consider a transaction that includes a request for the "off-hook" event. When the transaction is initiated the phone is "on-hook" and this pre-condition is therefore met. If the hook-state changes to "off-hook" before the transaction completes, the pre-condition is no longer met, and the transaction therefore immediately fails. The "off-hook" event will now be stored in the "quarantine" buffer which then gets processed.

4.4.3.4 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 drawbacks. 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 response to a previous one is received, there is no guarantee that the previous one will not be received in second position.

Call Agents and gateways that want to guarantee consistent operation of the endpoints can use the rules specified below:

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 MUST be ignored, and an error returned (error code 515 – incorrect connection-id).
4. On a given endpoint, there should normally be only one outstanding NotificationRequest command at any time. The RequestId parameter is used to correlate Notify commands with the triggering NotificationRequest.
5. In some cases, an implicitly or explicitly wild-carded 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 should not be sent until a response to the wild-carded DeleteConnection command is received.
6. When commands are embedded within each other, sequencing requirements for all commands MUST be adhered to. For example a CreateConnection command with a notification request in it must adhere to the sequencing requirements for CreateConnection and NotificationRequest at the same time.
7. AuditEndpoint and AuditConnection are not subject to any sequencing.
8. RestartInProgress must always be the first command sent by an endpoint as defined by the restart procedure (see Section 4.4.3.5). 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.

Those of the above rules that specify gateway behavior MUST be adhered to by embedded clients, however the embedded client MUST NOT make any assumptions as to whether Call Agents follow the rules or not. Consequently gateways MUST always respond to commands, regardless of whether they adhere to the above rules or not.

To ensure consistent operation, embedded clients SHOULD behave as specified below when one or more of the above rules are not followed:

- Where a single outstanding command is expected (ModifyConnection, NotificationRequest), but the same command is received in a new transaction before the old finishes executing, the gateway SHOULD fail the previous command. This includes the case where one or more of the commands were encapsulated. The use of error code 407 (transaction aborted) is RECOMMENDED.

- If a ModifyConnection command is received for a pending CreateConnection command, the ModifyConnection command SHOULD simply be rejected. The use of error code 400 (transient error) is RECOMMENDED. Note that this situation constitutes a Call Agent programming error.

Note, that where reception of a new command leads to aborting an old command, the old command SHOULD be aborted regardless of whether the new command succeeds or not. For example, if a ModifyConnection command is aborted by a DeleteConnection command which itself fails due to an encapsulated NotificationRequest, the ModifyConnection command is still aborted.

4.4.3.5 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 MUST be followed:

1. When a gateway is powered on, it initiates a restart timer to a random value, uniformly distributed between 0 and a provisionable maximum waiting delay (MWD), e.g., 360 seconds (see below). Care MUST be taken to avoid synchronicity of the random number generation between multiple gateways 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, such as for example an off-hook transition on a residential gateway. A pre-existing off-hook condition results in the generation of an off-hook event.
3. When the restart timer elapses, when a command is received, or when an activity or pre-existing off-hook condition is detected, the gateway initiates the restart procedure.

The restart procedure simply states that the endpoint MUST send a **RestartInProgress** command to the Call Agent informing it about the restart and furthermore guarantee that the first message (command or response) that the Call Agent sees from this endpoint MUST be this **RestartInProgress** command. The endpoint MUST take full advantage of piggy-backing in achieving this. For example, if an off-hook activity occurs prior to the restart timer expiring, a packet containing the **RestartInProgress** command, and with a piggy-backed Notify command for the off-hook event will be generated. In the case where the restart timer expires without any other activity, the gateway simply sends a RestartInProgress message.

Note that if the RestartInProgress is piggy-backed with the response (R) to a command received while restarting, then retransmission of the RestartInProgress does not require piggy-backing of the response R. However, while the endpoint is restarting, a resend of the response R does require the RestartInProgress to be piggy-backed to ensure in-order delivery of the two. The restart procedure is complete once a success response has been received. If an error response is received, the subsequent behavior depends on the error code in question:

- If the error code indicates a transient error (4xx), then the restart procedure MUST be initiated again (as a new transaction).
- If the error code is 521, then the endpoint is redirected, and the restart procedure MUST be initiated again (as a new transaction). The 521 response should have included a NotifiedEntity which then is the "notified entity" towards which the restart is initiated.
- If the error is any other permanent error (5xx), then it is RECOMMENDED that the endpoint no longer initiates the restart procedure on its own (until rebooted) unless otherwise specified. If a command is received, the endpoint MUST initiate the restart procedure again.

Should the gateway enter the "disconnected" state while carrying out the restart procedure, the disconnected procedure specified in Section 4.4.3.6 MUST be carried out, except that a "restart" rather than "disconnected" message is sent during the procedure.

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, on 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 transactions between each endpoint and the Call Agent. This simple calculation shows that the Call Agent is expected to handle 5 to 6 transactions for each endpoint, every 30 minutes on average, or, to put it otherwise, about one transaction per endpoint every 5 to 6 minutes on average. This suggests that a reasonable value of MWD for a residential gateway would be 10 to 12 minutes. In the absence of explicit configuration, embedded clients MUST use a default value of 600 seconds for MWD.

4.4.3.6 Disconnected Endpoints

In addition to the restart procedure, embedded clients also have a "disconnected" procedure, which is initiated when an endpoint becomes "disconnected" as described in Section 4.4.2. 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 (Td_{init}), 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 MUST initiate the "disconnected" procedure with a new transaction ID for the endpoint. In the case of local user activity, a provisionable "disconnected" minimum waiting delay (Td_{min}) must furthermore have elapsed since the gateway became disconnected or the last time it ended 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, a new value for the "disconnected" timer is selected. The timer value MUST be selected from the range defined by 1.5 times the last timer value and double the last timer value, and MAY be randomly generated. In either case, the new timer value is subject to a provisionable "disconnected" maximum waiting delay (Td_{max}), 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. During each initiation of "disconnected" procedure, the command MUST observe the normal retransmission and transaction identifiers requirements. (See Section 4.4.2.) 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.

Note that if a disconnected procedure is already in progress when a command is received, the existing disconnect procedure MUST be terminated and a new procedure MUST be started. This is to support a possible call agent redirection.

Also note, that if the RestartInProgress is piggy-backed with the response (R) to a command received while being disconnected, then retransmission of the RestartInProgress does not require piggy-backing of the response R. However, while the endpoint is disconnected, resending the response R does require the RestartInProgress to be piggy-backed as well to ensure the in-order delivery of the two.

The disconnected procedure is complete once a success response has been received. Error responses are handled similarly to the restart procedure (Section 4.4.3.5). If the "disconnected" procedure is to be initiated again following an error response, the rate-limiting timer considerations specified above still apply. A disconnected endpoint may wish to send a command (besides RestartInProgress) while it is disconnected. Doing so will only succeed once the Call Agent is reachable again, which raises the question of what to do with such a command meanwhile. At one extreme, the endpoint could drop the command right away, however that would not work very well when the Call Agent was in fact available, but the endpoint had not yet completed the "disconnected" procedure (consider for example the case where a NotificationRequest was just received which immediately resulted in a Notify being generated). To prevent such scenarios, disconnected endpoints MUST NOT blindly drop new commands to be sent for a period of Ts_{max} seconds after they receive a non-audit command. One way of satisfying this requirement is to employ a temporary buffering of commands to be sent, however in doing so, the endpoint must ensure, that it:

- does not build up a long queue of commands to be sent,
- does not swamp the Call Agent by rapidly sending too many commands once it is connected again.

Buffering commands for Ts_{max} seconds and, once the endpoint is connected again, limiting the rate at which buffered commands are sent to one outstanding command per endpoint is considered safe. If the endpoint is not connected within Ts_{max} seconds, but a "disconnected" procedure is initiated within Ts_{max} seconds, the endpoint MAY piggy-back the buffered command(s) with that RestartInProgress. Note, that once a command has been sent, regardless of whether it was buffered initially, or piggy-backed earlier, retransmission of that command MUST cease Ts_{max} seconds after the initial send as described in Section 4.4.2. 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 on affected endpoints.

The default value for Td_{init} is 15 seconds, the default value for Td_{min}, is 15 seconds, and the default value for Td_{max} is 600 seconds.

4.5 Return Codes and Error Codes

All MGCP commands receive a response. The response carries a return code that indicates the status of the command. The return code is an integer number, for which five value ranges have been defined:

- value 000 indicates a response acknowledgement[24],
- 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 defined are listed in Table 4 below:

Table 4 - Return Code Definitions

Code	Meaning
000	Response acknowledgement.
100	The transaction is currently being executed. An actual completion message will follow later.
200	The requested transaction was executed normally.
250	The connection(s) 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.
407	Transaction aborted. The transaction was aborted by some external action, e.g., a ModifyConnection command aborted by a DeleteConnection command.
500	The transaction could not be executed because the endpoint is unknown.
501	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.
503	"All of" wildcard not fully supported. The transaction contained an "all of" wildcard, however the gateway does not fully support these. Note that this is currently only permissible for non-empty NotificationRequests.
505	Unsupported RemoteConnectionDescriptor. This SHOULD be used when one or more mandatory parameters or values in the RemoteConnectionDescriptor is not supported.
506	Unable to satisfy both LocalConnectionOptions and RemoteConnectionDescriptor. This SHOULD be used when the LocalConnectionOptions and RemoteConnectionDescriptor contain one or more mandatory parameters or values that conflict with each other and/or cannot be supported at the same time (except for codec negotiation failure – see error code 534).
508	Unknown or unsupported quarantine handling.
510	The transaction could not be executed because a protocol error was detected.
511	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.

[24] Response acknowledgement is used for provisional responses (see Section 5.8)

Code	Meaning
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.
532	Unsupported value(s) in LocalConnectionOptions.
533	Response too big.
534	Codec negotiation failure.
538	Event/signal parameter error (e.g., missing, erroneous, unsupported, unknown, etc.).

4.6 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 Call Agent of the reason for the restart. The reason code is an integer number. Defined values of the reason code are listed in Table 5 below:

Table 5 - Reason Code Definitions

Code	Meaning
000	Endpoint state is normal. (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).
903	QoS resource reservation was lost.

4.7 Use of Local Connection Options and Connection Descriptors

The normal sequence in setting up a bi-directional connection involves at least three steps:

1. The Call Agent asks the first gateway to "create a connection" on an endpoint. The gateway allocates resources to that connection, and responds to the command by providing a "session description" (referred

to as its LocalConnectionDescriptor). The session description contains the information necessary for another party to send packets toward the newly created connection.

2. The Call Agent then asks the second gateway to "create a connection" on an endpoint. The command carries the "session description" provided by the first gateway (now referred to as the RemoteConnectionDescriptor). The gateway allocates resources to that connection, and responds to the command by providing its own "session description" (LocalConnectionDescriptor).

3. The Call Agent uses a "modify connection" command to provide this second "session description" (now referred to as the RemoteConnectionDescriptor) to the first endpoint. Once this is done, communication can proceed in both directions.

   When the Call Agent issues a Create or Modify Connection command, there are thus three parameters that determine the media supported by that connection:

   - LocalConnectionOptions: Supplied by the Call Agent to control the media parameters used by the gateway for the connection. When supplied, the gateway must conform to these media parameters until either the connection is deleted, or a ModifyConnection command is received.
   - RemoteConnectionDescriptor: Supplied by the Call Agent to convey the media parameters supported by the other side of the connection. When supplied, the gateway must conform to these media parameters until either the connection is deleted, or a ModifyConnection command is received.
   - LocalConnectionDescriptor: Supplied by the gateway to the Call Agent to convey the media parameters it supports for the connection. When supplied, the gateway must honor the media parameters until either the connection is deleted, or the gateway issues a new LocalConnectionDescriptor.

   Codec and packetization period selection must only be performed, as described in this section, if either:

   a) The gateway receives a CRCX, or

   b) The gateway receives a MDCX and any of the following parameters are present:

      - encoding method (a: in LocalConnectionOptions)
      - packetization period (p: in LocalConnectionOptions)
      - multiple packetization period (mp: in LocalConnectionOptions)
      - RemoteConnectionDescriptor

   Furthermore, this codec and packetization period selection process must only use the information present in the connection request and not retain any of the values that may have been received in previous connection requests. For example, if a gateway received a MDCX with all necessary LCO parameters but was missing a RemoteConnectionDescriptor, it will negotiate as if no RemoteConnectionDescriptor had ever been received for that connection. As well, if all of the above parameters are omitted in a MDCX command the existing negotiated codecs and packetization periods will remain intact.

   In determining which codec(s) and packetization period(s) to provide in the LocalConnectionDescriptor, there are three lists of codecs and packetization periods that a gateway needs to consider:

   - A list of codecs and packetization periods allowed by the LocalConnectionOptions. A codec is allowed by the LocalConnectionOptions if it satisfies the constraints specified by the encoding method, packetization period and multiple packetization periods fields. If one or more of these fields are omitted, the omitted fields do not impose any constraints on the allowed codecs.
   - A list of codecs and packetization periods in the RemoteConnectionDescriptor.
   - An internal list of codecs and packetization periods that the gateway can support for the connection. A gateway may support one or more codecs and packetization periods for a given connection.

Codec selection (including all relevant media parameters) can then be described by the following steps:

1. An approved list of codecs/packetization periods is formed by taking the intersection of the internal list of codecs/packetization periods and codecs/packetization periods allowed by the LocalConnectionOptions. If LocalConnectionOptions was not provided, the approved list of codecs/packetization periods thus contains the internal list. If the LocalConnectionOptions was provided but the codecs parameter was omitted, the LocalConnectionOptions implicitly allows all codecs in the internal list, provided they are not incompatible with any packetization period(s) specified. Similarly, if the LocalConnectionOptions was provided but the packetization period(s) was omitted, the LocalConnectionOptions implicitly contains the set of packetization periods supported by the internal list.
2. If the approved list of codecs/packetization periods is empty, a codec negotiation failure has occurred and an error response is generated (error code 534 – codec negotiation failure – is recommended).
3. Otherwise, a negotiated list of codecs/packetization periods is formed by taking the intersection of the approved list of codecs/packetization periods and codecs/packetization periods allowed by the RemoteConnectionDescriptor. If a RemoteConnectionDescriptor was not provided, the negotiated list of codecs/packetization periods thus contains the approved list of codecs/packetization periods. If the RemoteConnectionDescriptor does not contain any media stream lines, a codec negotiation failure has occurred and an error response is generated (error code 534 – codec negotiation failure – is recommended). If the RemoteConnectionDescriptor contains multiple media streams, the MTA SHOULD only accept one of these and reject the others by setting their port to zero in the LocalConnectionDescriptor. If the RemoteConnectionDescriptor was provided but the packetization period(s) was omitted, the negotiated list of packetization periods contains the set of packetization periods from the approved list. The MTA MUST choose reasonable defaults [4] if the packetization period is explicitly omitted from both the LocalConnectionOptions and the RemoteConnectionDescriptor.
4. If the negotiated list of codecs/packetization periods is empty, a codec negotiation failure has occurred and an error response is generated (error code 534 – codec negotiation failure – is recommended).
5. Otherwise, codec negotiation has succeeded, and the negotiated list of codecs/packetization periods is returned in the LocalConnectionDescriptor.

Note that both LocalConnectionOptions and the RemoteConnectionDescriptor can contain a list of codecs ordered by preference. When both are supplied, the gateway should adhere to the preferences provided in the LocalConnectionOptions. It should be noted, that the above procedure negotiates both encoding methods and packetization periods as opposed to just encoding methods. This is done to enable consistent local and far-end QoS operation in the segmented QoS model used in PacketCable.

In the case that a gateway supports more than one codec per endpoint, there are two options the gateway can use in deciding how many codecs it wants to support for that connection:

1. Gateway supports multiple codecs and can switch between different codecs in real-time. The gateway returns all negotiated codecs in the SDP media stream line and reserves the Least-Upper-Bound (LUB) as per the DQoS specification [20]. The LUB is reserved to guarantee that a switch to any of these codecs will succeed. Multiple codecs in the m= line means the device must be ready to receive media packets from any of the negotiated codecs. As well, the gateway may send media packets from any of the negotiated codecs and switch between them as required.
2. Gateway supports one or more codecs but cannot switch between different codecs in real-time. The gateway therefore negotiates and returns only one codec in the SDP media stream line, (optionally, gateway also puts additional supported codecs in the SDP ‘X-pc-codecs’ attribute) and reserves the bandwidth for the single negotiated codec in the media stream line as per the DQoS specification. With this method, a codec change must be initiated by the CMS in order to change codecs at which time the resulting change in bandwidth is re-established as per the DQoS specification.

5 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 commands:

- 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 can also send a DeleteConnection as defined in Section 4.3.6. The Call Agent can send either of the Audit commands to the gateway and, finally, the gateway can send a RestartInProgress command to the Call Agent.

5.1 General Description

All commands are composed of a Command header, which for some commands may be followed by a session description.

All responses are composed of a Response header, which for some commands may be 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, optionally, a single line-feed character). The headers are separated from the session description by an empty line.

MGCP uses a transaction identifier with a value between 1 and 999999999 to correlate commands and responses. The transaction identifier is encoded as a component of the command header and is repeated as a component of the response header.

5.2 Command Header

The command header is composed of:

- A command line identifying the requested action or verb, the transaction identifier, the endpoint toward 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.

5.2.1 Command Line

The command line is composed of:

- The name of the requested verb,
- The identification of the transaction,
- The name of the endpoint(s) that should execute the command (in notifications or restarts, the name of the endpoint(s) 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. Embedded clients SHOULD use exactly one ASCII space separator, however they MUST be able to parse messages with additional white space characters.

5.2.1.1 Requested Verb Coding

Requested verbs are encoded as four letter upper- and/or lower-case ASCII codes (comparisons MUST be case insensitive) as defined in Table 6 below:

Table 6 - Requested Verb Codings

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

New verbs may be defined in future versions of the protocol. It may be necessary, for experimental 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 (e.g., XPER).

An embedded client that receives a command with an experimental verb it does not support MUST return an error (error code 511 – unrecognized extension).

5.2.1.2 Transaction Identifiers

Transaction identifiers are used to correlate commands and responses.

An embedded client 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 embedded client MUST be unique for the maximum lifetime of the transactions within the collection of Call Agents that control that embedded client (see Section 5.5). Thus, regardless of the sending Call Agent, embedded clients 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 embedded client MUST be unique for the maximum lifetime of the transactions (see Section 5.5) regardless of which Call Agent the command is sent to. Thus, a Call Agent can always detect a duplicate transaction from an embedded client by the combination of the domain-name of the endpoint and the transaction identifier. The embedded client in turn can always detect a duplicate response acknowledgement by looking at the transaction id(s).

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. Transaction identifiers should not use any leading zeroes. Equality is based on numerical value and leading zeroes are ignored. 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.

5.2.1.3 Endpoint, Call Agent and NotifiedEntity Name Coding

The endpoint names and Call Agent names are encoded as e-mail addresses, as defined in [22]. 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. Both components MUST be case insensitive.

Examples of such names are shown in Table 7 below:

Table 7 - Example Name Coding

aaln/1@ncs2.whatever.net	Analog access line 1 in the embedded client ncs2 in the "whatever" network.
Call-agent@ca.whatever.net	Call Agent for the "whatever" network.

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

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

In case the port number is omitted, the default MGCP Call Agent port (2727, unless provisioned otherwise) will be used. Additional detail on endpoint names can be found in Section 4.1.1.

5.2.1.4 Protocol Version Coding

The protocol version is coded as the keyword "MGCP" followed by a white space and the version number, which again is followed by the profile name "NCS" and a profile version number. The version numbers are composed of a major version number, a dot, and a minor version number. The major and minor version numbers are coded as decimal numbers. The profile version number defined by this specification is 1.0. The protocol version for this specification MUST be encoded as:

```
MGCP 1.0 NCS 1.0
```

The "NCS 1.0" portion signals that this is the NCS 1.0 profile of MGCP 1.0.

An entity that receives a command with a protocol version it does not support, MUST respond with an error (error code 528 – Incompatible Protocol Version).

5.2.2 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. Parameter names and values are still case-insensitive though. The parameters that can be present in commands are defined in Table 8 below:

Table 8 - Parameter Definitions

Parameter name	Code	Parameter value
ResponseAck[25]	K	See description.
CallId	C	Hexadecimal string, MUST NOT exceed 32 characters. Call Identifiers are compared as strings rather than numerical values.
ConnectionId	I	Hexadecimal string, MUST NOT exceed 32 characters. Connection Identifiers are compared as strings rather than numerical values.
NotifiedEntity	N	An identifier, in RFC 2821 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.whatever.net:5234
RequestIdentifier	X	Hexadecimal string, length MUST NOT exceed 32 characters.
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	See description.
SpecificEndPointId	Z	An identifier, in RFC 2821 format, composed of an arbitrary string, optionally followed by an "@" followed by the domain name of the embedded client to which this endpoint is attached.
MaxEndPointIds	ZM	Decimal string, length MUST NOT exceed 16 characters.
NumEndPoints	ZN	Decimal string, length MUST NOT exceed 16 characters.
RequestedInfo	F	See description.
QuarantineHandling	Q	See description.
DetectEvents	T	See description.
EventStates	ES	See description.
ResourceID	DQ-RI	See description.
RestartMethod	RM	See description.
RestartDelay	RD	A number of seconds encoded as a decimal number.
Capabilities	A	See description.
VersionSupported	VS	See description.
MaxMGCPDatagram	MD	See description.

[25] The ResponseAck parameter was not shown in Section 4.3 as transaction identifiers are not visible in our example API. Implementers may choose a different approach.

The parameters are not necessarily present in all commands. Table 9 provides the association between parameters and commands. The letter M stands for mandatory, O for optional, and F for forbidden:

Table 9 - Association of Parameters with Commands

Parameter name	CRCX	MDCX	DLCX	RQNT	NTFY	AUEP	AUCX	RSIP
ResponseAck[25]	O	O	O	O	O	O	O	O
CallId	M	M	O	F	F	F	F	F
ConnectionId	F	M	O	F	F	F	M	F
RequestIdentifier	O	O	O	M	M	F	F	F
LocalConnectionOptions	O	O	F	F	F	F	F	F
Connection Mode	M	O	F	F	F	F	F	F
RequestedEvents	O*	O*	O*	O*	F	F	F	F
SignalRequests	O*	O*	O*	O*	F	F	F	F
NotifiedEntity	O	O	O	O	O	F	F	F
ReasonCode	F	F	O	F	F	F	F	O
ObservedEvents	F	F	F	F	M	F	F	F
DigitMap	O	O	O	O	F	F	F	F
Connection parameters	F	F	O	F	F	F	F	F
Specific Endpoint Id	F	F	F	F	F	O	F	F
MaxEndPointIds	F	F	F	F	F	O	F	F
NumEndPoints	F	F	F	F	F	F	F	F
RequestedInfo	F	F	F	F	F	O	O	F
QuarantineHandling	O	O	O	O	F	F	F	F
DetectEvents	O	O	O	O	F	F	F	F
EventStates	F	F	F	F	F	F	F	F
ResourceID	F	F	F	F	F	F	F	F
RestartMethod	F	F	F	F	F	F	F	M
RestartDelay	F	F	F	F	F	F	F	O
Capabilities	F	F	F	F	F	F	F	F
VersionSupported	F	F	F	F	F	F	F	F
MaxMGCPDatagram	F	F	F	F	F	F	F	F
RemoteConnection Descriptor	O	O	F	F	F	F	F	F

* The RequestedEvents and SignalRequests parameters are optional in the NotificationRequest. If these parameters are omitted, the corresponding lists will be considered empty. For the connection handling commands, omission of these two parameters when the command includes a RequestIdentifier means the corresponding lists will be considered empty.

Embedded clients and Call Agents SHOULD always provide mandatory parameters before optional ones, however embedded clients MUST NOT fail if this recommendation is not followed.

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

```
X-FlowerOfTheDay: Daisy
```

Parameter names that start with "X+" are mandatory parameter extensions. A gateway that receives a mandatory parameter extension that it cannot understand MUST respond with an error (error code 511 – unrecognized extension).

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

It should be noted that experimental verbs are of the form X*ABC*, whereas experimental parameters are of the form X-ABC.

If a parameter line is received with a forbidden parameter, or any other formatting error, the receiving entity should respond with the most specific error code for the error in question. The least specific error code is 510 – protocol error. Commentary text can always be provided.

5.2.2.1 Response Acknowledgement

The response acknowledgement parameter[25] is used to support the three-way handshake described in Section 5.7. It contains a comma separated list of "confirmed transaction-id ranges".

Each "confirmed transaction-id range" 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 a response acknowledgement is:

```
K: 6234-6255, 6257, 19030-19044
```

5.2.2.2 RequestIdentifier

The request identifier correlates a Notify command with the NotificationRequest that triggered it. A RequestIdentifier is a hexadecimal string; length MUST not exceed 32 characters. RequestIdentifiers are compared as strings rather than numerical values. The string "0" is reserved for reporting of persistent events in the case where no NotificationRequest has been received yet (see Section 4.3.2).

5.2.2.3 Local Connection Options

The local connection options describe the operational parameters that the Call Agents instructs the gateway to use for a connection. These parameters are:

- The packetization period in milliseconds, encoded as the keyword "p" followed by a colon and a decimal number.
- The multiple packetization period in milliseconds for each codec in the encoding method LCO, encoded as the keyword "mp" followed by a colon and a list of decimal numbers or hyphens, with one entry for each entry in the Encoding Method Field. Each packetization period value is separated from its successor by a single semicolon. The first entry in the list MUST be a decimal number. Subsequent entries in the list MUST be either a decimal number or a hyphen.
- The literal name of the compression algorithm as specified in the Packet Cable Audio/Video Codec Specification [19], encoded as the keyword "a" followed by a colon and a character string. These literal names MUST be used and are equivalent to the codec definitions in RTP Parameters [17]. It is RECOMMENDED that other well-known variants of the literal codec names be supported as well.
- The echo-cancellation parameter, encoded as the keyword "e" 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 silence suppression parameter, encoded as the keyword "s" followed by a colon and the value "on" or "off".

The LocalConnectionOptions parameters used for Dynamic Quality of Service are:

- The D-QoS GateID encoded as the keyword "dq-gi" followed by a colon and a string of up to 8 hex characters corresponding to a 32 bit identifier for the GateID.
- The D-QoS Resource Reservation parameter encoded as the keyword "dq-rr" followed by a colon and a character string. A list of values may be specified in which case the values will be separated by a semicolon. The possible values are shown in Table 10 below:

Table 10 - DQoS Resource Reservation Parameter Values

Mode	Meaning
sendresv	Reserve in the send direction only.
recvresv	Reserve in the receive direction only.
snrcresv	Reserve in the send and receive direction.
sendcomt	Commit in the send direction only.
recvcomt	Commit in the receive direction only.
snrccomt	Commit in the send and receive direction.

- The ResourceID encoded as the keyword "dq-ri" followed by a colon and a string of up to 8 hex characters corresponding to a 32 bit identifier for the ResourceID.
- The ReserveDestination is encoded as the keyword "dq-rd" followed by a colon and an IP-address encoded similarly to an IP-address for the domain name portion of an endpoint name. The ReserveDestination may optionally be followed by a colon and up to 5 decimal characters for a UDP port number to use.

The LocalConnectionOptions parameters used for Security are encoded as follows:

- The RTP ciphersuite is encoded as the keyword "sc-rtp" followed by a colon and an RTP ciphersuite string as defined below. A list of values may be specified in which case the values MUST be separated by a single semicolon.
- The RTCP ciphersuite is encoded as the keyword "sc-rtcp" followed by a colon and an RTCP ciphersuite string as defined below. A list of values may be specified in which case the values MUST be separated by a single semicolon.

The RTP and RTCP ciphersuite strings follow the grammar:

ciphersuite = [AuthenticationAlgorithm] "/" [EncryptionAlgorithm]
AuthenticationAlgorithm = 1*(ALPHA / DIGIT / "-" / "_")
EncryptionAlgorithm = 1*(ALPHA / DIGIT | "-" / "_")

where ALPHA, and DIGIT are defined in [26].

Whitespaces MUST NOT be sent within a ciphersuite, or between adjacent ciphersuites when multiple ciphersuites are provided. The following example illustrates the formatting of a ciphersuite and a ciphersuite list:

sc-rtp:62/51;64/51;60/50

The actual list of PacketCable supported ciphersuites is provided in the PacketCable Security Specification [21].

When several parameters are present, the values are separated by a comma. It MUST be considered an error to include a parameter without a value (error code 524 – LocalConnectionOptions inconsistency.)

Examples of local connection options are:

```
L: p:10, a:PCMU
L: p:10, a:PCMU, e:off, t:20, s:on
L: p:30, a:G729, e:on, t:A0, s:off
```

The type of service hex value “20” implies an IP precedence of 1, and a type of service hex value of “A0” implies an IP precedence of 5.

This set of attributes may be extended by extension attributes. Extension attributes are composed of an attribute name, followed by a colon, and a semicolon separated list of attribute values. The attribute name MUST start with the two characters "x+", for a mandatory extension, or "x-", for a non mandatory extension. If a gateway receives a mandatory extension attribute that it does not recognize, it MUST reject the command with an error (error code 525 – Unknown extension in LocalConnectionOptions).

5.2.2.4 Capabilities

Capabilities inform the Call Agent about the endpoint’s 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:

- The packetization period in milliseconds, encoded as the keyword “p” followed by a colon and a decimal number. A range may be specified as two decimal numbers separated by a hyphen.
- The literal name of the compression algorithm, encoded as the keyword “a” followed by a colon and a character string. The literal names defined in the PacketCable Audio/Video Codecs Specification [19] MUST be used. A list of values may be specified in which case the 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. A range may be specified as two decimal numbers separated by a hyphen.
- The echo-cancellation parameter, encoded as the keyword “e” followed by a colon and the value “on” if echo cancellation is supported, “off” otherwise.
- The type of service parameter, encoded as the keyword “t” followed by a colon and the value “0” if type of service is not supported, all other values indicate support for type of service.
- The silence suppression parameter, encoded as the keyword “s” followed by a colon and the value “on” if silence suppression is supported, “off” otherwise.[26]
- The event packages supported by this endpoint encoded as the keyword “v” followed by a colon and then a semicolon-separated list of package names supported. The first value specified will be the default package for the endpoint.
- The connection modes supported by this endpoint encoded as the keyword “m” followed by a colon and a semicolon-separated list of connection modes supported as defined in Section 5.2.2.7.
- The keyword “dq-gi” if Dynamic Quality of Service is supported.

[26] Silence suppression on is equivalent to VAD enable and silence suppression off is equivalent to VAD disable.

- The keyword “sc-rtp” followed by a colon and a semi-colon separated list of RTP ciphersuites, using the same encoding as in the LocalConnectionOptions.
- The keyword “sc-rtcp” followed by a colon and a semi-colon separated list of RTCP ciphersuites, using the same encoding as in the LocalConnectionOptions.

When several parameters are present, the values are separated by a comma.

Examples of capabilities are:

```
A: a:PCMU;p:10-30, e:on, s:off, v:L;S,
  m:sendonly;recvonly;sendrecv;inactive

A: a:G729; p:10-20, e:on, s:off, v:L;S,
  m:sendonly;recvonly;sendrecv;inactive

A: a:G729; p:30-90, e:on, s:on, v:L;S,
  m:sendonly;recvonly;sendrecv;inactive;confrnce,
  dq-gi, sc-rtp: 64/51;60/51, sc-rtcp: 71/81
```

Note that the codecs and security algorithms are merely examples – separate PacketCable specifications detail the actual codecs and algorithms supported, as well as the encoding used (see [19] and [21]). Note also that each set of capabilities is provided on a single line. The examples above show each set on multiple lines due only to formatting restraints of this document.

5.2.2.5 Connection Parameters

Connection parameters are encoded as a string of type and value pairs, where the type is one of the codes in Table 11 below, and the value is a decimal integer. Types are separated from values by an “=”sign. Parameters are separated from each other by a comma.

Table 11 - Connection Parameters

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.
Remote Packets sent	PC/RPS	The number of packets that were sent on the connection from the perspective of the remote endpoint.
Remote Octets sent	PC/ROS	The number of octets that were sent on the connection from the perspective of the remote endpoint.
Remote Packets lost	PC/RPL	The number of packets that were not received on the connection, as deduced from gaps in the sequence number from the perspective of the remote endpoint.
Remote Jitter	PC/RJI	The average inter-packet arrival jitter, in milliseconds, expressed as an integer number from the perspective of the remote endpoint.

Extension connection parameter names are composed of the string "X-" followed by a two or three letter extension parameter name. Call Agents that receive unrecognized extensions MUST silently ignore these extensions. If an endpoint receives RTCP packets with these statistics, it MUST return the Remote parameters (Rxx above) in the response to the Delete-Connection and Audit-Connection commands.

An example of a connection parameter encoding is:

```
P: PS=1245, OS=62345, PR=0, OR=0, PL=0, JI=0, LA=48, PC/RPS=0, PC/ROS=0,
PC/RPL=0, PC/RJI=0
```

5.2.2.6 Reason Codes

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

```
E: 900 Endpoint malfunctioning
```

A list of reason-codes can be found in Section 4.6.

5.2.2.7 Connection Mode

The connection mode describes the connection's operation mode. The possible values are shown in Table 12 below:

Table 12 - Connection Mode

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 send and receive packets according to conference mode.
M: inactive	The gateway should neither send nor receive packets.
M: replcate	The gateway should only send packets according to replicate mode.
M: netwloop	The gateway should place the endpoint in Network Loopback mode.
M: netwtest	The gateway should place the endpoint in Network Continuity Test mode.

5.2.2.8 Event/Signal Name Coding

Event/signal 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, DetectEvents, ObservedEvents, and EventStates parameters. Each event is identified by an event code. These ASCII encodings are not case sensitive. Values such as "hu", "Hu", "HU" or "hU" should be considered equal.

Table 13 provides examples of event names:

Table 13 - Event Name Examples

L/hu	On-hook transition, in the line package.
L/0	Digit 0 in the Line package.
Hf	Flash-hook, assuming that the line package is the default package for the endpoint.
L/rt@0A3F58	Ringback on connection "0A3F58".

In addition, the range and wildcard notation of events can be used, instead of individual names, in the RequestedEvents and DetectEvents (but not SignalRequests, ObservedEvents, or EventStates): Table 14 provides examples of valid range and wildcard notation.

Table 14 - Event Range and Wildcard Notation

L/[0-9]	Digits 0 to 9 in the Line package.
L/X	Digits 0 to 9 in the Line package.
[0-9*#A-D]	All digits and letters in the Line package (default for endpoint).
L/all	All events in the Line package.

Finally, the star sign can be used to denote "all connections", and the dollar sign can be used to denote the "current" connection. Table 15 provides examples of valid use of the star and dollar sign notations:

Table 15 - "All" and "'Current" Connection Notation

L/ma@*	The RTP media start event on all connections for the endpoint.
L/rt@$	Ringback on the current connection.

An initial set of event packages for embedded clients can be found in Appendix A.

5.2.2.9 RequestedEvents

The RequestedEvents parameter provides the list of events that have been requested. The currently defined event codes are described in Appendix A.

Each event can be qualified by a requested action, or by a list of actions. Not all actions can be combined – please refer to Section 4.3.1 for valid combinations. 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 shown in Table 16 below:

Table 16 - Requested Events Actions

Action	Code
Notify immediately	N
Accumulate	A
Accumulate according to digit map	D
Ignore	I
Keep Signal(s) active	K
Embedded NotificationRequest	E
Embedded ModifyConnection	C

If a digit map is not provided when the “accumulate according to digit map” action is specified, the endpoint simply uses its current digit map. If the endpoint does not have any digit maps currently, an error MUST be returned (error code 519 – no digit map).

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 discarded, except for persistent events.

The digit-map action can only be specified for the digits, letters, and timers.

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

```
R: hu(N), hf(N)         Notify on-hook, notify hook-flash.
R: hu(N), [0-9#T](D)    Notify on-hook, accumulate digits according to digit
                        map.
```

The embedded NotificationRequest follows the format:

```
E ( R( <RequestedEvents>), D( <Digit Map>), S( <SignalRequests>) )
```

with each of R, D, and S being optional and possibly supplied in another order. The following example illustrates the use of Embedded NotificationRequest:

```
R: hd(A, E(S(dl), R( oc(N), [0-9#T](D) ), D((1xxxxxxxxxx|9011x.T)) ) )
```

On off-hook, accumulate the event, provide dial-tone and start accumulating digits according to the digit map supplied. Stop dial-tone when the first digit is input, or, if no digit is input before the dial-tone times out, Notify the operation complete. Otherwise, notify the off-hook and collected digits when a match, mismatch, or inter-digit timeout has occurred. It should be noted, that since on-hook is a persistent event, it will still be detected and notified although it has not been specified here.

The embedded ModifyConnection action follows the format:

```
C(M(<ConnectionMode1>( <ConnectionID1> )) , … ,
               M(<ConnectionModen>(ConnectionIDn )))
```

The following example illustrates the use of Embedded ModifyConnection:

R: hf(A, C(M(inactive(X43DC)), M(sendrecv($)))), oc(N), of(N)

On hook-flash, change the connection mode of connection “X43DC” to “inactive”, and then change the connection mode of the “current connection” to “send receive”. Notify events on “operation complete” and “operation failure”.

5.2.2.10 SignalRequests

The SignalRequests parameter provides the name of the signals that have been requested. The currently defined signals can be found in Appendix A. A given signal can only appear once in the list, and all signals will, by definition, be applied at the same time. The MTA MUST support, at a minimum, a single signal on each endpoint and simultaneously support the generation of one signal on each connection for a given endpoint. Specific packages MAY define requirements beyond these minimum capabilities. For signal combinations beyond this minimum requirement that the MTA does not support, it SHOULD return error code 502.

Some signals can be qualified by signal parameters. When a signal is qualified by multiple signal parameters, the signal parameters are separated by commas. Each signal parameter MUST follow the format specified below (white spaces allowed):

```
signal-parameter = signal-parameter-value |
                   signal-parameter-name “=”signal-parameter-value |
                   signal-parameter-name “(” signal-parameter-list “)”

signal-parameter-list = signal-parameter-value 0*( “,” signal-parameter-value)
```

where signal-parameter-value may be either a string or a quoted string, i.e., a string surrounded by two double quotes. Two consecutive double-quotes in a quoted string will escape a double-quote within that quoted string. For example, `“ab””c”` will produce the string `ab”c.`

Each signal has one of the following signal-types associated with it (see Section 4.3.1):

- On/Off (OO),
- Time-out (TO),
- Brief (BR).

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
```

Time-out signals can be parameterized with the signal parameter "TO" and a time-out value that overrides the default time-out value. If a time-out signal is not parameterized with a time-out value the default time-out value will be used. Both of the following will apply the ringing signal for 6 seconds:

```
rg(to=6000)
rg(to(6000))
```

Individual signals may define additional signal parameters.

The signal parameters will be enclosed within parenthesis, as in (assuming "Line" is the default package):

```
S: ci(10/14/17/26, "555 1212", CableLabs)
```

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

```
S: rg, rt@FDE234C8
```

5.2.2.11 ObservedEvents

The observed events parameters provide the list of events that have been observed. The event codes are the same as those used in the NotificationRequest. When an event is detected on a connection, the observed event will identify the connection the event was detected on using the "@<connection>" syntax. Examples of observed events are:

```
O: hu
O: ma@A43B81
O: 8,2,9,5,5,5,5,T
O: hf,hf,hu
O: 8,2,9,5,mt,5,5,5,T
```

Events that have been accumulated according to digit map, are reported as individual events in the order they were detected. Other events may be mixed in between them. It should be noted that if the "current dial string" is non-empty with a partial match, and another event occurs that results in a Notify message being generated, the partially matched "current dial string" will be included in the list of observed events, and the "current dial string" will then be cleared – please refer to Section 4.4.3.1 for details.

5.2.2.12 RequestedInfo

The RequestedInfo parameter contains a comma separated list of parameter codes, as defined in the "Parameter lines" section – Section 4.3.8 lists the parameters that can be audited. The values listed in Table 17 below are supported as well:

Table 17 - RequestedInfo Parameter Values

RequestedInfo Parameter	Code
LocalConnectionDescriptor	LC
RemoteConnectionDescriptor	RC

For example, if one wants to audit the value of the NotifiedEntity, RequestIdentifier, RequestedEvents, SignalRequests, DigitMap, DetectEvents, EventStates, LocalConnectionDescriptor, and RemoteConnectionDescriptor parameters, the value of the RequestedInfo parameter will be:

```
F: N,X,R,S,D,T,ES,LC,RC
```

The capabilities request, for the AuditEndPoint command, is encoded by the parameter code “A", as in:

```
F: A
```

5.2.2.13 QuarantineHandling

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

- The keyword "process" or "discard" to indicate the treatment of quarantined and observed events. If neither process nor discard is present, process is assumed.
- The keyword "step" or "loop" to indicate whether at most one notification is expected, or whether multiple notifications are allowed. If neither "step" nor "loop" is present, "step" is assumed. Support for these two keywords is mandatory.

The following values are valid examples:

Q: loop

Q: process

Q: loop, discard

5.2.2.14 DetectEvents

The DetectEvents 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.

5.2.2.15 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.

5.2.2.16 ResourceID

The ResourceID parameter is a return parameter used for Dynamic Quality of Service to signal the resource ID assigned for the gate in question. The ResourceID is encoded as a string of up to 8 hex characters, such as for example:

```
DQ-RI: AB345DC
```

5.2.2.17 RestartMethod

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

```
RM: restart
```

5.2.2.18 VersionSupported

The VersionSupported parameter is encoded as a comma separated list of versions supported, such as for example:

```
VS: MGCP 1.0, MGCP 1.0 NCS 1.0
```

5.2.2.19 MaxMGCPDatagram

The MaxMGCPDatagram parameter is encoded as a string of up to nine decimal digits – leading zeroes are not permitted. The following example illustrate the use of this parameter:

```
MD: 8100
```

5.3 Response Header Formats

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

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 optional commentary preceded by a white space, e.g.:

```
200 1201 OK
```

Table 18 below summarizes the response parameters whose presence is mandatory or optional in a response header, as a function of the command that triggered the response. The reader should still study the individual command definitions though as this table only provides summary information. The letter M stands for mandatory, O for optional, and F for forbidden.

Table 18 - Association of Response Header Parameters and Commands

Parameter name	CRCX	MDCX	DLCX	RQNT	NTFY	AUEP	AUCX	RSIP
ResponseAck[25]	O[1]	O[1]	O[1]	O[1]	O[1]	O[1]	O[1]	O[1]
CallId	F	F	F	F	F	F	O	F
ConnectionId	O[2]	F	F	F	F	O	F	F
RequestIdentifier	F	F	F	F	F	O	F	F
LocalConnectionOptions	F	F	F	F	F	O	O	F
Connection Mode	F	F	F	F	F	F	O	F
RequestedEvents	F	F	F	F	F	O	F	F
SignalRequests	F	F	F	F	F	O	F	F
NotifiedEntity	F	F	F	F	F	O	O	O
ReasonCode	F	F	F	F	F	O	F	F
ObservedEvents	F	F	F	F	F	O	F	F
DigitMap	F	F	F	F	F	O	F	F
ConnectionParameters	F	F	O[3]	F	F	F	O	F
Specific Endpoint ID	O	F	F	F	F	O	F	F
MaxEndPointIds	F	F	F	F	F	F	F	F
NumEndPoints	F	F	F	F	F	O	F	F
RequestedInfo	F	F	F	F	F	F	F	F
QuarantineHandling	F	F	F	F	F	F	F	F
DetectEvents	F	F	F	F	F	O	F	F
EventStates	F	F	F	F	F	O	F	F
ResourceID	O	O	F	F	F	F	F	F
RestartMethod	F	F	F	F	F	F	F	F
RestartDelay	F	F	F	F	F	F	F	F
Capabilities	F	F	F	F	F	O	F	F
VersionSupported	F	F	F	F	F	O	F	O
MaxMGCPDatagram	F	F	F	F	F	O	F	F
LocalConnection Descriptor	O[4]	O[4]	F	F	F	F	O	F
RemoteConnection Descriptor	F	F	F	F	F	F	O	F

NOTE:

1. The ResponseAck parameter MUST NOT be used with any other responses than a final response issued after a provisional response for the transaction in question. In that case, the presence of the ResponseAck parameter MUST trigger a Response Acknowledgement message – any ResponseAck values provided will be ignored.

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

3. Connection-Parameters are only valid in a successful response to a non-wildcarded DeleteConnection command sent by the Call Agent

4. A LocalConnectionDescriptor MUST be transmitted with a positive response (code 200) to a CreateConnection. It MUST also be transmitted in response to a ModifyConnection command, if the modification resulted in a change of the Local Connection Descriptor. The LocalConnectionDescriptor is encoded as a "session description", as defined in section 5.4. It is separated from the response header by an empty line.

The response parameters are described for each of the commands in the following.

5.3.1 CreateConnection

In the case of a CreateConnection message, the response line is followed by a Connection-Id parameter with a successful response (code 200). A LocalConnectionDescriptor is furthermore transmitted with a positive response. The LocalConnectionDescriptor is encoded as a “session description”, as defined in Section 5.4. It is separated from the response header by an empty line, e.g.:

```
200 1204 OK
I: FDE234C8

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 96 97 0
a=rtpmap:96 G726-32/8000
a=rtpmap:97 telephone-event/8000
a=mptime: 10 - 10
```

When a provisional response has been issued previously, the final response may furthermore contain the Response Acknowledgement parameter, and when Dynamic Quality of Service is used, the final response may also contain a ResourceID, as in:

```
200 1204 OK
K:
I: FDE234C8
DQ-RI: 23DB4A43

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 96 97 0
a=rtpmap:96 G726-32/8000
a=rtpmap:97 telephone-event/8000
a=mptime: 10 - 10
```

The final response is acknowledged by a Response Acknowledgement:

```
000 1204
```

5.3.2 ModifyConnection

In the case of a successful ModifyConnection message, the response line is followed by a LocalConnectionDescriptor, if the modification resulted in a modification of the session parameters (e.g., changing only the mode of a connection does not alter the session parameters). The LocalConnectionDescriptor is encoded as a "session description", as defined in Section 5.4. It is separated from the response header by an empty line.

```
200 1207 OK

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
a=mptime: 20
```

The response may also contain a ResourceID when Dynamic Quality of Service is used as in:

```
200 1207 OK
DQ-RI: 12345
```

When a provisional response has been issued previously, the final response may furthermore contain the Response Acknowledgement parameter as in:

```
526 1207 No bandwidth
K:
```

The final response is acknowledged by a Response Acknowledgement:

```
000 1207 OK
```

5.3.3 DeleteConnection

Depending on the variant of the DeleteConnection message, the response line may be followed by a Connection Parameters parameter line, as defined in Section 5.2.2.5.

```
250 1210 OK
P: PS=1245, OS=62345, PR=780, OR=45123, PL=10, JI=27, LA=48,
                      PC/RPS=782, PC/ROS=45238, PC/RPL=5, PC/RJI=26
```

5.3.4 NotificationRequest

A NotificationRequest response does not include any additional response parameters.

5.3.5 Notify

A Notify response does not include any additional response parameters.

5.3.6 AuditEndpoint

In the case of an AuditEndPoint the response line may be followed by information for each of the parameters requested–each parameter will appear on a separate line. Parameters for which no value currently exists, e.g., digit map, will still be provided.

Each local endpoint name "expanded" by a wildcard character will appear on a separate line using the "SpecificEndPointId" parameter code, e.g.:

```
200 1200 OK

Z: aaln/1@rgw.whatever.net

Z: aaln/2@rgw.whatever.net
```

An example of a response to an AuditEndPoint message containing a non-wildcarded endpoint name is shown below. Note that the SpecificEndPointId is not provided in this case. Note also that each set of

capabilities is provided on a single line. The example below shows each set on multiple lines due only to formatting restraints of this document.

```
200 1200 OK

A: a:PCMU; p:10, e:on, s:off, t:1, v:L,
      m:sendonly;recvonly;sendrecv;inactive

A: a:G728; p:20, e:on, s:off, t:1, v:L,
      m:sendonly;recvonly;sendrecv;inactive

A: a:G729; p:30, e:on, s:on, t:1, v:L,
      m:sendonly;recvonly;sendrecv;inactive;confrnce
```

5.3.7 AuditConnection

In the case of an AuditConnection, the response may be followed by information for each of the parameters requested. Parameters for which no value currently exists will still be provided. Connection descriptors will always appear last and each will be preceded by an empty line, as for example:

```
200 1203 OK
C: A3C47F21456789F0
N: CA-1@myhost.whatever.net:2345
L: mp:20;10, a:PCMU;G728
M: sendrecv
P: PS=622, OS=31172, PR=390, OR=22561, PL=5, JI=29, LA=50,
                        PC/RPS=391, PC/ROS=22619, PC/RPL=5, PC/RJI=26

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1296 RTP/AVP 96
a=rtpmap:96 G728/8000
a=mptime: 10
```

If both a local and a remote connection descriptor are provided, the local connection descriptor will be the first of the two. 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.

5.3.8 RestartInProgress

The response to a RestartInProgress may include the name of another Call Agent to contact, for instance when the Call Agent redirects the endpoint to another Call Agent as in:

```
521 1204 Redirect
N: CA-1@ca.whatever.net
```

5.4 Session Description Encoding

The session description is encoded in conformance with the session description protocol (SDP), however, embedded clients may make certain simplifying assumptions about the session description as specified in the following. It should be noted, that session descriptions are case sensitive per [4].

SDP usage depends on the type of session, as specified in the “media” parameter:

- If the media is set to “audio”, the session description is for an audio service,
- If the media is set to “video”, the session description is for a video service.

For an audio service, the gateway will consider the information provided in SDP for the “audio” media, and for a video service the gateway will consider the information provided in SDP for the “video” media.

5.4.1 SDP Audio Service Use

In a voice-only gateway, we only have to describe sessions that use exactly one media, audio. The parameters of SDP that are relevant for the voice based application are specified below. Embedded clients MUST support session descriptions that conform to these rules and in the following order:

1. The SDP profile presented below
2. SDP: Session Description Protocol [4]

The CMS should be careful if it decides it is necessary to alter the SDP received from an endpoint. SDP provides a means of communicating the capabilities of an endpoint to another endpoint. If the CMS chooses to modify the SDP, it MUST NOT alter the SDP such that it violates the rules defined in this section.

The SDP profile provided describes the use of the session description protocol in NCS. The general description and explanation of the individual parameters can be found in [4], however below we detail what values NCS endpoints need to provide for these fields (send) and what NCS endpoints should do with values supplied or not supplied for these fields (receive).

5.4.1.1 Protocol Version (v=)

v= <version>
"v= 0"

Send: MUST be provided in accordance with [4] (i.e., v=0).

Receive: MUST be provided in accordance with [4].

5.4.1.2 Origin (o=)

The origin field consists (o=) of 6 sub-fields in [4]:

o= <username> <session-ID> <version> <network-type> <address-type> <address>

"o= - 2987933615 2987933615 IN IP4 126.16.64.4"

Username:

Send: Hyphen MUST be used as username when privacy is requested. Hyphen SHOULD be used otherwise. [27]

Receive: This field SHOULD be ignored.

Session-ID:

Send: MUST be in accordance with [4] for interoperability with non-PacketCable clients.

Receive: This field SHOULD be ignored.

Version:

Send: Must be in accordance with [4].

Receive: This field SHOULD be ignored.

Network Type:

Send: Type 'IN' MUST be used.

Receive: This field SHOULD be ignored.

Address Type:

Send: Type "IP4" MUST be used

[27] Since NCS endpoints do not know when privacy is requested, they SHOULD always use a hyphen.

Receive: This field SHOULD be ignored.

Address:

Send: MUST be in accordance with [4] for interoperability with non-PacketCable clients.

Receive: This field MUST be ignored.

5.4.1.3 Session Name (s=)

s= <session-name>
”s= -“

Send: Hyphen MUST be used as Session name.

Receive: This field MUST be ignored.

5.4.1.4 Session and Media Information (i=)

i= <session-description>

Send: For NCS, the field MUST NOT be used.

Receive: This field MUST be ignored.

5.4.1.5 URI (u=)

u= <URI>

Send: For NCS, the field MUST NOT be used.

Receive: This field MUST be ignored.

5.4.1.6 E-Mail Address and Phone Number (e=, p=)

e= <e-mail-address>
p= <phone-number>

Send: For NCS, the field MUST NOT be used.

Receive: This field MUST be ignored.

5.4.1.7 Connection Data (c=)

The connection data consists of 3 sub-fields:

c= <network-type> <address-type> <connection-address>
“c= IN IP4 10.10.111.11”

Network Type:

Send: Type ‘IN’ MUST be used.

Receive: Type “IN” MUST be present.

Address Type:

Send: Type “IP4” MUST be used

Receive: Type “IP4” MUST be present.

Connection Address:

Send: This field MUST be filled with a unicast IP address at which the application will receive the media stream. Thus a TTL value MUST NOT be present and a “number of addresses” value MUST NOT be present. The field MUST NOT be filled with a fully-qualified domain name instead of an IP address. A non-zero address specifies both the send and receive address for the media stream(s) it covers.

Receive: A unicast IP address or a fully qualified domain name MUST be present. A non-zero address specifies both the send and receive address for the media stream(s) it covers.

5.4.1.8 Bandwidth (b=)

b= <modifier> : <bandwidth-value>
"b= AS : 64"

Send: Bandwidth information is optional in SDP but it SHOULD always be included.[28] When an rtpmap or a non well-known codec[29] is used, the bandwidth information MUST be used.

Receive: Bandwidth information SHOULD be included. If a bandwidth modifier is not included, the receiver MUST assume reasonable default bandwidth values for well-known codecs.

Modifier:

Send: Type 'AS' MUST be used.

Receive: Type "AS" MUST be present.

Bandwidth Value:

Send: The field MUST be filled with the Maximum Bandwidth requirement of the Media stream in kilobits per second.

Receive: The maximum bandwidth requirement of the media stream in kilobits per second MUST be present. Refer to the PacketCable Codec specification [19], for the details of calculating the bandwidth value.

5.4.1.9 Time, Repeat Times and Time Zones (t=, r=, z=)

t= <start-time> <stop-time>
"t= 36124033 0"
r= <repeat-interval> <active-duration> <list-of-offsets-from-start-time>
z= <adjustment-time> <offset>

Send: Time MUST be present; start time MAY be zero, but SHOULD be the current time, and stop time SHOULD be zero. Repeat Times, and Time Zones SHOULD NOT be used, if they are used it should be in accordance with [4].

Receive: If any of these fields are present, they SHOULD be ignored.

5.4.1.10 Encryption Keys:

k= <method>
k= <method> : <encryption-keys>

Security services for PacketCable are defined by the PacketCable Security specification [21]. The security services specified for RTP and RTCP do not comply with those of [2], [3], and [4]. In the interest of interoperability with non-PacketCable devices, the "k" parameter will therefore not be used to convey security parameters.

Send: MUST NOT be used.

Receive: This field SHOULD be ignored.

[28] If this field is not used, the Gate Controller might not authorize the appropriate bandwidth.

[29] A non well-known codec is a codec not defined in the PacketCable codec specification [21].

5.4.1.11 Attributes (a=)

a= <attribute> : <value>
a= rtpmap : <payload type> <encoding name>/<clock rate> [/<encoding parameters>]
a= rtpmap : 0 PCMU / 8000
a= fmtp:<format><format specific parameters>
a= X-pc-codecs: <alternative 1> <alternative 2> …
a= mptime: <alternative 1> <alternative 2 > …
a= X-pc-secret: <method>:<encryption key> [pad]
a= X-pc-csuites-rtp: <alternative 1> <alternative 2> …
a= X-pc-csuites-rtcp: <alternative 1> <alternative 2> …
a= X-pc-nrekey: <value>=<attribute>
a= recvonly
a= sendrecv
a= sendonly
a= ptime

Send: One or more of the "a" attribute lines specified below MAY be included.

Receive: One or more of the "a" attribute lines specified below MAY be included and MUST be acted upon accordingly.

rtpmap:

Send: When used, the field MUST be used in accordance with [4]. This field MAY be used for well known as well as non well-known codecs. The encoding names used are provided in a separate PacketCable specification (see [19]).The mapping of codec to RTP dynamic payload type given with this attribute defines the payload type that this sender is prepared to receive on the connection. It also provides a strong hint to the other party that it should also use this payload mapping for its receive side, although there may be cases where this is not possible. On a given connection, once an MTA has mapped a dynamic payload type to a given encoding method for its receive media stream, that payload type MUST NOT subsequently be mapped to another encoding method for its receive media stream.

Receive: When used, the field MUST be used in accordance with [4]. This attribute defines the mapping of codec to RTP payload type that the other side of the connection is prepared to receive. MTAs MUST, therefore, use this payload type mapping when transmitting media on this connection. When received in a CreateConnection command, the MTA SHOULD use this payload type mapping for its own receive side (i.e., it should return a Local Connection Descriptor containing the same rtpmap attribute). If an MTA receives a rtpmap attribute in a ModifyConnection command with a different mapping, the MTA MUST leave its own receive payload type mapping as is (so that asymmetric payload types are used).

fmtp:

Send: This field MAY be used to provide parameters specific to a particular format. For example, the field could be used to describe telephone events supported for an [27] format. When used, the format MUST be one of the formats specified for the media. The parameters specified are provided in a separate specification that details the usage of the format.

Receive: When used, the field MUST be used in accordance with [4].

X-pc-codecs:

This attribute is a media-level attribute defined by PacketCable.

Send: The field contains a list of alternative codecs that the endpoint is capable of using for this connection. The list is ordered by decreasing degree of preference, i.e., the most

preferred alternative codec is the first one in the list. A codec is encoded similarly to "encoding name" in rtpmap.

Receive: Conveys a list of codecs that the remote endpoint is capable of using for this connection. The codecs MUST NOT be used until signaled through a media (m=) line.

mptime:

This attribute is a media-level attribute defined by PacketCable. The mptime attribute defines a list of packetization period values the endpoint is capable of using (sending and receiving) for this connection.

Send: The mptime attribute MUST be present. There MUST be precisely one entry in the list for each <format> entry provided in the "m=" line. Entry number j in this list defines the packetization period for entry number j in the "m=" line. The first entry in the list MUST be a decimal number whereas subsequent entries in the list MUST be either a decimal number or a hyphen. For those media formats where a single packetization rate does not apply (e.g., non- voice codecs such as telephone-event or comfort noise), a hyphen ("-") MUST be encoded at the corresponding location in the list of packetization periods.

Receive: Conveys the list of packetization periods that the remote endpoint is capable of using for this connection; one for each media format in the "m=" line. For media formats whose packetization period is specified as a hyphen ("-"), the endpoint MUST use one of the packetization periods that was actually specified in the list. If the "mptime" attribute is absent, then the value of the "ptime" attribute, if present, MUST be taken as indicating the packetization period for all codecs present in the "m=" line.

X-pc-secret:

This attribute is a media-level attribute defined by PacketCable.

Send: The field contains an end-to-end secret and (possibly) the PAD to be used for RTP and RTCP security. The secret and pad are encoded similarly to the encryption key (k=) parameter of [4] with the following constraints:

- The encryption key MUST NOT contain a ciphersuite, only a passphrase.
- The <method> specifying the encoding of the pass-phrase MUST be either "clear" or "base64" as defined in [25], except for the maximum line length which is not specified here. The method "clear" MUST NOT be used if the secret or pad contains any characters that are prohibited in SDP.

The requirements for when to transmit PAD are described in [21]. If present, it MUST be separated by at least one whitespace from the secret. Pad and secret MUST use the same encode method.

Receive: Conveys the end-to-end secret and PAD to be used for RTP and RTCP security. If present, its use is as described in [21] and it MUST be separated by at least one white space from the secret. PAD and secret MUST use the same encode method.

X-pc-csuites-rtp:

X-pc-csuites-rtcp:

These attributes are media-level attributes defined by PacketCable.

Send: The field contains a list of ciphersuites that the endpoint is capable of using for this connection (respectively RTP and RTCP). The first ciphersuite listed is what the endpoint is currently expecting to use. Any remaining ciphersuites in the list represent alternatives ordered by decreasing degree of preference, i.e. the most preferred alternative ciphersuite is the second one in the list. A ciphersuite is encoded as specified below:

ciphersuite = [AuthenticationAlgorithm] "/" [EncryptionAlgorithm]

AuthenticationAlgorithm = 1*(ALPHA / DIGIT / "-" / "_")

EncryptionAlgorithm = 1*(ALPHA / DIGIT / "-" / "_")

where ALPHA, and DIGIT are defined in [26]. Whitespaces are not allowed within a ciphersuite. The following example illustrates the use of ciphersuite: 62/51

The actual list of ciphersuites is provided in the PacketCable Security Specification [21].

Receive: Conveys a list of ciphersuites that the remote endpoint is capable of using for this connection. Any other ciphersuite than the first in the list cannot be used until signaled through a new ciphersuite line with the desired ciphersuite listed first.

recvonly:

Send: The field MUST be used in accordance with [6]. Currently, this attribute should not be supplied by an embedded client.

Receive: The field MUST be used in accordance with [6].

sendrecv:

Send: The field MUST be used in accordance with [6]

Receive: The field MUST be used in accordance with [6].

sendonly:

Send: The field MUST be used in accordance with [6], except that the IP address and port number MUST NOT be zeroed. Currently, this attribute should not be supplied by an embedded client.

Receive: The field MUST be used in accordance with [6].

ptime:

Send: The ptime SHOULD be sent if it was received in a Remote Connection Descriptor or if the CMS used the packetization period ('p:') LocalConnectionOption.

Receive: The field MUST be ignored if the SDP contains the "mptime" attribute (as required in PacketCable compliant devices). If the "mptime" attribute is not present, then this field is used to define the packetization interval for all codecs present in the SDP description and the MTA MUST use the ptime in the calculation of QoS reservations.

X-pc-nrekey:

This attribute is a media-level attribute defined by PacketCable.

Send: The field contains a 16 bit, integer, counter for the number of rekey events. This field may be required when voice security is used. Requirements for its usage are defined in [21].

Receive: Conveys the number of rekey events. The field may be present when RTP security is used and its use is as defined in [21].

5.4.1.12 Media Announcements (m=)

Media Announcements (m=) consists of 4 sub-fields:

m= <media> <port> <transport> <fmt list>
"m= audio 3456 RTP/AVP 0 97"

Media:

Send: The 'audio' media type MUST be used.

Receive: The type received MUST be 'audio'.

Port:

Send: MUST be filled in accordance with [4]. The port specified is the receive port, regardless of whether the stream is unidirectional or bi-directional. The sending port may be different.

Receive: MUST be used in accordance with [4]. The port specified is the receive port. The sending port may be different.

Transport:

Send: The transport protocol 'RTP/AVP' MUST be used.

Receive: The transport protocol MUST be 'RTP/AVP'.

Media Formats:

Send: Appropriate media type as defined in [4] MUST be used. Specifically, this field contains a list of one or more RTP payload types that this MTA is prepared to receive on the connection and that it would prefer to send with. Each payload type is mapped uniquely to a codec, either statically or dynamically. The static mapping SHOULD be used if available (e.g., 0 for PCMU, 8 for PCMA). If a dynamic payload mapping is used, an RTPMAP attribute MUST also be present and the guidelines in Section 5.4.1.11 MUST be followed.

Receive: In accordance with **[4]**. Specifically this indicates the payload type(s) that the other side of this connection is prepared to receive.

5.4.2 SDP Video Service Use

Details on SDP use for video service will be provided in a future version of this document.

5.5 Transmission Over UDP

5.5.1 Reliable Message Delivery

MGCP messages are transmitted over UDP. Commands are sent to one of the IP addresses defined in the Domain Name System (DNS) for the specified endpoint or Call Agent. The responses are sent back to the source address of the command. However, it should be noted that the response may, in fact, come from another IP address than the one to which the command was sent.

When no port is provisioned for the endpoint,[30] the commands MUST be sent to the default MGCP port, which is 2427 for commands sent to Gateways and 2727 for commands sent to Call Agent. To minimize backward compatibility issues it is RECOMMENDED that the Call Agent always explicitly state the MGCP port to use in NCS messages (and not rely on the default). MGCP messages, 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 sent to recent transactions, i.e., a list of all the responses sent over the last Tt_{hist} seconds, as well as a list of the transactions that are being executed currently. Transaction identifiers of incoming commands are compared to transaction identifiers of the recent responses. If a match is found, the MGCP entity does not execute the transaction, but simply repeats the response. If no match is found, the MGCP entity examines the list of currently executing transactions. If a match is found, the MGCP entity will not execute the transaction: If the command is a CreateConnection or ModifyConnection command a provisional response is sent, otherwise the command is simply ignored.

It is the responsibility of the requesting entity to provide suitable timeouts for all outstanding commands and to retry commands when timeouts have been exceeded. A retransmission strategy is specified in Section 5.5.2.

Furthermore, when repeated commands fail to get a response, the destination entity is assumed to be unavailable. It is the responsibility of the requesting entity to seek redundant services and/or clear existing or pending connections as specified in Section 4.4.

5.5.2 Retransmission Strategy

This specification avoids specifying any static values for the retransmission timers since these values are typically network-dependent. Normally, the retransmission timers should estimate the timer by measuring the time spent between sending a command and the return of a response. Embedded clients MUST implement a retransmission strategy using exponential back-off with configurable initial and maximum retransmission timer values.

Embedded clients SHOULD use the algorithm implemented in TCP/IP, which uses two variables (see, e.g., [16]):

- 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, RTO, in TCP, is set to the sum of the average delay plus N times the average deviation, where N is a constant.

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 (RTO) to the minimum of:
  - the sum of that random value and N times the average deviation.
  - RTO_{max}, where the default value for RTO_{max} is 4 seconds.

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 subject to the needs of real-time communication. Because it includes a random component, it will break the potential synchronization between notifications triggered by the same external event.

[30] Each endpoint may be provisioned with a separate Call Agent address and port.

The initial value used for the retransmission timer is 200 milliseconds by default and the maximum value for the retransmission timer is 4 seconds by default. These default values may be altered by the provisioning process.

5.5.3 Maximum Datagram Size, Fragmentation and Reassembly

MGCP messages being transmitted over UDP rely on IP for fragmentation and reassembly of large datagrams. The maximum theoretical size of an IP datagram is 65535 bytes. With a 20-byte IP header, and an 8-byte header, this leaves us with a maximum theoretical MGCP message size of 65507 bytes when using UDP.

However, IP does not require a host to receive IP datagrams larger than 576 bytes [23] which would provide an unacceptably small MGCP message size. Consequently, MGCP mandates that implementations MUST support MGCP datagrams up to at least 4000 bytes, which requires the corresponding IP fragmentation and reassembly to be supported. Note, that the 4000 byte limit applies to the MGCP level. Lower layer overhead will require support for IP datagrams that are larger than this: UDP and IP overhead will be at least 28 bytes, and e.g., IPsec will add more as well.

It should be noted, that the above applies to both Call Agents and endpoints. Call Agents can audit endpoints to determine if they support larger MGCP datagrams than specified above. Endpoints do currently not have a similar capability to determine if a Call Agent supports larger MGCP datagram sizes.

5.6 Piggy-Backing

There are cases when a Call Agent will want to send several messages at the same time to one or more endpoints in a gateway and vice versa. When several messages have to be sent in the same UDP packets, they are separated by a line of text that contains a single dot, as in for example:

```
200 2005 OK
.
DLCX 1244 aaln/2@rgw.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
```

The piggy-backed messages MUST be processed as if they had been received one at a time in several separate datagrams. Each message in the datagram must be processed to completion and in order starting with the first message, and each command MUST be responded to.

Errors encountered in a message that was piggybacked MUST NOT affect any of the other messages received in that packet – each message is processed on its own.

Piggy-backing can be used to achieve two things:

- Guaranteed in-order delivery and processing of messages.
- Fate sharing of message delivery.

When piggy-backing is used to guarantee in-order delivery of messages, entities MUST ensure that this in-order delivery property is retained on retransmissions of the individual messages. An example of this is when multiple Notify(s) are sent using piggy-backing (as described in Section 4.4.3.1).

Fate sharing of message delivery ensures that either all the messages are delivered, or none of them are delivered. When piggy-backing is used to guarantee this fate-sharing, entities MUST also ensure that this property is retained upon retransmission. For example, upon receiving a Notify from an endpoint operating in lockstep mode, the Call Agent may wish to send the response and a new NotificationRequest command in a single datagram to ensure message delivery fate-sharing of the two.

5.7 Transaction Identifiers And Three Ways Handshake

Transaction identifiers are integer numbers in the range from 1 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 any call agent sent to a particular gateway within a period of Tt_{hist} seconds. Gateways can simply detect duplicate transactions by looking at the transaction identifier only.

The Response Acknowledgement parameter can be found in any command. It carries a set of "confirmed transaction-id ranges" for final responses received – provisional responses MUST NOT be confirmed.

MGCP gateways may choose to delete the copies of the responses to transactions whose id is included in "confirmed transaction-id ranges" received in a message, however the fact that the transaction was executed MUST still be retained for Tt_{hist} seconds. Also, when a Response Acknowledgement message[31] is received, the response that is being acknowledged by it can be deleted. Gateways should silently discard further commands from that Call Agent when the transaction-id falls within these ranges, and the response was issued less than Tt_{hist} seconds ago.

Let $term_{new}$ and $term_{old}$ be the endpoint-name in respectively a new command, cmd_{new}, and some old command. cmd_{old}. The transaction-ids to be confirmed in cmd_{new} SHOULD then be determined as follows:

1. If $term_{new}$ does not contain any wildcards:
    a. Unconfirmed responses to old commands where $term_{old}$ equals $term_{new}$.
    b. Optionally, one or more unconfirmed responses where $term_{old}$ contained the "any-of" wildcard, and the endpoint-name returned in the response was $term_{new}$.
    c. Optionally, one or more unconfirmed responses where $term_{old}$ contained the "all" wildcard, and $term_{new}$ is covered by the wildcard in $term_{old}$.
    d. Optionally, one or more unconfirmed responses where $term_{old}$ contained the "any-of" wildcard, no endpoint-name was returned, and $term_{new}$ is covered by the wildcard in $term_{old}$.
2. If $term_{new}$ contains the "all" wildcard:
    a. Optionally, one or more unconfirmed responses where $term_{old}$ contained the "all" wildcard, and $term_{new}$ is covered by the wildcard in $term_{old}$.
3. If $term_{new}$ contains the "any of" wildcard:
    a. Optionally, one or more unconfirmed responses where $term_{old}$ contained the "all" wildcard, and $term_{new}$ is covered by the wildcard in $term_{old}$ if the "any of" wildcard in $term_{new}$ was replaced with the "all" wildcard.

[31] As opposed to a command with a Response Acknowledgement parameter.

A given response SHOULD NOT be confirmed in two separate messages.

The following examples illustrate the use of these rules:

- If $term_{new}$ is "aaln/1" and $term_{old}$ is "aaln/1" then the old response can be confirmed per rule 1a.
- If $term_{new}$ is "aaln/1" and $term_{old}$ is "*" then the old response can be confirmed per rule 1c.
- If $term_{new}$ is "aaln/*" and $term_{old}$ is "*" then the old response can be confirmed per rule 2a.
- If $term_{new}$ is "aaln/$" and $term_{old}$ is "aaln/*" then the old response can be confirmed per rule 3a.

The "confirmed transaction-id ranges" values SHOULD NOT be used if more than Tt_{hist} 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.

Also, a response SHOULD NOT be confirmed if the response was received more than Tt_{hist} seconds ago.

Messages that confirm responses may be transmitted and received in disorder. The gateway shall retain the union of the confirmed transaction-ids received in recent commands.

5.8 Provisional Responses

In some cases, transaction completion times may be significantly longer than otherwise[32]. NCS uses UDP as the transport protocol and reliability is achieved by selective time-out based retransmissions where the time-out is based on an estimate of the sum of the network roundtrip time and transaction completion time. Significant variance in the transaction completion time is therefore problematic when rapid message loss detection without excessive overhead is desired.

In order to overcome this problem, a provisional response MUST therefore be issued if the transaction completion time is expected to exceed a small period of time (200ms is RECOMMENDED). The provisional response acknowledges the receipt of the command although the outcome of the command may not yet be known, e.g., due to a pending resource reservation. As a guideline, a transaction that requires external communication to complete, e.g., network resource reservation, should issue a provisional response. Furthermore, if a duplicate CreateConnection or ModifyConnection command is received, and the transaction has not yet finished executing, a provisional response MUST then be sent back.

Pure transactional semantics would imply, that provisional responses should not return any other information than the fact that the transaction is currently executing, however an optimistic approach allowing some information to be returned enables a reduction in the delay that would otherwise be incurred in the system.

Provisional responses MUST only be sent in response to a CreateConnection or ModifyConnection command. In order to reduce the delay in the system, a connection identifier and session description MUST be included in the provisional response to the CreateConnection command. If a session description will be returned by the ModifyConnection command, the session description MUST be included in the provisional response here as well. If the transaction completes successfully, the information returned in the provisional response MUST be repeated in the final response. It is considered a protocol error not to repeat this information or to change any of the previously supplied information in a successful response. If the transaction fails, an error code is returned – the information returned previously is no longer valid.

A currently executing CreateConnection or ModifyConnection transaction MUST be cancelled if a DeleteConnection command for the endpoint is received. In that case, a response for the cancelled transaction SHOULD still be returned automatically, and a response for the cancelled transaction MUST be returned if a retransmission of the cancelled transaction is detected. (Error Code 407 SHOULD be used.)

[32] For instance when resources are reserved and committed externally as part of a transaction.

When a provisional response is received, the timeout period for the transaction in question MUST be set to a significantly higher value for this transaction ($Tt_{longtran}$). The purpose of this timer is primarily to detect endpoint failure. The default value of $Tt_{longtran}$ is 5 seconds, however the provisioning process may alter this.

When the transaction finishes execution, the final response is sent and the by now obsolete provisional response is deleted. In order to ensure rapid detection of a lost final response, final responses issued after provisional responses for a transaction MUST be acknowledged. The endpoint MUST therefore include an empty “ResponseAck” parameter in those, and only those, final responses. The presence of the “ResponseAck” parameter in the final response will trigger a “Response Acknowledgement” response to be sent back to the endpoint. Thus, the CMS MUST issue a "Response Acknowledgement" response whenever it receives a final response containing an empty "ResponseAck" parameter regardless of the receipt of a provisional response to the transaction since the provisional response may have been lost. The “Response Acknowledgement” response will include the transaction-id of the response it acknowledges in the response header. Receipt of this “Response Acknowledgement” response is subject to the same time-out and retransmission strategies and procedures as responses to commands (see Section 4.4), i.e., the sender of the final response will retransmit it if the “Response Acknowledgement” is not received in time. The “Response Acknowledgment ” response is never acknowledged.

6 SECURITY

If unauthorized entities could use the MGCP, they would be able to set up unauthorized calls or interfere with authorized calls. Security is not provided as an integral part of MGCP. Instead MGCP assumes the existence of a lower layer providing the actual security.

Security requirements and solutions for NCS are provided in the PacketCable Security Specification [21] which should be consulted for further information.

7 ACKNOWLEDGEMENTS

This specification was developed and influenced by numerous individuals representing many different vendors and organizations. PacketCable hereby wishes to thank everybody who participated directly or indirectly in this effort. In particular, PacketCable wants to recognize the authors of the original SGCP, MGCP, and IPDC specifications for the work they have done. Furthermore, PacketCable wishes to thank the following individuals for their involvement and contributions to this specification:

Flemming Andreasen (Cisco -Primary Author)
Brian Bailey (3Com)
Burcak Beser (Pacific Broadband)
David Bukovinsky (Blue Sky)
Rex Coldren (AG Communications)
Graeme Currie (Nortel)
Chad Griffiths (Broadcom)
Jack Fijolek (3Com)
Bill Foster (Cisco)
Billy Hare (Arris)
Barry Hoffner (Telcordia)
Christian Huitema (Telcordia)
Keith Kelly (Netspeak)
David Lide (Telogy)
Bill Marshall (AT&T)
Michael Mannette (3Com)
Jiri Matousek (Arris)
Sasha Medvinsky (Motorola)
Steve Michelson (AT&T)
Debbie Ng (Nortel)
David Oran (Cisco)
Jeff Orwick (NetSpeak)
Matt Osman (CableLabs)
Daniel Paul (Arris)
John Pickens (Com21)
David Reiter (Motorola)
Don Stanwyck (Lucent)
Bob Stein (Motorola)
Kurt Steinbrenner (3Com)
Michael Thomas (Cisco)
Raymond Yeung (Pacific Broadband)

Venkatesh Sunkad (CableLabs Team Lead)

8 REFERENCES

[1]. IETF RFC 3435, F. Andreasen, B. Foster, Media Gateway Control Protocol (MGCP) Version 1.0, January 2001.

[2]. IETF RFC 1889, RTP: A Transport Protocol for Real-Time Applications, January 1996.

[3]. IETF RFC 1890, RTP Profile for Audio and Video Conferences with Minimal Control, January 1996.

[4]. IETF RFC 2327, SDP: Session Description Protocol, April 1998.

[5]. IETF RFC 2974, SAP - Session Announcement Protocol, October 2000.

[6]. IETF RFC 2543, Session Initiation Protocol, March 1999.

[7]. IETF RFC 2326, Real-time Streaming Protocol (RTSP), April 1998.

[8]. ITU-T Recommendation Q.761, Functional Description of the IDSN User Part of Signalling System No. 7, (Malaga-Torremolinos, 1984; modified at Helsinki, 1993).

[9]. ITU-T Recommendation Q.762, General Function of messages and Signals of the IDSN User part of Signalling System No. 7, (Malaga-Torremolinos, 1984; modified at Helsinki, 1993).

[10]. ITU-T Recommendation H.323, Visual Telephone Systems and Equipment for Local Area Networks Which Provide a Non-guaranteed Quality-of-Service, November 2000.

[11]. ITU-T Recommendation H.225, Call Signaling Protocols and Media Stream Packetization for Packet Based Multimedia Communications Systems, November 2000.

[12]. ITU-T Recommendation H.245, Line Transmission of Non-telephone Signals, July 2001.

[13]. IETF RFC 1825, Security Architecture for the Internet Protocol, August 1995.

[14]. IETF RFC 1826, IP Authentication Header, August 1995.

[15]. IETF RFC 1827, Encapsulating Security Payload (ESP), August 1995.

[16]. Stevens, W. Richard, TCP/IP Illustrated, Volume 1, *The Protocols*, Addison-Wesley, 1994.

[17]. RTP Parameters, http://www.Iana.org/assignments/rtp-parameters/.

[18]. Bellcore, Bellcore Notes on the Networks, SR-2275.

[19]. PacketCable Audio/Video Codecs Specification, PKT-SP-CODEC-I05-040113, January 13, 2004, Cable Television Laboratories, Inc., http://www.PacketCable.com.

[20]. PacketCable Dynamic Quality of Service Specification, PKT-SP-DQOS-I08-040402, April 2, 2004, Cable Television Laboratories, Inc., http://www.PacketCable.com.

[21]. PacketCable Security Specification, PKT-SP-SEC-I10-040113, January 13, 2004, Cable Television Laboratories, Inc., http://www.PacketCable.com.

[22]. IETF RFC 2821, J. Klensin, Editor, Simple Mail Transfer Protocol, April 2001.

[23]. IETF RFC 1122, Requirements for Internet Hosts -- Communication Layers, October 1989.

[24]. IETF RFC 1034, Domain Names – Concepts and Facilities, November 1987.

[25]. IETF RFC 2045, Multipurpose Internet Mail Extensions Part 1, November 1996.

[26]. IETF RFC 2234, Augmented BNF for Syntax Specifications: ABNF, November 1997.

[27]. IETF RFC 2833, RTP Payload for DTMF Digits, Telephony Tones and Telephony Signals, May 2000.

[28]. GR-506, LSSGR: Signaling for Analog Interfaces, November 1996.

[29]. ITU-T Recommendation T.30, Procedures for Document Facsimile Transmission in the General Switched Telephone Network, April 1999.

[30]. ITU-T Recommendation V.8, Procedures for Starting Sessions of Data Transmission over the Public Switched Telephone Network, November, 2002.

[31]. ITU-T Recommendation V.21, 300 Bits Per Second Duplex Modem Standardized for Use in the General Switched Telephone Network, 1984.

[32]. ITU-T Recommendation V.25, Automatic Answering Equipment and General Procedures for Automatic Calling Equipment on the General Switched Telephone Network Including Procedures for Disabling of Echo Control Devices for both Manually And Automatically Established Calls, October 1996.

[33]. ITU-T Recommendation V.18, Operational and interworking requirements for DCEs operating in the text telephone mode, November 2000

[34]. PacketCable Provisioning Specification, PKT-SP-PROV-I09-040402, April 2, 2004, Cable Television Laboratories, Inc., http://www.PacketCable.com.

Appendix A. Event Packages

This section defines an initial set of event packages for the various types of endpoints currently defined by PacketCable for embedded clients.

Each package defines a package name for the package and event codes and definitions for each of the events in the package. In the tables of events/signals for each package, there are five columns:

Code The package unique event code used for the event/signal.

Description A short description of the event/signal.

Event A check mark appears in this column if the event can be Requested by the Media Gateway Controller. Alternatively, one or more of the following symbols may appear:

- **"P"** indicating that the event is persistent,
- "**S"** indicating that the event is an event-state that may be audited,
- **"C"** indicating that the event/signal may be detected/applied on a connection.

Signal If nothing appears in this column for an event, then the event cannot be signaled on command by the Media Gateway Controller. Otherwise, the following symbols identify the type of event:

- "**OO"** On/Off signal. The signal is turned on until commanded by the Media Gateway Controller 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. Default time-out values are supplied. A value of zero indicates that the time-out period is infinite. The provisioning process may alter these default values.
- **"BR"** Brief signal. The event has a short, known duration.

Additional info Provides additional information about the event/signal, e.g., the default duration of TO signals.

Unless otherwise stated, all of the events/signals are detected/applied on endpoints and audio generated by them is not forwarded on any connection the endpoint may have. Audio generated by events/signals that are detected/applied on a connection will however be forwarded on the associated connection irrespective of the connection mode.

A.1 Analog Access Lines

The following packages are currently defined for Analog Access Line endpoints:

- Line

A.2 Line Package

Package name: L

The codes listed in Table 19 below are used to identify events and signals for the "line" package for "analog access lines":

Table 19 - Line Package Codes for Events and Signals

Code	Description	Event	Signal	Additional Info
0-9,*,#,A, B,C,D	DTMF tones	√	BR	
bz	Busy tone	-	TO	Time-out = 30 seconds.
cf	Confirmation tone	-	BR	
ci(ti, nu, na)	Caller Id	-	BR	"ti" denotes time, "nu" denotes number, and "na" denotes name.
dl	Dial tone	-	TO	Time-out = 16 seconds.
ft	Fax tone	√	-	
hd	Off-hook transition	P, S	-	
hf	Flash hook	P	-	
hu	On-hook transition	P, S	-	
L	DTMF long duration	√	-	
ld	Long duration connection	C	-	
ma	Media start	C	-	
mt	Modem tones	√	-	
mwi	Message waiting indicator	-	TO	Time-out = 16 seconds.
oc	Operation complete	√	-	
of	Operation failure	√	-	
osi	Open interval	-	TO	Default=900ms.
ot	Off-hook warning tone	-	TO	Time-out = infinite.
r0, r1, r2, r3, r4, r5, r6 or r7	Distinctive ringing (0..7)	-	TO	Time-out = 180 seconds.
rg	Ringing	-	TO	Time-out = 180 seconds.
ro	Reorder tone	-	TO	Time-out = 30 seconds.
rs	Ringsplash	-	BR	
rt	Ring back tone	-	C, TO	Time-out = 180 seconds.
sl	Stutter dial tone	-	TO	Time-out = 16 seconds.
t	Timer	√	-	
TDD	Telecomm Devices for the Deaf (TDD) tones	√	-	
vmwi	Visual message waiting indicator	-	OO	
wt1, wt2, wt3, wt4	Call waiting tones	-	TO	Time-out = [(MaxReps + 1) + (MaxReps * Delay)] seconds, where the default values for MaxReps = 1 and Delay = 10 as defined in the NCS Signalling MIB specification.
X	DTMF tones wildcard	√	-	Matches any of the digits "0-9".

The definition of the individual events and signals are as follows:

DTMF tones (0-9,*,#,A, B,C,D): Detection and generation of DTMF signals is described in GR-506-CORE – LSSGR: SIGNALING [28], Section 15. It is considered an error to try and play DTMF tones on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Busy tone (bz): Station Busy 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 0.5 seconds on followed by 0.5 seconds off, repeating. See , Section 17.2.6. It is considered an error to try and play busy tone on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Confirmation tone (cf): Confirmation Tone uses the same frequencies and levels as dial tone (350 and 440 Hertz) but with a cadence of 0.1 second on, 0.1 second off repeated three times. See [27], Section 17.2.4. It is considered an error to try and play confirmation tone on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Caller Id (ci(time, number, name)): See TR-NWT-001188, [27], and TR-NWT-000031. Each of the three fields are optional, however each of the commas will always be included.

- The **time** parameter is coded as "MM/DD/HH/MM", where the first MM is a two-digit value for Month between 01 and 12, DD is a two-digit value for Day between 1 and 31, and Hour and Minute are two-digit values coded according to military local time, e.g., 00 is midnight, 01 is 1 a.m., and 13 is 1 p.m.
- The **number** parameter is coded as an ASCII character string of decimal digits that identify the calling line number. White spaces are permitted if the string is quoted, however they will be ignored.
- The **name** parameter is coded as a string of ASCII characters that identify the calling line name. White spaces, commas, and parentheses are permitted if the string is quoted.

A "P" in the number or name field is used to indicate a private number or name, and an "O" is used to indicate an unavailable number or name. The following example illustrates the use of the caller-id signal:

```
S: ci(10/14/17/26, "555 1212", CableLabs)
```

In addition to the generic signaling requirements described in Section 5.2.2.10, the MTA MUST support, at a minimum, the combination of one signal on an endpoint along with CallerId (ci) within the same SignalRequest line (e.g., S: rg, ci(time,number,name)) and a signal on each connection associated with the endpoint.

Dial-tone (dl): Dial Tone is a combination of two continuous AC tones with frequencies of 350 and 440 Hertz and levels of –13dBm each to give a combined level of –10 dBm. See [27], Section 17.2.1. It is considered an error to try and play dial-tone on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Fax tone (ft): The fax tone event is generated whenever a fax call is detected by presence of V.21 fax preamble. The fax tone event SHOULD also be generated when the T.30 CNG tone is detected. See ITU-T Recommendations T.30 [29] and V.21 [31].

Off-hook transition (hd): See [27], Section 12.

Flash hook (hf): See [27], Section 12.

On-hook transition (hu): See [27]. The timing for the on-hook signal is for flash response enabled.

DTMF Long duration (L): The "DTMF Long duration" is observed when a DTMF signal is produced for a duration longer 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.

Long duration connection (ld): The "long duration connection" is detected when a connection has been established for more than a certain period of time. The default value is 1 hour, however this may be changed by the provisioning process.

The event is detected on a connection. When no connection is specified, the event applies to all connections for the endpoint, regardless of when the connections are created.

Media start (ma): The media start event occurs on a connection when the first valid[33] RTP media packet is received on the connection. This event can be used to synchronize a local signal, e.g., ringback, with the arrival of media from the other party.

The event is detected on a connection. When no connection is specified, the event applies to all connections for the endpoint, regardless of when the connections are created.

Modem tones (mt): Modem tone (mt): The modem tone event is generated whenever a data call is detected by presence of V.25 answer tone (ANS) with or without phase reversal or V.8 modified answer tone (ANSam) with or without phase reversal. See ITU-T Recommendation V.25 [32] and V.8 [30].

Message Waiting Indicator (mwi): Message Waiting indicator tone uses the same frequencies and levels as dial tone (350 and 440 Hertz at –13dBm each) but with a cadence of 0.1 second on, 0.1 second off repeated 10 times followed by steady application of dial tone. See [27], Section 17.2.3. It is considered an error to try and play message waiting indicator on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Open Switch Interval (osi): See [27]: Voiceband Data Transmission Interface, Section 2.2.2. See also [27], Section 4.5.2.1.

Operation complete (oc): The operation complete event is generated when the gateway was asked to apply one or several signals of type TO on the endpoint, and one or more of those signals 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 a parameter the name of the signal that came to the end of its live time, as in:

```
O: L/oc(L/dl)
```

When the reported signal was applied on a connection, the parameter supplied will include the name of the connection as well, as in:

```
O: L/oc(L/rt@0A3F58)
```

When the operation complete event is requested, it cannot be parameterized with any event parameters. When the package name is omitted, the default package name is assumed.

The operation complete event may additionally be generated as defined in the base protocol, e.g., when an embedded ModifyConnection command completes successfully, as in:

```
O: L/oc(B/C)
```

Note the use of “B” above as the prefix for the parameter reported.

Operation failure (of): In general, the operation failure event may be generated when the endpoint was asked to apply one or several signals of type TO on the endpoint, and one or more of those signals failed prior to timing out. The completion report may carry as a parameter the name of the signal that failed, as in:

```
O: L/of(L/rg)
```

When the reported signal was applied on a connection, the parameter supplied will include the name of the connection as well, as in:

```
O: L/of(L/rt@0A3F58)
```

When the operation failure event is requested, event parameters can not be specified. When the package name is omitted, the default package name is assumed.

[33] When authentication and integrity security services are used, an RTP packet is not considered valid until it has passed the security checks.

The operation failure event may additionally be generated as specified in the base protocol, e.g., when an embedded ModifyConnection command fails, as in:

```
O: L/of(B/C(M(sendrecv(AB2354))))
```

Note the use of "B" above as the prefix for the parameter reported.

Off-hook warning tone (ot): Receiver Off Hook Tone (ROH Tone) is the irritating noise a telephone makes when it is not hung up correctly. ROH Tone is generated by combining four tones at frequencies of 1400 Hertz, 2060 Hertz, 2450 Hertz and 2600 Hertz at a cadence of 0.1 second on, 0.1 second off, repeating. GR-506-CORE, Section 17.2.8 contains details about required power levels. It is considered an error to try and play off-hook warning tone on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Distinctive ringing (r0, r1, r2, r3, r4, r5, r6 or r7): See [27], Section 14 The values for r1 to r5 are as defined for distinctive ringing pattern 1 to 5 as defined in [27]. The provisioning process MAY define the ringing cadence for each of the r0-r7 signals. The MTA MUST support provisioning for r0, r6 and r7. It is considered an error to try and ring a phone that is off hook and an error should consequently be returned when such attempts are made (error code 401 – phone off hook).

Ringing (rg): See [27], Section 14. The value for rg is as defined for distinctive ringing pattern 1 as defined in [27]. The provisioning process MAY define the ringing cadence. The ringing signal may be parameterized with the signal parameter "rep" which specifies the maximum number of ringing cycles (repetitions) to apply. The following will apply the ringing signal for up to 6 ringing cycles:

```
S: rg(rep=6)
```

It is considered an error to try and ring a phone that is off hook and an error should consequently be returned when such attempts are made (error code 401 – phone off hook).

Reorder tone (ro): 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 reorder tone is 0.25 seconds on followed by 0.25 seconds off, repeating continuously. See [27], Section 17.2.7. It is considered an error to try and play reorder tone on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Ringsplash (rs): 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 Call Forwarding 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. The provisioning process MAY define the cadence for ringsplash. It is considered an error to try and ring a phone that is off hook and an error should consequently be returned when such attempts are made (error code 401 – phone off hook).

Ring back tone (rt): Audible Ring Tone is 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. In the U.S. the cadence for Audible Ring Tone is defined to be 2 seconds on followed by 4 seconds off. The definition of the tone is defined by the national characteristics of the Ringback Tone, and MAY be established via provisioning. See [27], Section 17.2.5.

The ringback signal can be applied to both an endpoint and a connection.

When the ringback signal is applied to an endpoint, it is considered an error to try and play ring back tones, if the endpoint is considered on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook). When the ringback signal is applied to a connection, no such check is to be made.

Stutter Dial tone (sl): Stutter Dial Tone (also called Recall Dial Tone) is generated by supplying Confirmation Tone, followed by continuous Dial Tone. See [27], Section 17.2.2. The stutter dial tone signal may be parameterized with the signal parameter "del" which will specify a delay in milliseconds to apply

between the confirmation tone and the dial tone[34]. The following will apply stutter dial tone with a delay of 1.5 seconds between the confirmation tone and the dial tone:

```
S: sl(del=1500)
```

It is considered an error to try and play stutter dial tone on a phone that is on hook and an error should consequently be returned when such attempts are made (error code 402 – phone on hook).

Timer (t): As described in Section 4.1.5, timer T is a provisionable timer that can only be cancelled by DTMF input. When timer T is used with the "accumulate according to digit map" action, 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 and takes on one of two values, T_{par} or T_{crit}. 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_{par}, corresponding to partial dial timing. If a timer is all that is required to produce a match, timer T takes on the value T_{crit} corresponding to critical timing. An example use is:

```
S: dl
R: [0-9T](D)
```

When timer T is used without the "accumulate according to digit map" action, timer T takes on the value T_{crit}, and 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 inter-digit timer when overlap sending is used, e.g.:

```
R: [0-9](N), T(N)
```

Note, that only one of the two forms can be used at a time, since a given event can only be specified once.

The default value for T_{par} is 16 seconds and the default value for T_{crit} is 4 seconds. The provisioning process may alter both of these.

Telecomm Devices for the Deaf tones (TDD): The TDD event is generated whenever a TDD call is detected – see e.g., ITU-T recommendation V.18 [33].

Visual Message Waiting Indicator (vmwi): The transmission of the VMWI messages will conform to the requirements in TR-H-000030 – Section 2.3.2, "On-hook Data Transmission Not Associated with Ringing", and the CPE guidelines in SR-TSV-002476. VMWI messages will only be sent from the embedded client to the attached equipment 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. When the endpoint restarts, it should not attempt to turn off a visual message waiting indicator, even if it believes it should be off. The Call Agent should periodically refresh the CPE's visual indicator. See TR-NWT-001401 – Visual Message Waiting Indicator Generic Requirements; and [27] Voiceband Data Transmission Interface.

Call Waiting tone1 (wt1, .., wt4): Call Waiting tones are defined in [27], Section 14.2 – the number refers to the tone pattern used. The duration of the tone pattern lasts for up to 1 second. GR-571-CORE (FSD 01-02-1201) indicates that two tone patterns should be played separated by a period of 10 seconds. The default maximum number of repetitions and delay between repetitions may be altered by the provisioning process. The default Call Waiting tone is a 440-Hz tone applied for 300 ± 50 ms. The talking path should be interrupted for a maximum of 400 ms for the application of each CW tone pattern. When this signal is requested, the embedded client will play the two tone patterns as specified in [27] before the "TO" signal times out. It is considered an error to try and apply call waiting tones on a phone that is on hook and an error SHOULD consequently be returned when such attempts are made (error code 402 – phone on hook).

The default time-out is calculated based on the default (or provisioned) number of repetitions, default (or provisioned) delay between repetitions and length of time to generate the call waiting tone itself (1 second). For example, if Call Waiting Delay is set to 12 seconds and Call Waiting Maximum Number of Repetitions is set to 2, then the call waiting tone will be heard up to three times (played once and repeated twice). Therefore, the default time-out is calculated by adding the length of time it takes to play the call waiting tone three times (3 seconds) and the delay between repetitions (2 * 12 seconds). The calculated default timeout in

[34] This feature is needed for, e.g., Speed Dialing.

this example is therefore 27 seconds. Upon request the call waiting signal may be parameterized with the signal parameter “to” which in essence overrides the calculated default timeout described above.

DTMF tones wildcard (X): The DTMF tones wildcard matches any DTMF digit between 0 and 9.

A.3 Video

Event packages for video will be provided in a future version of this document.

Appendix B. Connection Mode

An MGCP connection can establish one or more media streams. These streams are either incoming or outgoing. The "connection mode" parameter controls the flow of media on the media stream. When there is only one connection to an endpoint, the mapping of these streams is straightforward. However, when several connections are established to an endpoint, there can be many incoming and outgoing streams. Depending on the connection mode used, these streams may mix differently with each other and the streams going to/from the handset. Table 20 below describes how media from different connections should be mixed when one or more connections exist. The table assumes that there are no signals being applied on a connection. Table 20 uses the following conventions:

- A_{in} is the incoming media stream from Connection A
- B_{in} is the incoming media stream from Connection B
- H_{in} is the incoming media stream from the Handset Microphone
- A_{out} is the outgoing media stream to Connection A
- B_{out} is the outgoing media stream to Connection B
- H_{out} is the outgoing media stream to the Handset earpiece
- NA indicates no stream whatsoever

Table 20 - Control of Media Streams by Connection Mode

		Connection A Mode						
		sendonly	recvonly	sendrecv	confrnce	inactive	netwloop/ netwtest	replcate
Connection B Mode	sendonly	$A_{out}= H_{in}$ $B_{out}= H_{in}$ $H_{out}=NA$	$A_{out}=NA$ $B_{out}=H_{in}$ $H_{out}= A_{in}$	$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=A_{in}$	$A_{out}=H_{in}$ $B_{out}=H_{in}$ $H_{out}=A_{in}$	$A_{out}= NA$ $B_{out}= H_{in}$ $H_{out}=NA$	$A_{out}=A_{in}$ $B_{out}= H_{in}$ $H_{out}=NA$	$A_{out}= H_{in}$ $B_{out}= H_{in}$ $H_{out}=NA$
	recvonly		$A_{out}= NA$ $B_{out}= NA$ $H_{out}=A_{in}+B_{in}$	$A_{out}= H_{in}$ $B_{out}= NA$ $H_{out}=A_{in}+B_{in}$	$A_{out}=H_{in}$ $B_{out}=NA$ $H_{out}=A_{in}+B_{in}$	$A_{out}=NA$ $B_{out}=NA$ $H_{out}= B_{in}$	$A_{out}= A_{in}$ $B_{out}= NA$ $H_{out}= B_{in}$	$A_{out}= H_{in}+B_{in}$ $B_{out}= NA$ $H_{out}= B_{in}$
	sendrecv			$A_{out}= H_{in}$ $B_{out}= H_{in}$ $H_{out}=A_{in}+B_{in}$	$A_{out}= H_{in}$ $B_{out}=H_{in}$ $H_{out}=A_{in}+B_{in}$	$A_{out}= NA$ $B_{out}= H_{in}$ $H_{out}= B_{in}$	$A_{out}= A_{in}$ $B_{out}= H_{in}$ $H_{out}= B_{in}$	$A_{out}= H_{in}+B_{in}$ $B_{out}= H_{in}$ $H_{out}= B_{in}$
	confrnce				$A_{out}=H_{in}+B_{in}$ $B_{out}=H_{in}+A_{in}$ $H_{out}=A_{in}+B_{in}$	$A_{out}= NA$ $B_{out}= H_{in}$ $H_{out}= B_{in}$	$A_{out}= A_{in}$ $B_{out}= H_{in}$ $H_{out}= B_{in}$	$A_{out}= H_{in}+B_{in}$ $B_{out}= H_{in}$ $H_{out}= B_{in}$
	inactive					$A_{out}= NA$ $B_{out}= NA$ $H_{out}= NA$	$A_{out}= A_{in}$ $B_{out}= NA$ $H_{out}=NA$	$A_{out}= H_{in}$ $B_{out}= NA$ $H_{out}=NA$
	netwloop/ netwtest						$A_{out}= A_{in}$ $B_{out}= B_{in}$ $H_{out}=NA$	$A_{out}= H_{in}$ $B_{out}= B_{in}$ $H_{out}=NA$
	replcate							$A_{out}= H_{in}$ $B_{out}= H_{in}$ $H_{out}=NA$

For clarity, Table 20 above is repeated below in graphical form:

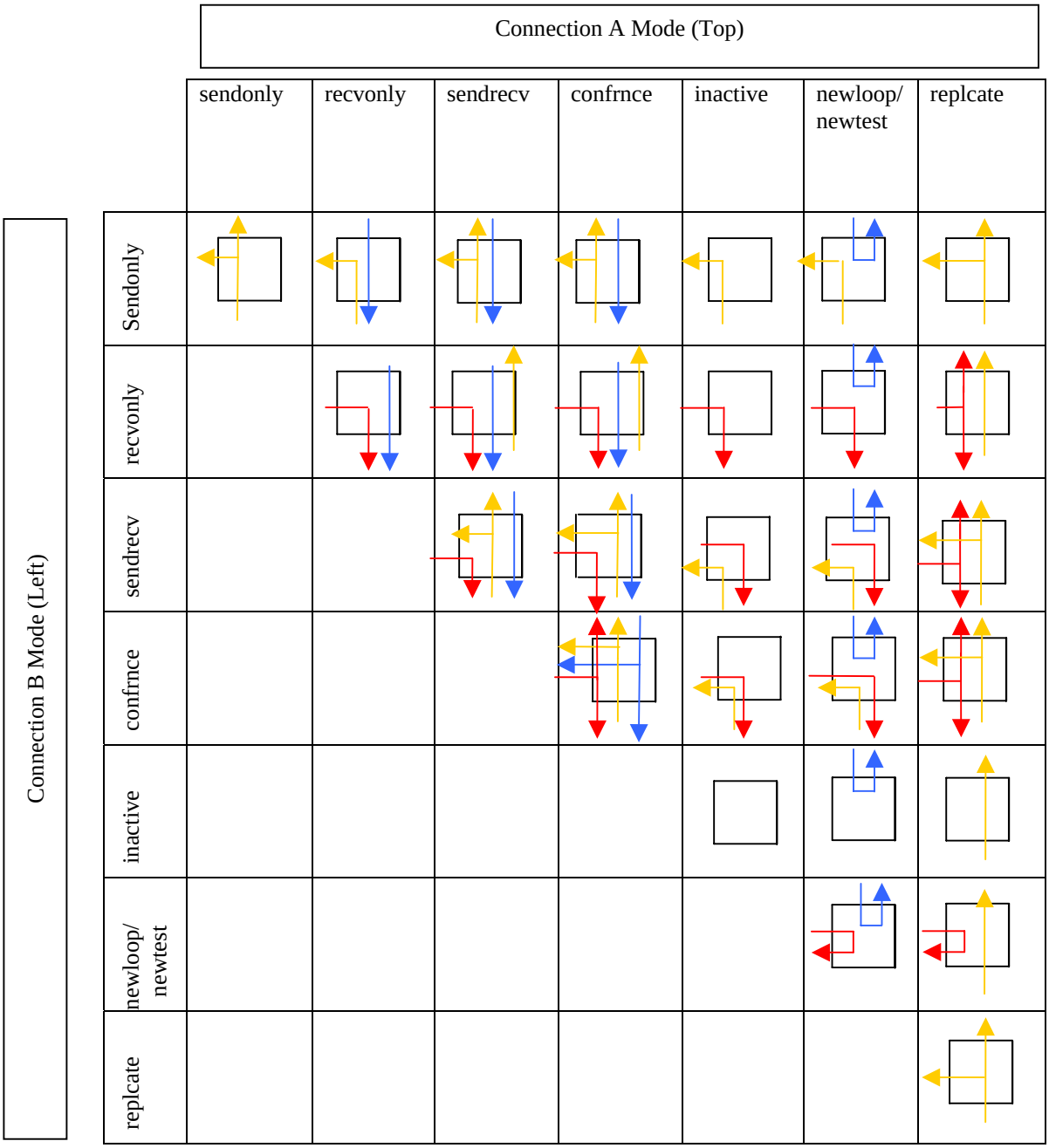

Figure 6 - Control of Media Streams by Connection Mode - Graphical Representation

If there are three or more connections, their media will still be mixed as defined in the Table 20 above. If internal resources are not available such that the media can not be mixed, the gateway should return error code 502 (insufficient resources).

These connections can be graphically represented as such:

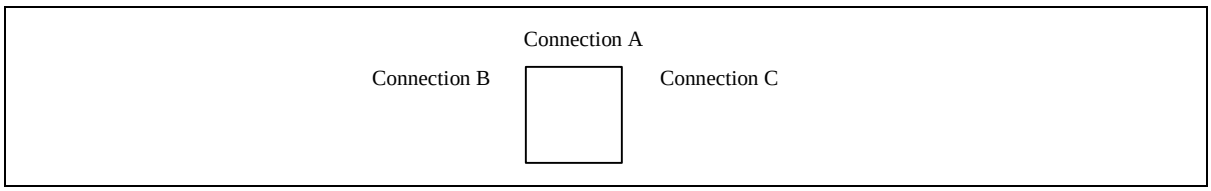

Figure 7 - Graphical Representation of Connections

For example, if Connection A is sendrecv, Connection B is confrnce, and Connection C is recvonly, from Table 20 above, the outputs in each mode will be illustrated in Figure 8 below:

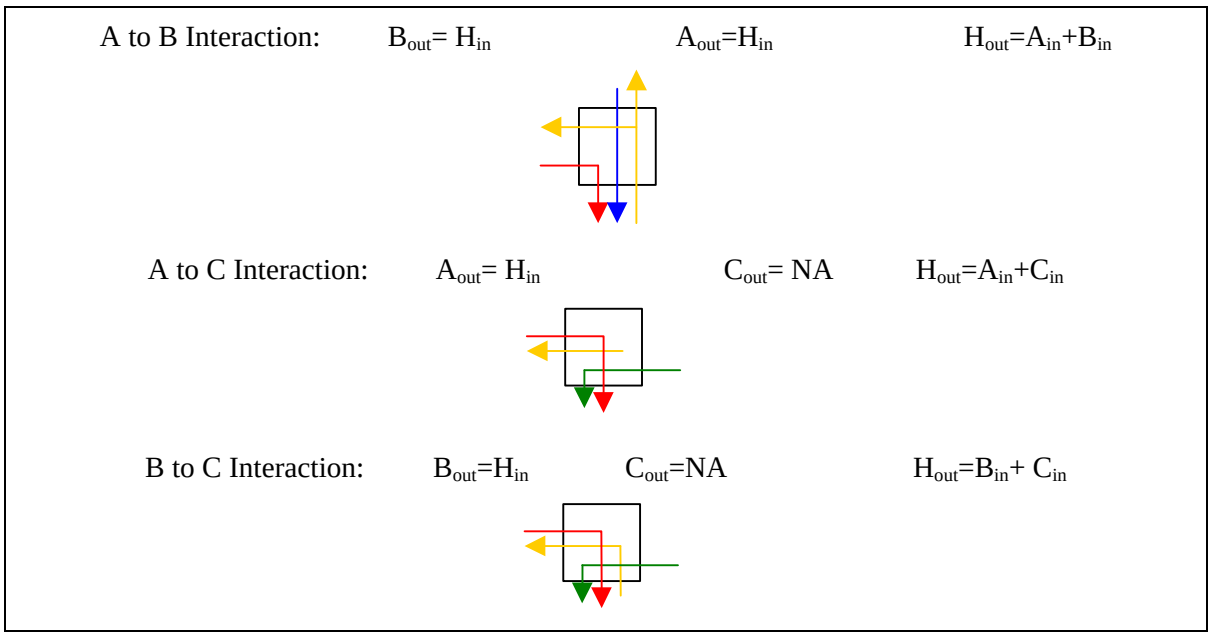

Figure 8 - Individual Media Streams on Connections

A to B to C interaction:

$A_{out}=H_{in}$

$B_{out}=H_{in}$

$C_{out}=NA$

$H_{out}= B_{in+} A_{in+} C_{in}$

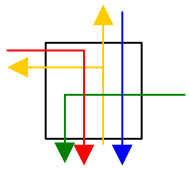

Figure 9 - A to B to C Interaction

Appendix C. Dynamic Quality-of-Service

In this appendix, we provide additional detail on the usage of Dynamic Quality-of-Service (D-QoS) in NCS. We describe the expected MTA behavior in more detail and include a state machine and pseudo-code that the MTA MUST follow to support the D-QoS behavior described. The PacketCable Dynamic Quality-of-Service Specification [20] should be consulted for further details.

C.1 Introduction

MTAs implementing support for Dynamic Quality of Service need to store and maintain D-QoS state on a per connection basis. Whenever D-QoS has been used for a connection, the endpoint will keep the following D-QoS information associated with the connection until it is deleted:

- **GateID** The current GateID used for the connection.
- **ResourceID** The current ResourceID used for the connection.
- **Last reservation** The parameters for the most recent reservation for the connection. This includes classifiers as well as media parameters in both the send and receive direction.
- **Last commit** The parameters for the most recent commit for the connection. This includes classifiers as well as media parameters in both the send and receive direction.
- **Reserve Destination** An IP address and port that may be used to enable resource reservations where the remote address info is not yet known as explained below.
- **Gate Location** The IP address and port where the D-QoS commit message should be sent to when using RSVP. The MTA learns this address through the RSVP QoS messages.

The GateID is the key to resource reservation. Once a GateID has been provided for a connection, a D-QoS state machine is created for the connection, and all of the above information will be maintained for the connection until it is deleted.

Resources can be reserved and committed independently in both the send and receive direction by the MTA. The send destination IP address and port as well as the source IP address are taken from the RemoteConnectionDescriptor, when a RemoteConnectionDescriptor has been provided. In that case, the MTA MUST use the following classifiers identified in Table 21 below for the resource reservation and commit:

Table 21 - Classifiers for Resource Reservation and Commit: Remote Connection Descriptor Provided

		MTA-o (DOCSIS/RSVP)
Downstream/receive		
	Source IP	IP(SDP-t)
	Source Port	*
	Destination IP	IP(SDP-o)
	Destination Port	Port(SDP-o)
Upstream/send		
	Source IP	IP(SDP-o)
	Source Port	Port(o)
	Destination IP	IP(SDP-t)
	Destination Port	Port(SDP-t)

where

- **IP(SDP-o)** refers to the media IP address in MTA-o's LocalConnectionDescriptor.
- **IP(SDP-t)** refers to the media IP address in MTA-o's RemoteConnectionDescriptor.
- **Port(SDP-o)** refers to the media port in MTA-o's LocalConnectionDescriptor.
- **Port(SDP-t)** refers to the media port in MTA-t's LocalConnectionDescriptor.
- **Port(o)** refers to the source port MTA-o will be using when sending media on this connection. Note, that this may or may not be the same as Port(SDP-o).

When a RemoteConnectionDescriptor has not yet been provided, the actual send destination IP address and port is unknown and the ReserveDestination address is therefore used instead. For the receive direction, the source IP address and port is wild-carded. This enables a reservation and a receive commit of the resource on the access link. The classifiers identified in Table 22 MUST be used:

Table 22 - Classifiers for Resource Reservation and Commit: Remote Connection Descriptor Not Provided

		MTA-o (DOCSIS/RSVP)
Downstream/receive		
	Source IP	*
	Source Port	*
	Destination IP	IP(SDP-o)
	Destination Port	Port(SDP-o)
Upstream/send		
	Source IP	IP(SDP-o)
	Source Port	Port(o)
	Destination IP	IP(RD-o)
	Destination Port	Port(RD-o)

where

- **IP(RD-o)** refers to the IP address in the ReserveDestination supplied.
- **IP(Port-o)** refers to the port number in the ReserveDestination supplied. If no port number is specified a default value of 9 applies.

Once the actual send destination and receive source media addresses and port are known, the reservations are updated with the appropriate classifiers.

When RSVP is used as the resource reservation protocol, the destination address used for the RSVP PATH message will be the ReserveDestination IP address supplied until a RemoteConnectionDescriptor is supplied.

C.2 NCS/D-QoS State Machine

As explained above, the MTA maintains state for the Dynamic Quality of Service used on a connection. The state is derived from a state machine which is driven by the following:

- **Current state** which consists of the pair (SendQoSState, ReceiveQoSState), where each QoS state may be one of the following:
  - **N** No resource reservation exists for the direction.
  - **R** A resource reservation exists for the direction, but no resources are currently committed.
  - **C** A resource reservation exists for the direction, some resources are currently committed.

- **Connection mode** which is the NCS connection mode. The connection modes "Conference", "Network Loopback", and "Network Continuity Test" are not shown explicitly in the state machine, as they are all similar to "SendReceive". The connection mode "Replicate" is also not shown as it is similar to "SendOnly".
- **Resource Change** which is one or more of the following:
  - RemoteConnectionDescriptor IP address or port changes (classifier needs to be updated). This includes the case where it arrives for first time.
  - Codec changes
  - Ptime changes
  - etc.
- The **D-QoS** rules provided in Section 4.3.3.

As a result of a CreateConnection or a ModifyConnection command, if the DOCSIS QoS parameters, Classifier or Authorization Block changed from the previous state, then a new DSA/DSC message MUST be sent by the E-MTA to the CMTS containing the updated parameters. An E-MTA MAY send a DSC message to the CMTS if the DOCSIS QoS parameters, Classifier and Authorization Block have not changed as a result of a CreateConnection or a ModifyConnection command. If a ResourceID is supplied as well and it is the same as the old ResourceID, the reservation(s) for the new state machine MUST be performed before the reservation(s) for the old state machine are released.

The set of possible states are:

- (N, N) Send resources not reserved, receive resources not reserved.
- (R, R) Send resources reserved, receive resources reserved.
- (C, R) Send resources reserved and committed, receive resources reserved.
- (R, C) Send resources reserved, receive resources reserved and committed.
- (C, C) Send resources reserved and committed, receive resources reserved and committed.
- (R, N) Send resources reserved, receive resources not reserved.
- (C, N) Send resources reserved and committed, receive resources not reserved.
- (N, R) Send resources not reserved, receive resources reserved.
- (N, C) Send resources not reserved, receive resources reserved and committed.

Once resources have been reserved and/or committed for a direction, a reservation for that direction will exist for the lifetime of the connection. The relationship between states and connection mode or D-QoS reservation parameters is shown in Table 23 below:

Table 23 - States Related to Connection Modes and DQoS Reservation Parameters

	SendState	RecvState
No Reserve/Commit parameter supplied – connection mode:		
inactive	R	R
sendonly, replcate	C	R
recvonly	R	C
sendrecv, confrnce, netwloop, netwtest	C	C
Reserve/Commit parameter supplied:		
sendresv	R	N, R*
recvresv	N, R*	R
snrcresv	R	R
sendcomt	C	N, R*
recvcomt	N, R*	C
snrccomt	C	C

* If resources have been reserved or committed previously for the direction, the state will be R, otherwise the state will be N.

The actual state transition diagram is depicted in Figure 10 and Figure 11 below:

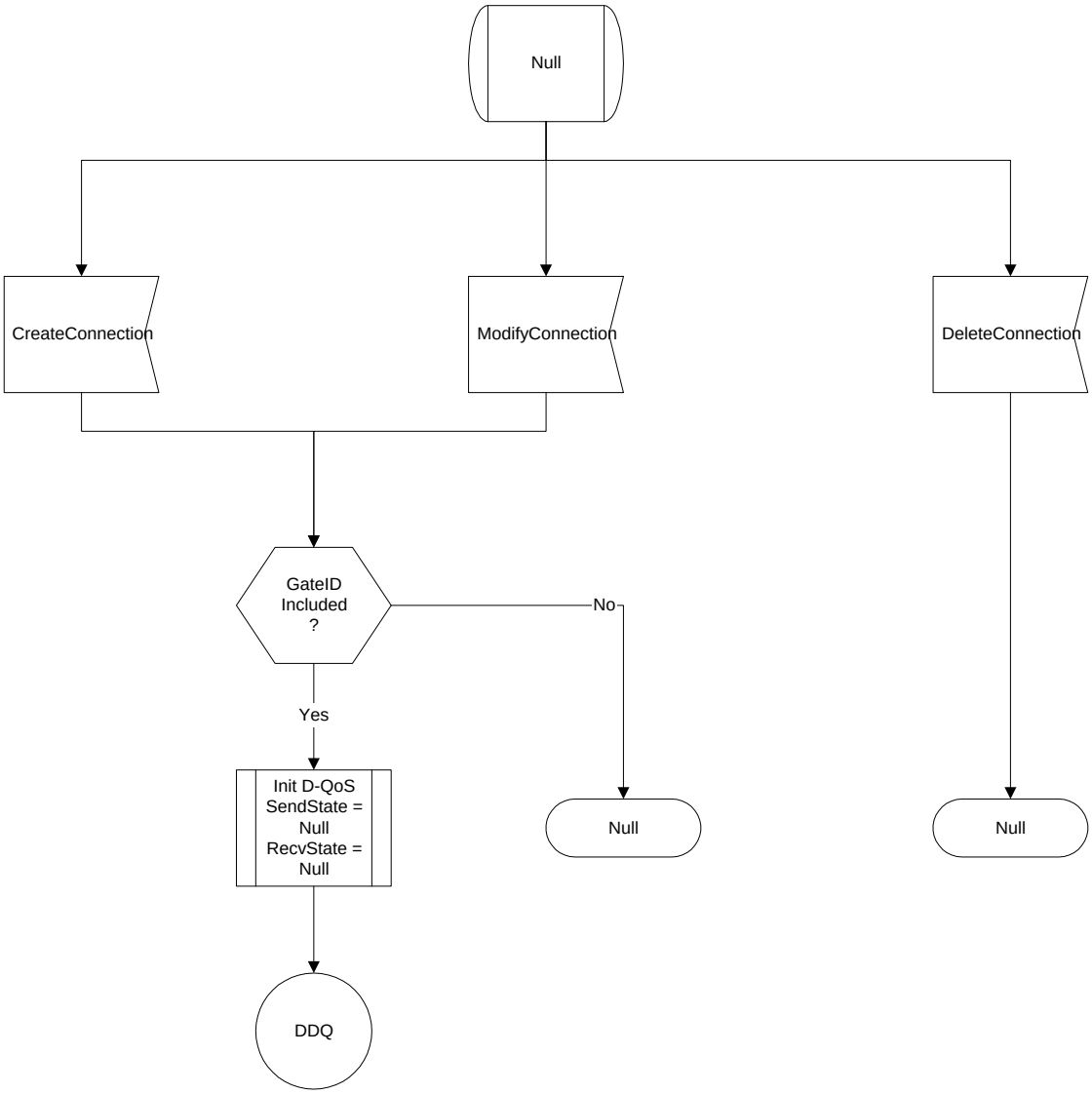

Figure 10 - NCS/D-QoS State Diagram (1:2)

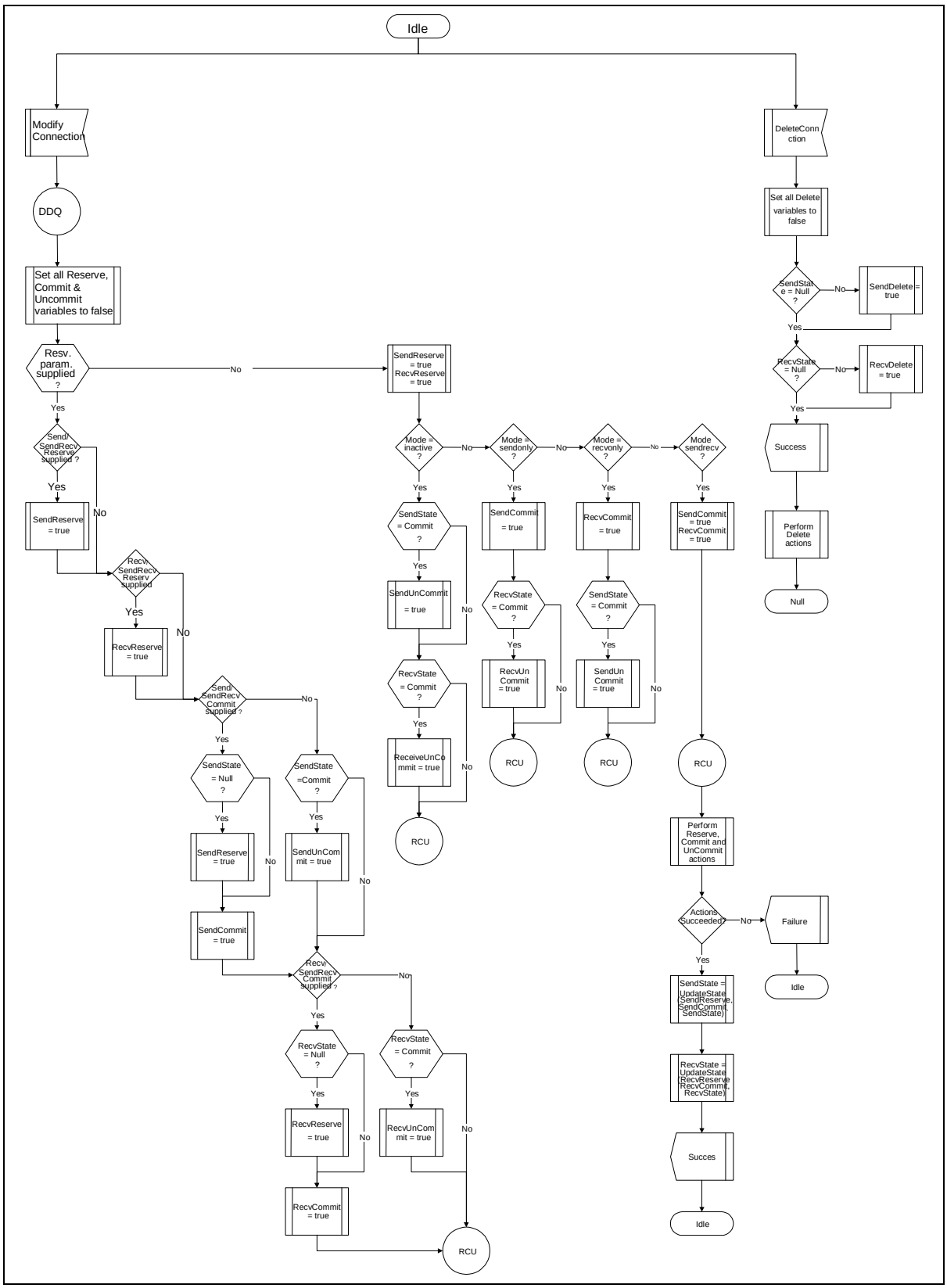

Figure 11 - NCS/D-QoS State Diagram (2:2)

When executing the state machine, boolean variables will be set to indicate whether reserve, unreserve, commit, and uncommit operations are to be performed. The pseudo-code below then provides details on individual D-QoS procedures that are to be executed as indicated by these booleans. The following *actions* specify the D-QoS actions to be taken in each of these procedures:

- SR means a D-QoS Send Reservation will be performed,
- RR means a D-QoS Receive Reservation will be performed,
- SC means a D-QoS Send Commit will be performed,
- RC means a D-QoS Receive Commit will be performed,
- SD means a D-QoS Send Reservation Delete will be performed,
- RD means a D-QoS Receive Reservation Delete will be performed,
- SU means a D-QoS Send Uncommit, i.e., lower committed send resources to zero, will be performed,
- RU means a D-QoS Receive Uncommit, i.e., lower committed send resources to zero, will be performed.

SendReserve()

```
If <current resources reserved ≠ resources to reserve> then { -- skip reservation if existing reservation OK
        If <RemoteConnectionDescriptor provided> then
            SR(RemoteConnectionDescriptor)   -- Use RemoteConnectionDescriptor classifier
        else if <ReserveDestination provided> then
            SR(ReserveDestination)           -- Use ReserveDestination classifier, send to
                                             -- ReserveDestination if RSVP
        else ERROR
}
```

ReceiveReserve()

```
If <current resources reserved ≠ resources to reserve> then {      -- skip reservation if existing reservation OK
        If <RemoteConnectionDescriptor provided> then
            RR(RemoteConnectionDescriptor)   -- Use RemoteConnectionDescriptor classifier
        else if <(DOCSIS QoS) or (RSVP and ReserveDestination provided )> then
            RR(*)                            -- Use wildcard classifier, send to ReserveDestination
                                             -- if RSVP
        else ERROR
}
```

SendCommit()

```
If <current resources committed ≠ resources to commit> then {      -- skip commit if existing OK
        If <RemoteConnectionDescriptor provided> then {
            If not <resources to commit ⊂ resources reserved > then {      -- old reservation does not
                    SR(RemoteConnectionDescriptor)           -- satisfy what is about to be
            }                                                -- committed, so update reservation
            if <(DOCSIS QoS) or (RSVP and ReserveDestination provided )> then {
                    SC(RemoteConnectionDescriptor)           -- send to ReserveDestination if
                                                             -- RSVP
            } else
                    ERROR
        } else ERROR.          -- Cannot commit send direction without RemoteConnectionDescriptor
```

```
}
ReceiveCommit()
If <current resources committed ≠ resources to commit> then {       -- skip commit if existing OK
        If not <resources to commit ⊂ resources reserved> then {
            If <RemoteConnectionDescriptor provided> then
                    RR(RemoteConnectionDescriptor)
            else if <(DOCSIS QoS) or (RSVP and ReserveDestination provided )> then
                    RR(*)               -- Use wildcard classifier, send to ReserveDestination if RSVP
            else
                    ERROR
         }
         If <RemoteConnectionDescriptor provided> then
            RC(RemoteConnectionDescriptor)
         else if <(DOCSIS QoS) or (RSVP and ReserveDestination provided )> then
            RC(*)               -- Use wildcard classifier, send to ReserveDestination if RSVP
         else
            ERROR
}
SendReserveDelete()
If <send resources reserved> then
        SD()                    -- delete the reservation
ReceiveReserveDelete()
If <receive resources reserved> then
        RD()                    -- delete the reservation
SendUnCommit()
If <send resources committed> then
        SU()                    -- uncommit committed resources
ReceiveUnCommit()
If <receive resources committed> then
        RU()                    -- uncommit committed resources
State UpdateState(DoCommit, DoReserve, OldState)
If <DoCommit = true> then
        return Commit
else if <DoReserve = true> then
        return Reserve
else
        return OldState
```

Appendix D. Example Command Encodings

This appendix provides examples of commands and responses shown with the actual encoding used. Examples are provided for each command. All commentary shown in the commands and responses is optional.

D.1 NotificationRequest

The first example illustrates a NotificationRequest that will ring a phone and look for an off-hook event:

```
RQNT 1201 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
R: hd(N)
S: rg
```

The response indicates that the transaction was successful:

```
200 1201 OK
```

The second example illustrates a NotificationRequest that will look for and accumulate an off-hook event, and then provide dial-tone and accumulate digits according to the digit map provided. The "notified entity" is set to "ca@ca1.whatever.net:5678", and since the SignalRequests parameter is empty[35], all currently active TO signals will be stopped. All events in the quarantine buffer will be processed, and the list of events to detect in the "notification" and "lockstep" state will include fax tones in addition to the "requested events" and persistent events:

```
RQNT 1202 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
R: hd(A, E(S(dl), R(oc, hu, [0-9#*T](D))))
D: (0T|00T|#xxxxxx|*xx|91xxxxxxxxx|9011x.T)
S:
Q: process
T: ft
```

The response indicates that the transaction was successful:

```
200 1202 OK
```

D.2 Notify

The example below illustrates a Notify message that notifies an off-hook event followed by a 12-digit number beginning with "91". A request identifier correlating the Notify with the NotificationRequest it results from is included. The command is sent to the current "notified entity", which typically will be the actual value supplied in the NotifiedEntity parameter, i.e., "ca@ca1.whatever.net:5678" – a failover situation could have changed this:

```
NTFY 2002 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
O: hd,9,1,2,0,1,8,2,9,4,2,6,6
```

The Notify response indicates that the transaction was successful:

```
200 2002 OK
```

[35] It could have been omitted as well.

D.3 CreateConnection

The first example illustrates a CreateConnection command to create a connection on the endpoint specified. The connection will be part of the specified CallId. The LocalConnectionOptions specify that G.711 u-law will be the codec used and the packetization period will be 10 ms. The connection mode will be "receive only":

```
CRCX 1204 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
L: p:10, a:PCMU
M: recvonly
```

The response indicates that the transaction was successful, and a connection identifier for the newly created connection is therefore included. A session description for the new connection is included as well – note that it is preceded by an empty line.

```
200 1204 OK
I: FDE234C8

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
a=mptime:10
```

The second example illustrates a CreateConnection command containing a notification request and a RemoteConnectionDescriptor:

```
CRCX 1205 aaln/1@rgw-2569.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
L: p:10, a:PCMU
M: sendrecv
X: 0123456789AD
R: hd
S: rg

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
a=mptime:10
```

The response indicates that the transaction failed, because the phone was already off-hook. Consequently, neither a connection-id nor a session description is returned:

```
401 1205 Phone off-hook
```

Our third example illustrates the use of the provisional response and the three-way handshake. We create another connection this time using dynamic quality of service and acknowledging the previous response received:

```
CRCX 1206 aaln/1@rgw-2569.whatever.net MGCP 1.0 NCS 1.0
K: 1205
C: A3C47F21456789F0
L: p:10, a:PCMU, dq-gi:A735C2
M: inactive

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
```

```
t=0 0
m=audio 3456 RTP/AVP 0 18
 a=mptime:10 10
```

A provisional response is returned initially:

```
100 1206 Pending
I: DFE233D1

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 3456 RTP/AVP 0
a=mptime:10
```

Note that the endpoint elected to support only the PCMU codec, i.e., payload number 0.

A little later, the final response is received:

```
200 1206 OK
K:
DQ-RI: A12D5F1
I: DFE233D1

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 3456 RTP/AVP 0
a=mptime:10
```

The Call Agent acknowledges the final response as requested:

```
000 1206
```

and the transaction is complete.

D.4 ModifyConnection

The first example shows a ModifyConnection command that simply sets the connection mode of a connection to “send/receive” – the “notified entity” is set as well:

```
MDCX 1209 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
N: ca@ca1.whatever.net
M: sendrecv
```

The response indicates that the transaction was successful:

```
200 1209 OK
```

In the second example, we pass a session description and include a notification request with the ModifyConnection command. The endpoint will start playing ring-back tones to the user until it detects audio on the connection specified for the media start event:

```
MDCX 1210 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
M: recvonly
X: 0123456789AE
R: hu, ma@FDE234C8
S: rt
```

```
v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 3456 RTP/AVP 0
a=mptime:10
```

The response indicates that the transaction was successful:

```
200 1206 OK
```

D.5 DeleteConnection (From the Call Agent)

In this example, the Call Agent simply instructs the embedded client to delete the connection FDE234C8 on the endpoint specified:

```
DLCX 1210 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
```

The response indicates success, and that the connection was deleted. Connection parameters for the connection are therefore included as well:

```
250 1210 OK
P: PS=1245, OS=62345, PR=780, OR=45123, PL=10, JI=27, LA=48,
                        PC/RPS=782, PC/ROS=45238, PC/RPL=5, PC/RJI=26
```

D.6 DeleteConnection (From the Embedded Client)

In this example, the embedded client sends a DeleteConnection command to the Call Agent to instruct it that a connection on the specified endpoint has been deleted. The ReasonCode specifies the reason for the deletion, and Connection Parameters for the connection are provided as well:

```
DLCX 1210 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
E: 900 – Hardware error
P: PS=1245, OS=62345, PR=780, OR=45123, PL=10, JI=27, LA=48,
                        PC/RPS=782, PC/ROS=45238, PC/RPL=5, PC/RJI=26
```

The Call Agent sends a success response to the gateway:

```
200 1210 OK
```

D.7 DeleteConnection (Multiple Connections From the Call Agent)

In the first example, the Call Agent instructs the embedded client to delete all connections related to call "A3C47F21456789F0" on the specified endpoint:

```
DLCX 1210 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
```

The response indicates success and that the connection(s) were deleted:

```
250 1210 OK
```

In the second example, the Call Agent instructs the embedded client to delete all connections related to all of the endpoints specified:

```
DLCX 1210 aaln/*@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
```

The response indicates success:

```
250 1210 OK
```

D.8 AuditEndpoint

In the first example, the Call Agent wants to learn what endpoints are present on the embedded client specified, hence the use of the "all of" wild-card for the local portion of the endpoint-name:

```
AUEP 1200 *@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
```

The embedded client indicates success and includes a list of endpoint names:

```
200 1200 OK
Z: aaln/1@rgw-2567.whatever.net
Z: aaln/2@rgw-2567.whatever.net
```

In the second example, the capabilities of one of the endpoints is requested:

```
AUEP 1201 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
F:A
```

The response indicates success and the capabilities as well. Two codecs are supported, however with different capabilities. Consequently two separate capability sets are returned. Again, each capability set is to be returned on a single line. The example below shows multiple lines due to formatting restraints:

```
200 1201 OK
A: a:PCMU,p:10-100,e:on,s:off, v:L;S, m:sendonly;
     recvonly;sendrecv;inactive;netwloop;netwtest
A: a:G729,p:30-90, e:on,s:on,v:L;S, m:sendonly;
     recvonly;sendrecv;inactive;confrnce;netwloop
```

In the third example, the Call Agent audits all possible information for the endpoint:

```
AUEP 2002 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
F: R,D,S,X,N,I,T,O,ES,VS,E,MD
```

The response indicates success:

```
200 2002 OK
 R: L/hd,L/hu,oc(N),[0-9](N)
D:
S: vmwi(+)
X: 0123456789B1
N: Call-agent@ca.whatever.net
I: 32F345E2
T: L/hd,L/hu,L/ft
O: hd,9,1,2
ES: hd
VS: MGCP 1.0, MGCP 1.0 NCS 1.0
E: 000
MD: 4000
```

The list of requested events contains three events. Where no package name is specified, the default package is assumed. The same goes for actions, so the default action – Notify - must therefore be assumed for the "L/hu" event. The omission of a value for the "digit map" means the endpoint currently does not have a digit map. There are currently no active time-out signals, however the OO signal "vmvi" is currently on and is consequently included – in this case it was parameterized, however the parameter could have been excluded. The current "notified entity" refers to an IP-address and only a single connection exists for the endpoint. The current value of DetectEvents is "ft", and the list of ObservedEvents contains the four events specified.

Finally, the event-states audited reveals that the phone was off-hook at the time the transaction was processed.

D.9 AuditConnection

The first example shows an AuditConnection command where we audit the CallId, NotifiedEntity, LocalConnectionOptions, Connection Mode, LocalConnectionDescriptor, and the Connection Parameters:

```
AUCX 2003 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
I: 32F345E2
F: C,N,L,M,LC,P
```

The response indicates success and includes information for the RequestedInfo:

```
200 2003 OK
C: A3C47F21456789F0
N: ca@ca1.whatever.net
L: p:10, a:PCMU
M: sendrecv
P: PS=395, OS=22850, PR=615, OR=30937, PL=7, JI=26, LA=47,
                        PC/RPS=615, PC/ROS=30937, PC/RPL=5, PC/RJI=26

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1296 RTP/AVP 0
a=mptime:10
```

In the second example, we request to audit RemoteConnectionDescriptor and LocalConnectionDescriptor:

```
AUCX 1203 aaln/2@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
I: FDE234C8
F: RC,LC
```

The response indicates success, and includes information for the RequestedInfo. In this case, no RemoteConnectionDescriptor exists, hence only the protocol version field is included for the RemoteConnectionDescriptor:

```
200 1203 OK

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1296 RTP/AVP 0
a=mptime:10

v=0
```

D.10 RestartInProgress

The first example illustrates a RestartInProgress message sent by an embedded client to inform the Call Agent that the specified endpoint will be taken out of service in 300 seconds:

```
RSIP 1200 aaln/1@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
RM: graceful
RD: 300
```

The Call Agent's response indicates that the transaction was successful:

```
200 1200 OK
```

In the second example, the RestartInProgress message sent by the embedded client informs the Call Agent, that all of the embedded client's endpoints are being placed in service in 0 seconds, i.e., they are back in service. The delay could have been omitted as well:

```
RSIP 1204 *@rgw-2567.whatever.net MGCP 1.0 NCS 1.0
RM: restart
RD: 0
```

The Call Agent's response indicates success, and furthermore provides the endpoints in question with a new "notified entity":

```
200 1204 OK
N: CA-1@ca.whatever.net
```

Alternatively, the command could have failed with a new "notified entity" as in:

```
521 1204 OK
N: CA-1@ca.whatever.net
```

In that case, the command would then have to be retried (as a new transaction) in order to satisfy the "restart procedure" (see Section 4.4.3.5), this time going to Call Agent "`CA-1@whatever.net`".

Appendix E. Example Call Flow

In this section we provide an example call flow between two embedded clients, EC-1 and EC-2. It should be noted, that this call flow, although a valid one, is merely an example that may or may not be used in practice.

In the call flow described in Table 24 below, CA refers to the Call Agent, CDB refers to a configuration database, and ACC refers to an accounting database.

Table 24 - Example Call Flow

Usr-1	EC-1	CA	CDB	ACC	EC-2	Usr-2
	<–	Notification Request				
	Ack	–>				
Off-hook	Notify	–>				
	<–	Ack				
(Dial-tone)	<–	Create Connection + Notification Request				
	Ack(SDP1)	–>				
Digits	Notify	–>				
	<–	Ack				
(progress)	<–	Notification Request				
	Ack	–>				
		Query(E.164)	–>			
		<–	IP			
		Create Connection(SDP1) + Notification Request	– – –	– – –	–>	
		<–	– – –	– – –	P-Ack(SDP2)	
		<–	– – –	– – –	Ack(SDP2)	(ringing)
		Ack	– – –	– – –	–>	
(ringback)	<–	Modify Connection(SDP2) + Notification Request				
	Ack	–>				
		<–	– – –	– – –	Notify	Off-hook
		Ack	– – –	– – –	– >	
	<–	ModifyConnection + Notification Request				
	Ack	–>				
	(cut in)	Call start	– – –	– >		
		Notification Request	– – –	– – –	–>	
		<–	– – –	– – –	Ack	
		(Call Established)				
		<–	– – –	– – –	Notify	on hook

Usr-1	EC-1	CA	CDB	ACC	EC-2	Usr-2
		Ack	– – –	– – –	–>	
	<–	Delete Connection				
		Delete Connection	– – –	– – –	–>	
	Ack (Perf Data)	–>				
		<–	– – –	– – –	Ack(Perf data)	
		Call end	– – –	–>		
		Notification Request	– – –	– – –	–>	
		<–	– – –	– – –	Ack	
On-hook	Notify	–>				
	<–	Ack				
	<–	Notification Request				
	Ack	–>				

During these exchanges the NCS profile of MGCP is used by the Call Agent to control both embedded clients. The exchanges occur on two sides.

The first command is a NotificationRequest, sent by the Call Agent to the ingress embedded client. The request will consist of the following lines:

```
RQNT 1201 aaln/1@ec-1.whatever.net MGCP 1.0 NCS 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AB
R: hd
```

The embedded client, at that point, is instructed to look for an off-hook event, and to report it. It will first send a response to the command, repeating in the response the transaction id that the Call Agent attached to the query and providing a return code indicating success:

```
200 1201 OK
```

When the off hook event is noticed, the embedded client sends a Notify message to the Call Agent:

```
NTFY 2001 aaln/1@ec-1.whatever.net MGCP 1.0 NCS 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AB
O: hd
```

The Call Agent immediately acknowledges the notification:

```
200 2001 OK
```

The Call Agent examines the services associated to an off hook event for this endpoint (it could take special actions in the case of a direct line, no current subscription, etc.). In most cases, it will send a combined CreateConnection and NotificationRequest command to create a connection, provide dial-tone, and collect DTMF digits[36]:

```
CRCX 1202 aaln/1@ec-1.whatever.net MGCP 1.0 NCS 1.0
```

[36] The actual digit map depends on dialing plan in the local area as well as services subscribed to. The digit map presented should be considered an example digit map only.

```
C: A3C47F21456789F0
L: p:10, a:PCMU
M: recvonly
N: ca@ca1.whatever.net:5678
X: 0123456789AC
R: hu, [0-9#*T](D)
D: (0T | 00T | [2-9]xxxxxx | 1[2-9]xxxxxxxxx | 011xx.T)
S: dl
```

The embedded client acknowledges the transaction, sending back the identification of the newly created connection and the session description used to receive audio data:

```
200 1202 OK
I: FDE234C8

v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
a=mptime:10
```

The SDP specification, in our example, specifies the address at which the embedded client is ready to receive audio data (128.96.41.1), the transport protocol (RTP), the RTP port (3456) and the audio profile (AVP). The audio profile refers to 9 [3], which defines that the payload type 0 has been assigned for G.711 u-law transmission (also, see [17]).

The embedded client will start accumulating digits according to the digit map. When a digit map match subsequently occurs, the embedded client will notify the observed events to the Call Agent:

```
NTFY 2002 aaln/1@ec-1.whatever.net MGCP 1.0 NCS 1.0
N: ca@ca1.whatever.net:5678
X: 0123456789AC
O: 1,2,0,1,8,2,9,4,2,6,6
```

The Call Agent immediately acknowledges that notification:

```
200 2002 OK
```

At this stage, the Call Agent will send a NotificationRequest, to stop collecting digits yet continue to watch for an on-hook transition. The Call Agent furthermore decides to acknowledge receipt of the responses for transaction 1202:

```
RQNT 1203 aaln/1@ec-1.whatever.net MGCP 1.0 NCS 1.0
K: 1202
X: 0123456789AD
R: hu
```

The embedded client immediately acknowledges that command:

```
200 1203 OK
```

The Call Agent must now create a connection on the egress embedded client, EC-2, and ring the phone attached to the embedded client as well. It does so by sending a combined CreateConnection and NotificationRequest command to the embedded client:

```
CRCX 2001 aaln/1@ec-2.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
L: p:10, a:PCMU
M: sendrecv
X: 0123456789B0
R: hd
S: rg


v=0
o=- 25678 753849 IN IP4 128.96.41.1
s=-
c=IN IP4 128.96.41.1
t=0 0
m=audio 3456 RTP/AVP 0
a=mptime:10
```

The egress embedded client, at that point, is instructed to ring the phone, and to look for an off-hook event, and report it. The off-hook event and ringing signal are synchronized, so when the off-hook event occurs, ringing will stop. The create connection portion of the command has the same parameters as the command sent to the ingress embedded client, with two differences:

- The endpoint identifier points towards the outgoing circuit,
- The message carries the session description returned by the ingress embedded client,
- Because the session description is present, the “mode” parameter is set to “send/receive”.

We observe that the call identifier is identical for the two connections. This is normal since the two connections belong to the same call.

We assume, this command does not finish executing immediately[37], and a provisional response is therefore returned by the egress embedded client acknowledging the command, sending in the session description its own parameters such as address, ports and RTP profile as well as the connection identifier for the new connection:

```
100 2001 Pending
I: 32F345E2


v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1297 RTP/AVP 0
a=mptime:10
```

[37] This could, e.g., be due to external resource reservation, although we did not include that in our example.

Once the transaction finishes execution, the embedded client sends the final response to the Call Agent, repeating the information it provided in the provisional response:

```
200 2001 OK
K:
I: 32F345E2

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1297 RTP/AVP 0
a=mptime:10
```

When the Call Agent receives the final response, it notices the presence of the empty Response Acknowledgement attribute and therefore issues a Response Acknowledgement for the transaction:

```
000 2001
```

The Call Agent will relay the information to the ingress embedded client, and instruct it to generate local ringback tones, using a combined ModifyConnection and NotificationRequest command:

```
MDCX 1204 aaln/1@ec-1.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
M: recvonly
X: 0123456789AE
R: hu
S: rt

v=0
o=- 4723891 7428910 IN IP4 128.96.63.25
s=-
c=IN IP4 128.96.63.25
t=0 0
m=audio 1297 RTP/AVP 0
a=mptime:10
```

The embedded client immediately acknowledges the modification:

```
200 1204 OK
```

At this stage, the Call Agent has established a half duplex transmission path. The phone attached to the ingress embedded client will be able to receive the signals, such as tones or announcements, that may be generated in case of any errors, as well as the initial speech that most likely will be generated when the egress user answers the phone.

When the off hook event is observed, the egress embedded client sends a Notify message to the Call Agent:

```
NTFY 3001 aaln/1@ec-2.whatever.net MGCP 1.0 NCS 1.0
X: 0123456789B0
O: hd
```

The call agent immediately acknowledges that notification:

```
200 3001 OK
```

The Call agent now sends a combined ModifyConnection and NotificationRequest to the ingress embedded client, to place the connection in send/receive mode and stop the ringback tones:

```
MDCX 1206 aaln/1@ec-1.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
M: sendrecv
X: 0123456789AF
R: hu
```

The embedded client immediately responds to the command:

```
200 1206 OK
```

In parallel, the Call Agent asks the egress embedded client to notify the occurrence of an on-hook event. It does so by sending a NotificationRequest to the embedded client[38]:

```
RQNT 2002 aaln/1@ec-2.whatever.net MGCP 1.0 NCS 1.0
X: 0123456789B1
R: hu
```

The embedded client immediately responds to the command:

```
200 2002 OK
```

At this point, the call is fully established.

At some later point in time, the phone attached to the egress embedded client, in our scenario, goes on-hook. This event is notified to the Call Agent, according to the policy received in the last NotificationRequest by sending a Notify command:

```
NTFY 2003 aaln/1@ec-2.whatever.net MGCP 1.0 NCS 1.0
X: 0123456789B1
O: hu
```

The Call Agent immediately responds to the command:

```
200 2003 OK
```

The Call Agent now determines that the call is ending, and it therefore sends both embedded clients a DeleteConnection command:

```
DLCX 1207 aaln/1@ec-1.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
I: FDE234C8
DLCX 2004 aaln/1@ec-2.whatever.net MGCP 1.0 NCS 1.0
C: A3C47F21456789F0
I: 32F345E2
```

The embedded clients will respond with acknowledgements that include the connection parameters for the connection:

[38] It should be noted, that although on-hook is a persistent event, lockstep mode requires the Call Agent to send a new NotificationRequest to the embedded client.

```
250 1207 OK

P: PS=1245, OS=62345, PR=780, OR=45123, PL=10, JI=27, LA=48,
                PC/RPS=790, PC/ROS=45700, PC/RPL=15, PC/RJI=26

250 2004 OK

P: PS=790, OS=45700, PR=1230, OR=61875, PL=15, JI=27, LA=48,
                PC/RPS=1245, PC/ROS=62345, PC/RPL=10, PC/RJI=27
```

The Call Agent will also issue a new NotificationRequest to the egress embedded client, to be ready to receive the next off-hook event detected by the embedded client:

```
RQNT 2005 aaln/1@ec-2.whatever.net MGCP 1.0 NCS 1.0

X: 0123456789B2
R: hd
```

The embedded client will acknowledge this message:

```
200 2005 OK
```

Finally, the ingress embedded client hangs up the phone thereby generating a Notify message to the Call Agent:

```
NTFY 1208 aaln/1@ec-1.whatever.net MGCP 1.0 NCS 1.0

X: 0123456789AF
O: hu
```

The Call Agent immediately responds to the command:

```
200 1208 OK
```

The Call Agent will then issue a new NotificationRequest to the ingress embedded client, to be ready to receive the next off-hook event detected by the embedded client:

```
RQNT 1209 aaln/1@ec-1.whatever.net MGCP 1.0 NCS 1.0

X: 0123456789B3
R: hd
```

The embedded client will acknowledge this message:

```
200 1209 OK
```

Both embedded clients, at this point, are ready for the next call.

Appendix F. Compatibility Information

This appendix provides NCS protocol compatibility information.

F.1 MGCP Compatibility

NCS is a profile of MGCP 1.0 [1], however NCS has introduced a couple of additions as well. The following lists NCS additions that are currently not included in MGCP:

- **Endpoint Naming Scheme** The rules for wildcarding are more restrictive than in MGCP.
- **Embedded ModifyConnection** A new Embedded ModifyConnection action has been introduced.
- **Dynamic Quality of Service** PacketCable Security services are supported in NCS. This affects the LocalConnectionOptions, Capabilities, and SDP. Also, a new return parameter; ResourceID, is added for CreateConnection and ModifyConnection.
- **Security** PacketCable Security services are supported in NCS. This affects the LocalConnectionOptions, Capabilities, and SDP.
- **Endpoint Name Retrieval** The AuditEndpoint command has been extended with a capability to return the number of endpoints that match a wildcard as well as mechanism for block-wise retrieval of these endpoint names. Besides extending the AuditEndpoint command, this implies the introduction of two new parameter names; MaxEndPointIds, and NumEndPoints.
- **Supported Versions** The RestartInProgress response and the AuditEndpoint command have been extended with a VersionSupported parameter to enable Call Agents and gateways to determine which protocol versions each support.
- **Error Codes** Two new error codes have been introduced; 532 and 533.
- **Usage of SDP** A new SDP usage profile is included in NCS. Most notably, the profile and all example use specifically require strict SDP compliance, regardless of the usefulness of the included fields. Also, PacketCable specific extensions have been added to SDP.
- **Provisional Response** Additional detail and specification of the provisional response mechanism has been included in NCS. A Response Acknowledgement response (000) has been introduced, an empty ResponseAck parameter has been permitted in final responses that follow provisional responses, and a procedure for the mechanism specified.
- **Signal Parameters** Signal parameter syntax has been extended to allow for the usage of balanced parenthesis within signal parameters. All Time-Out signals can have their time-out value altered by a signal parameter.
- **Event Packages** NCS introduces a set of new event packages.
- **Packetization Period** A new multiple packetization period LocalConnectionOption has been defined, and the MTA is not allowed to choose codecs with a frame size that is inconsistent with the packetization period(s) specified by the Call Agent. Also, codec negotiation differs by depending on and negotiating packetization periods as well as codecs.

Finally, it should be noted, that NCS provides interpretations of and in some cases additional specification or clarification of the base MGCP protocol behavior that may or may not reflect the intended MGCP behavior.

Appendix G. ABNF Grammar for NCS

RFC 3435 includes a formal description of the MGCP protocol syntax following the "Augmented BNF for Syntax Specifications". This formal description is referenced by developers for the creation of interoperable devices. A copy of the MGCP protocol syntax, annotated and edited to indicate its applicability to PacketCable specifications, is provided in this appendix.

Implementations SHOULD conform to the portions of this ABNF grammar that relate to their respective specifications, i.e. NCS, TGCP. Also note that there are a few parameter encodings (e.g., embedded request, digit maps, vendor extension names) where the NCS grammar and/or TGCP grammar differ from the MGCP grammar.

Five annotations are used to distinguish between four different cases:

1. The language of the RFC has been changed to accommodate NCS and TGCP requirements.

2. The language of the RFC is applicable to NCS only (and possibly MGCP).

3. The language of the RFC is applicable to TGCP only (and possibly MGCP).

4. The language of the RFC s only applicable to NCS and TGCP.

5. The language of the RFC s only applicable to MGCP.

The language in each case is indicated by a different typeface, as defined in the change description below.

;RFC 3435 grammar changed to accommodate NCS and TGCP
;Bold indicates NCS only (and possibly MGCP)
;Italics indicates TGCP only (and possibly MGCP)
;Bold italics indicates NCS and TGCP only
;Text in grey is for MGCP only

```
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  = ALPHA 3(ALPHA / DIGIT) ; experimental starts with X

transaction-id = 1*9(DIGIT)

endpointName      = LocalEndpointName "@" DomainName
LocalEndpointName = LocalNamePart 0*("/" LocalNamePart)
LocalNamePart     = AnyName / AllName / NameString
AnyName           = "$"
AllName           = "*"
NameString        = 1*(range-of-allowed-characters)
; VCHAR except "$", "*", "/", "@"
range-of-allowed-characters  = %x21-23 / %x25-29 / %x2B-2E
                      / %x30-3F / %x41-7E

DomainName = 1*255(ALPHA / DIGIT / "." / "-")    ; as defined
```

```
        / "#" number                    ; in RFC 821
        / "[" IPv4address / IPv6address "]"   ; see RFC 2373

; Rewritten to ABNF from RFC 821
number =  1*DIGIT

;From RFC 2373
IPv6address = hexpart [ ":" IPv4address ]
IPv4address = 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT
; this production, while occurring in RFC2373, is not referenced
; IPv6prefix  = hexpart "/" 1*2DIGIT
hexpart = hexseq / hexseq "::" [ hexseq ] / "::" [ hexseq ]
hexseq  = hex4 *( ":" hex4)
hex4    = 1*4HEXDIG

MGCPversion = "MGCP" 1*(WSP) 1*(DIGIT) "." 1*(DIGIT)
                [1*(WSP) ProfileName]
ProfileName      = "NCS 1.0"    ; For NCS
                        / "TGCP 1.0"   ; For TGCP
          / VCHAR *( WSP / VCHAR)

MGCPParameter = ParameterValue EOL

; Check infoCode if more parameter values defined
; Most optional values can only be omitted when auditing
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])  ; For NCS (and MGCP)
          / ("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])
          / ("PL" ":" 0*(WSP)  [PackageList])    ; Auditing only
          / ("MD" ":" 0*(WSP)  MaxMGCPDatagram)   ; Auditing only
          / (extensionParameter ":" 0*(WSP) [parameterString])
          / VersionSupported  ; NCS and TGCP - response only
          / MaxEndpointIds    ; NCS and TGCP
          / NumEndpoints      ; NCS and TGCP - response only
```

```
; <extensionParameter> ":" parameterString defined by NCS and TGCP
VersionSupported = "VS" ":" MGCPversion *( "," 0*(WSP) MGCPversion)
MaxEndpointIds   = "ZM" ":" 0*(WSP) 1*16(DIGIT)
NumEndpoints     = "ZN" ":" 0*(WSP) 1*16(DIGIT)  ; Responses only

; A final response may include an empty ResponseAck
ResponseAck =  confirmedTransactionIdRange
            *( ","  0*(WSP) confirmedTransactionIdRange )

confirmedTransactionIdRange = transaction-id ["-" transaction-id]

BearerInformation = BearerAttribute 0*("," 0*(WSP) BearerAttribute)
BearerAttribute   = ("e" ":" BearerEncoding)
                  / (BearerExtensionName [":" BearerExtensionValue])
BearerExtensionName  = PackageLCOExtensionName
BearerExtensionValue = LocalOptionExtensionValue
BearerEncoding = "A" / "mu"
CallId = 1*32(HEXDIG)

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

NotifiedEntity = [LocalName "@"] DomainName [":" portNumber]
LocalName  = LocalEndpointName           ; No internal structure

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) ; Only for capabilities in
                                        ; NCS and TGCP
                 / ("e"  ":" echoCancellation)
                 / ("gc" ":" gainControl)
                 / ("s"  ":" silenceSuppression)
                 / ("t"  ":" typeOfService)
                 / ("r"  ":" resourceReservation)
                 / ("k"  ":" encryptiondata)
                 / ("nt" ":" ( typeOfNetwork /
                               supportedTypeOfNetwork))
                 / (LocalOptionExtensionName
                       [":" LocalOptionExtensionValue])
                 / MPacketizationPeriod   ; NCS and TGCP only
                 / RTPciphersuite         ; NCS and TGCP only
                 / RTCPciphersuite        ; NCS and TGCP only
                 / DQoSGateID             ; NCS only
                 / DQoSReservation        ; NCS only
                 / DQoSResourceID         ; NCS only
                 / DQoSReserveDestination ; NCS only
                 / CallContentId          ; TGCP only
                 / CallContentDestination ; TGCP only
```

```
Capabilities    =  CapabilityValue 0*(WSP)
                   0*("," 0*(WSP) CapabilityValue 0*(WSP))
CapabilityValue = LocalOptionValue
                / ("v" ":" supportedPackages)
                / ("m" ":" supportedModes)

PackageList     = pkgNameAndVers 0*("," pkgNameAndVers)
pkgNameAndVers  = packageName ":" packageVersion
packageVersion  = 1*(DIGIT)

; For NCS and TGCP, range format is only allowed for capabilities
; and not for LocalConnectionOptions.
packetizationPeriod  = 1*4(DIGIT) ["-" 1*4(DIGIT)]
compressionAlgorithm = algorithmName 0*(";" algorithmName)
algorithmName        = 1*(SuitableLCOCharacter)
bandwidth            = 1*4(DIGIT) ["-" 1*4(DIGIT)]
echoCancellation     = "on" / "off"
gainControl          = "auto" / ["-"] 1*4(DIGIT)
silenceSuppression   = "on" / "off"
typeOfService        = 1*2(HEXDIG)     ; 1 hex only for capabilities
resourceReservation  = "g" / "cl" / "be"

;encryption parameters are coded as in SDP (RFC 2327)
;NOTE: encryption key may contain an algorithm as specified in RFC 1890
encryptiondata = ( "clear" ":" encryptionKey )
               / ( "base64" ":" encodedEncryptionKey )
               / ( "uri" ":" URItoObtainKey )
               / ( "prompt" ) ; defined in SDP, not usable in MGCP!
encryptionKey = 1*(SuitableLCOCharacter) / quotedString
; See RFC 2045
encodedEncryptionKey = 1*(ALPHA / DIGIT / "+" / "/" / "=")
URItoObtainKey = 1*(SuitableLCOCharacter) / quotedString

typeOfNetwork = "IN" / "ATM" / "LOCAL" / OtherTypeOfNetwork
; Registered with IANA - see RFC 2327
OtherTypeOfNetwork     = 1*(SuitableLCOCharacter)
supportedTypeOfNetwork = typeOfNetwork *(";" typeOfNetwork)
supportedModes    = ConnectionMode 0*(";" ConnectionMode)

supportedPackages = packageName 0*(";" packageName)

packageName = 1*(ALPHA / DIGIT / HYPHEN) ; Hyphen neither first or last

LocalOptionExtensionName = VendorLCOExtensionName
                         / PackageLCOExtensionName
                         / OtherLCOExtensionName
VendorLCOExtensionName   = "x" ("+"/"-") 1*32(SuitableExtLCOCharacter)
PackageLCOExtensionName  = packageName "/"
                           1*32(SuitablePkgExtLCOCharacter)
; must not start with "x-" or "x+"
OtherLCOExtensionName    = 1*32(SuitableExtLCOCharacter)

; <LocalOptionExtensionName> ":" <LocalOptionExtensionvalue>
; defined by NCS/TGCP
MPacketizationPeriod = "mp" ":" multiplepacketizationPeriod
multiplepacketizationPeriod = mpPeriod 0*(";" mpPeriod)
```

```
mpPeriod          = 1*4(DIGIT) / HYPHEN

RTPciphersuite   = "sc-rtp"  ":" ciphersuite
RTCPciphersuite  = "sc-rtcp" ":" ciphersuite
ciphersuite      = [AuthenticationAlgorithm] "/" [EncryptionAlgorithm]
AuthenticationAlgorithm = 1*( ALPHA / DIGIT / "-" / "_" )
EncryptionAlgorithm     = 1*( ALPHA / DIGIT / "-" / "_" )

; <LocalOptionExtensionName> ":" <LocalOptionExtensionvalue>
; defined by NCS only
DQoSGateID          = "dq-gi" [":" 1*8(HEXDIG)]  ; Only empty for
                                                 ; capabilities
DQoSReservation     = "dq-rr" ":" DQoSResMode *(";" DQoSResMode)
DQoSResMode         = "sendresv" / "recvresv" / "snrcresv" /
                      "sendcomt" / "recvcomt" / "snrccomt"
DQoSResourceID      = "dq-ri" ":" 1*8(HEXDIG)
DQoSReserveDestination = "dq-rd" ":" IPv4address [":" portNumber]

; <LocalOptionExtensionName> ":" <LocalOptionExtensionvalue>
; defined by TGCP only
CallContentId                   = "es-cci" ":" 1*8(HEXDIG)
CallContentDestination                  = "es-ccd" ":" IPv4address ":" portNumber

LocalOptionExtensionValue = (1*(SuitableExtLCOValChar)
                                / quotedString)
                  *(";" (1*(SuitableExtLCOValChar)
                                / quotedString))

;Note: No "data" mode.
ConnectionMode = "sendonly" / "recvonly" / "sendrecv"
          / "confrnce" / "inactive"
          / "loopback" / "conttest" ; TGCP (and MGCP) only
          / "replcate" ; NCS only
          / "netwloop" / "netwtest"
          / ExtensionConnectionMode
ExtensionConnectionMode = PkgExtConnectionMode
PkgExtConnectionMode    = packageName "/" 1*(ALPHA / DIGIT)

RequestedEvents = requestedEvent 0*("," 0*(WSP) requestedEvent)
requestedEvent  = (eventName ["(" requestedActions ")"])
           / (eventName "(" requestedActions ")"
                        "(" eventParameters ")" )
eventName = [(packageName / "*") "/"]
          (eventId / "all" / eventRange
                                                          / "*" / "#") ; for DTMF
                    ["@" (ConnectionId / "$" / "*")]
eventId = 1*(ALPHA / DIGIT / HYPHEN)   ; Hyphen neither first nor last
eventRange = "[" 1*(DigitMapLetter / (DIGIT "-" DIGIT) /
                 (DTMFLetter "-" DTMFLetter)) "]"
DTMFLetter = "A" / "B" / "C" / "D"

requestedActions = requestedAction 0*("," 0*(WSP) requestedAction)
requestedAction  = "N" / "A"
           / "D" ; For NCS (and MGCP)
```

```
            / "S" / "I" / "K"
            / "E" "(" EmbeddedRequest ")"
            / ExtensionAction
            / "C" "(" EmbeddedModeChange     ; For NCS and TGCP
                0*("," 0*WSP EmbeddedModeChange) ")"   ; only
; NCS and TGCP define the Embedded ModifyConnection action.
; MGCP grammar does not allow for the format used in NCS and TGCP:
EmbeddedModeChange   = "M" "(" ConnectionMode "(" EmConnectionId ")" ")"
EmConnectionId       = ConnectionId / "$"

ExtensionAction  = PackageExtAction
PackageExtAction = packageName "/" Action ["(" ActionParameters ")"]
Action           = 1*ALPHA

ActionParameters = eventParameters        ; May contain actions

;NOTE: Should tolerate different order when receiving, e.g., for NCS
EmbeddedRequest = (      "R" "(" EmbeddedRequestList ")"
                  ["," 0*(WSP) "S" "(" EmbeddedSignalRequest ")"]
                  ["," 0*(WSP) "D" "(" EmbeddedDigitMap ")"]      )
                / (      "S" "(" EmbeddedSignalRequest ")"
                  ["," 0*(WSP) "D" "(" EmbeddedDigitMap ")"] )
                / (      "D" "(" EmbeddedDigitMap ")" )
                / NCSTGCPEmbeddedRequest
;Text below is for NCS and TGCP only. The difference compared to MGCP
;is simply that the order of the items is not fixed. Also for TGCP Digit Maps
; are not used
NCSTGCPEmbeddedRequest = NCSTGCPEmbeddedRequestItem
                                    *2("," 0*(WSP) NCSTGCPEmbeddedRequestItem)
NCSTGCPEmbeddedRequestItem = ("R" "(" EmbeddedRequestList ")" )
                                    / ("S" "(" EmbeddedSignalRequest ")" )
                                    / ("D" "(" EmbeddedDigitMap ")" )


EmbeddedRequestList   = RequestedEvents
EmbeddedSignalRequest = SignalRequests
EmbeddedDigitMap = DigitMap

SignalRequests   = SignalRequest 0*("," 0*(WSP) SignalRequest )
SignalRequest    = eventName [ "(" eventParameters ")" ]

eventParameters  = eventParameter 0*("," 0*(WSP) eventParameter)
eventParameter   = eventParameterValue
            / eventParameterName "=" eventParameter
            / eventParameterName "(" eventParameters ")"
eventParameterString = 1*(SuitableEventParamCharacter)
eventParameterName   = eventParameterString
eventParameterValue  = eventParameterString / quotedString

; For NCS (and MGCP)
DigitMap          = DigitString  / "(" DigitStringList ")"
DigitStringList   = DigitString 0*( "|" DigitString )
DigitString       = 1*(DigitStringElement)
DigitStringElement = DigitPosition ["."]
DigitPosition     = DigitMapLetter / DigitMapRange
; NOTE "X" is now included
```

```
DigitMapLetter     = DIGIT / "#" / "*" / "A" / "B" / "C" / "D" / "T"
                   / "X" / ExtensionDigitMapLetter
ExtensionDigitMapLetter = "E" / "F" / "G" / "H" / "I" / "J" / "K"
                        / "L" / "M" / "N" / "O" / "P" / "Q" / "R"
                        / "S" / "U" / "V" / "W" / "Y" / "Z"
; NOTE "[x]" is now allowed in MGCP.
; In NCS, only the "x" form is allowed
DigitMapRange = "[" 1*DigitLetter "]"
              / "X" ; Added for NCS only
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 )
                     / RemotePacketsSent
                     / RemoteOctetsSent
                     / RemotePacketsLost
                     / RemoteJitter
; NCS and TGCP define the following four connection parameter extension
; names:
RemotePacketsSent    = "PC/RPS" "=" packetsSent
RemoteOctetsSent     = "PC/ROS" "=" octetsSent
RemotePacketsLost    = "PC/RPL" "=" packetsLost
RemoteJitter         = "PC/JI"  "=" jitter


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 = VendorCPExtensionName
                                 / PackageCPExtensionName
VendorCPExtensionName  = "X" "-" 2*ALPHA
                       / NCSTGCPVendorCPExtensionName
;Text below is for NCS and TGCP only. The difference compared to MGCP
;is simply that MGCP requires 2 alpha characters whereas NCS and TGCP
;allow 2 or 3 alpha characters for VendorCPExtensionName
NCSTGCPVendorCPExtensionName         = "X" "-" 2*3ALPHA
```

```
PackageCPExtensionName = packageName "/" CPName
CPName = 1*(ALPHA / DIGIT / HYPHEN)
ConnectionParameterExtensionValue = 1*9(DIGIT)

MaxMGCPDatagram = 1*9(DIGIT)

ReasonCode = 3DIGIT
        [1*(WSP) "/" packageName]    ; Only for 8xx
        [WSP 1*(%x20-7E)]

SpecificEndpointID = endpointName
SecondEndpointID   = endpointName

RequestedInfo = infoCode 0*("," 0*(WSP) infoCode)

infoCode = "B" / "C" / "I" / "N" / "X" / "L" / "M" / "R" / "S"
      / "D" ; For NCS (and MGCP) only
            / "O" / "P" / "E" / "Z" / "Q" / "T" / "RC" / "LC"
            / "A" / "ES" / "RM" / "RD" / "PL" / "MD" / extensionParameter
      / "VS" / "ZM" / "ZN" ; NCS and TGCP define these
                  ; three extensionParameters

;NCS and TGCP allows for process and loop control in either order
QuarantineHandling = loopControl / processControl
              / (loopControl "," 0*(WSP) processControl )
              / (processControl "," 0*(WSP) loopControl)
loopControl    = "step" / "loop"
processControl = "process" / "discard"

DetectEvents = SignalRequests

RestartMethod = "graceful" / "forced" / "restart" / "disconnected"
          / "cancel-graceful" / extensionRestartMethod
extensionRestartMethod = PackageExtensionRM
PackageExtensionRM     = packageName "/" 1*32(ALPHA / DIGIT / HYPHEN)
RestartDelay = 1*6(DIGIT)

extensionParameter = VendorExtensionParameter
              / PackageExtensionParameter
              / OtherExtensionParameter
VendorExtensionParameter  = "X" ("-"/"+") 1*6(ALPHA / DIGIT)
PackageExtensionParameter = packageName "/"
                  1*32(ALPHA / DIGIT / HYPHEN)
; must not start with "x-" or x+"
OtherExtensionParameter   = 1*32(ALPHA / DIGIT / HYPHEN)

;If first character is a double-quote, then it is a quoted-string
parameterString = (%x21 / %x23-7F) *(%x20-7F) ; first and last must not
                                ; be white space
                                      / quotedString

MGCPResponse = MGCPResponseLine 0*(MGCPParameter)
                                                      *2(EOL *SDPinformation)

MGCPResponseLine = responseCode 1*(WSP) transaction-id
              [1*(WSP) "/" packageName]    ; Only for 8xx
```

```
                        [WSP responseString] EOL
responseCode = 3DIGIT
responseString = *(%x20-7E)

SuitablePkgExtLCOCharacter =  SuitableLCOCharacter

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

SuitableLCOCharacter   = SuitableExtLCOCharacter / "/"

SuitableExtLCOValChar  = SuitableLCOCharacter / ":"

; VCHAR except '"', "(", ")", ",", and "="
SuitableEventParamCharacter = %x21 / %x23-27 / %x2A-2B
                / %x2D-3C / %x3E-7E

; NOTE: UTF8 encoded
quotedString  = DQUOTE 0*(quoteEscape / quoteChar) DQUOTE
quoteEscape   = DQUOTE DQUOTE
quoteChar = (%x00-21 / %x23-FF)

EOL = CRLF / LF

HYPHEN = "-"

; See RFC 2327 for proper SDP grammar instead.
SDPinformation = SDPLine CRLF *(SDPLine CRLF)       ; see RFC 2327
SDPLine       = 1*(%x01-09 / %x0B / %x0C / %x0E-FF) ; for proper def.
```

Appendix H. Terms and definitions

H.1 Terms and definitions

PacketCable specifications use the following terms:

Access Control	Limiting the flow of information from the resources of a system only to authorized persons, programs, processes, or other system resources on a network.
Active	A service flow is said to be "active" when it is permitted to forward data packets. A service flow must first be admitted before it is active.
Admitted	A service flow is said to be "admitted" when the CMTS has reserved resources (e.g., bandwidth) for it on the DOCSIS™ network.
A-link	A-Links are SS7 links that interconnect STPs and either SSPs or SCPs. 'A' stands for "Access."
Asymmetric Key	An encryption key or a decryption key used in public key cryptography, where encryption and decryption keys are always distinct.
Audio Server	An Audio Server plays informational announcements in PacketCable network. Media announcements are needed for communications that do not complete and to provide enhanced information services to the user. The component parts of Audio Server services are Media Players and Media Player Controllers.
Authentication	The process of verifying the claimed identity of an entity to another entity.
Authenticity	The ability to ensure that the given information is without modification or forgery and was in fact produced by the entity that claims to have given the information.
Authorization	The act of giving access to a service or device if one has permission to have the access.
Cipher	An algorithm that transforms data between plaintext and ciphertext.
Ciphersuite	A set which must contain both an encryption algorithm and a message authentication algorithm (e.g., a MAC or an HMAC). In general, it may also contain a key-management algorithm, which does not apply in the context of PacketCable.
Ciphertext	The (encrypted) message output from a cryptographic algorithm that is in a format that is unintelligible.
Cleartext	The original (unencrypted) state of a message or data. Also called plaintext.
Confidentiality	A way to ensure that information is not disclosed to anyone other then the intended parties. Information is encrypted to provide confidentiality. Also known as privacy.
Cryptanalysis	The process of recovering the plaintext of a message or the encryption key without access to the key.
Cryptographic algorithm	An algorithm used to transfer text between plaintext and ciphertext.
Decipherment	A procedure applied to ciphertext to translate it into plaintext.
Decryption	A procedure applied to ciphertext to translate it into plaintext.
Decryption key	The key in the cryptographic algorithm to translate the ciphertext to plaintext.

Digital certificate	A binding between an entity's public key and one or more attributes relating to its identity, also known as a public key certificate.
Digital signature	A data value generated by a public-key algorithm based on the contents of a block of data and a private key, yielding an individualized cryptographic checksum.
Downstream	The direction from the headend toward the subscriber location.
Encipherment	A method used to translate plaintext into ciphertext.
Encryption	A method used to translate plaintext into ciphertext.
Encryption Key	The key used in a cryptographic algorithm to translate the plaintext to ciphertext.
Endpoint	A Terminal, Gateway or Multipoint Conference Unit (MCU).
Errored Second	Any 1-second interval containing at least one bit error.
Event Message	A message capturing a single portion of a connection.
F-link	F-Links are SS7 links that directly connect two SS7 end points, such as two SSPs. 'F' stands for "Fully Associated."
Flow [DOCSIS Flow]	(a.k.a. DOCSIS-QoS "service flow") A unidirectional sequence of packets associated with a Service ID (SID) and a QoS. Multiple multimedia streams may be carried in a single DOCSIS Flow.
Flow [IP Flow]	A unidirectional sequence of packets identified by OSI Layer 3 and Layer 4 header information. This information includes source/destination IP addresses, source/destination port numbers, protocol ID. Multiple multimedia streams may be carried in a single IP Flow.
Gateway	Devices bridging between the PacketCable IP Voice Communication world and the PSTN. Examples are the Media Gateway, which provides the bearer circuit interfaces to the PSTN and transcodes the media stream, and the Signaling Gateway, which sends and receives circuit switched network signaling to the edge of the PacketCable network.
H.323	An ITU-T recommendation for transmitting and controlling audio and video information. The H.323 recommendation requires the use of the ITU-T H.225 and ITU-T H.245 protocol for communication control between a "gateway" audio/video endpoint and a "gatekeeper" function.
Header	Protocol control information located at the beginning of a protocol data unit.
Integrity	A way to ensure that information is not modified except by those who are authorized to do so.
IntraLATA	Within a Local Access Transport Area.
Jitter	Variability in the delay of a stream of incoming packets making up a flow such as a voice communication.
Kerberos	A secret-key network authentication protocol that uses a choice of cryptographic algorithms for encryption and a centralized key database for authentication.
Key	A mathematical value input into the selected cryptographic algorithm.
Key Exchange	The swapping of public keys between entities to be used to encrypt communication between the entities.
Key Management	The process of distributing shared symmetric keys needed to run a security protocol.

Key Pair	An associated public and private key where the correspondence between the two are mathematically related, but it is computationally infeasible to derive the private key from the public key.
Keying Material	A set of cryptographic keys and their associated parameters, normally associated with a particular run of a security protocol.
Keyspace	The range of all possible values of the key for a particular cryptographic algorithm.
Latency	The time, expressed in quantity of symbols, taken for a signal element to pass through a device.
Link Encryption	Cryptography applied to data as it travels on data links between the network devices.
Network Layer	Layer 3 in the Open System Interconnection (OSI) architecture that provides network information that is independent from the lower layers.
Network Management	The functions related to the management of data across the network.
Network Management OSS	The functions related to the management of data link layer and physical layer resources and their stations across the data network supported by the hybrid fiber/coax system.
Nonce	A random value used only once that is sent in a communications protocol exchange to prevent replay attacks.
Non-Repudiation	The ability to prevent a sender from denying later that he or she sent a message or performed an action.
Off-Net Call	A communication connecting a PacketCable subscriber out to a user on the PSTN.
On-Net Call	A communication placed by one customer to another customer entirely on the PacketCable Network.
One-way Hash	A hash function that has an insignificant number of collisions upon output.
Plaintext	The original (unencrypted) state of a message or data. Also called cleartext.
Pre-shared Key	A shared secret key passed to both parties in a communication flow, using an unspecified manual or out-of-band mechanism.
Privacy	A way to ensure that information is not disclosed to any one other then the intended parties. Information is usually encrypted to provide confidentiality. Also known as confidentiality.
Private Key	The key used in public key cryptography that belongs to an individual entity and must be kept secret.
Proxy	A facility that indirectly provides some service or acts as a representative in delivering information, thereby eliminating the need for a host to support the service.
Public Key	The key used in public key cryptography that belongs to an individual entity and is distributed publicly. Other entities use this key to encrypt data to be sent to the owner of the key.
Public Key Certificate	A binding between an entity's public key and one or more attributes relating to its identity, also known as a digital certificate.

Public Key Cryptography	A procedure that uses a pair of keys, a public key and a private key, for encryption and decryption, also known as an asymmetric algorithm. A user's public key is publicly available for others to use to send a message to the owner of the key. A user's private key is kept secret and is the only key that can decrypt messages sent encrypted by the user's public key.
Root Private Key	The private signing key of the highest-level Certification Authority. It is normally used to sign public key certificates for lower-level Certification Authorities or other entities.
Root Public Key	The public key of the highest level Certification Authority, normally used to verify digital signatures generated with the corresponding root private key.
Secret Key	The cryptographic key used in a symmetric key algorithm, which results in the secrecy of the encrypted data depending solely upon keeping the key a secret, also known as a symmetric key.
Session Key	A cryptographic key intended to encrypt data for a limited period of time, typically between a pair of entities.
Signed and Sealed	An "envelope" of information which has been signed with a digital signature and sealed using encryption.
Subflow	A unidirectional flow of IP packets characterized by a single source and destination IP address and single source and destination UDP/TCP port.
Symmetric Key	The cryptographic key used in a symmetric key algorithm, which results in the secrecy of the encrypted data depending solely upon keeping the key a secret, also known as a secret key.
Systems Management	Functions in the application layer related to the management of various Open Systems Interconnection (OSI) resources and their status across all layers of the OSI architecture.
Transit Delays	The time difference between the instant at which the first bit of a Protocol Data Unit (PDU) crosses one designated boundary, and the instant at which the last bit of the same PDU crosses a second designated boundary.
Trunk	An analog or digital connection from a circuit switch that carries user media content and may carry voice signaling (M_F, R_2, etc.).
Tunnel Mode	An IPsec (ESP or AH) mode that is applied to an IP tunnel, where an outer IP packet header (of an intermediate destination) is added on top of the original, inner IP header. In this case, the ESP or AH transform treats the inner IP header as if it were part of the packet payload. When the packet reaches the intermediate destination, the tunnel terminates and both the outer IP packet header and the IPsec ESP or AH transform are taken out.
Upstream	The direction from the subscriber location toward the headend.
X.509 certificate	A public key certificate specification developed as part of the ITU-T X.500 standards directory.

H.2 Abbreviations

PacketCable specifications use the following abbreviations.

AAA	Authentication, Authorization and Accounting.
AES	Advanced Encryption Standard. A block cipher, used to encrypt the media traffic in PacketCable.
AF	Assured Forwarding. This is a DiffServ Per Hop Behavior.
AH	Authentication header. An IPsec security protocol that provides message integrity for complete IP packets, including the IP header.
AMA	Automated Message Accounting. A standard form of call detail records (CDRs) developed and administered by Bellcore (now Telcordia Technologies).
ASD	Application-Specific Data. A field in some Kerberos key management messages that carries information specific to the security protocol for which the keys are being negotiated.
AT	Access Tandem.
ATM	Asynchronous Transfer Mode. A protocol for the transmission of a variety of digital signals using uniform 53-byte cells.
BAF	Bellcore AMA Format, also known as AMA.
BCID	Billing Correlation ID.
BPI+	Baseline Privacy Plus Interface Specification. The security portion of the DOCSIS 1.1 standard that runs on the MAC layer.
CA	Certification Authority. A trusted organization that accepts certificate applications from entities, authenticates applications, issues certificates and maintains status information about certificates.
CA	Call Agent. The part of the CMS that maintains the communication state, and controls the line side of the communication.
CBC	Cipher Block Chaining mode. An option in block ciphers that combine (XOR) the previous block of ciphertext with the current block of plaintext before encrypting that block of the message.
CBR	Constant Bit Rate.
CDR	Call Detail Record. A single CDR is generated at the end of each billable activity. A single billable activity may also generate multiple CDRs.
CIC	Circuit Identification Code. In ANSI SS7, a two-octet number that uniquely identifies a DSO circuit within the scope of a single SS7 Point Code.
CID	Circuit ID (Pronounced “kid”). This uniquely identifies an ISUP DS0 circuit on a Media Gateway. It is a combination of the circuit’s SS7 gateway point code and Circuit Identification Code (CIC). The SS7 DPC is associated with the Signaling Gateway that has domain over the circuit in question.
CIF	Common Intermediate Format.
CIR	Committed Information Rate.
CM	DOCSIS Cable Modem.
CMS	Cryptographic Message Syntax.
CMS	Call Management Server. Controls the audio connections. Also called a Call Agent in MGCP/SGCP terminology. This is one example of an Application Server.

CMTS	Cable Modem Termination System. The device at a cable headend which implements the DOCSIS RFI MAC protocol and connects to CMs over an HFC network.
CMSS	Call Management Server Signaling.
Codec	COder-DECoder.
COPS	Common Open Policy Service protocol. Currently an internet draft, which describes a client/server model for supporting policy control over QoS Signaling Protocols and provisioned QoS resource management.
CoS	Class of Service. The type 4 tuple of a DOCSIS configuration file.
CRCX	Create Connection.
CSR	Customer Service Representative.
DA	Directory Assistance.
DE	Default. This is a DiffServ Per Hop Behavior.
DES	Data Encryption Standard.
DF	Delivery Function.
DHCP	Dynamic Host Configuration Protocol.
DHCP-D	DHCP Default. Network Provider DHCP Server.
DNS	Domain Name Service.
DOCSIS®	Data-Over-Cable Service Interface Specifications.
DPC	Destination Point Code. In ANSI SS7, a 3-octet number which uniquely identifies an SS7 Signaling Point, either an SSP, STP, or SCP.
DQoS	Dynamic Quality-of-Service. Assigned on the fly for each communication depending on the QoS requested.
DSA	Dynamic Service Add.
DSC	Dynamic Service Change.
DSCP	DiffServ Code Point. A field in every IP packet that identifies the DiffServ Per Hop Behavior. In IP version 4, the TOS byte is redefined to be the DSCP. In IP version 6, the Traffic Class octet is used as the DSCP.
DTMF	Dual-tone Multi Frequency (tones).
EF	Expedited Forwarding. A DiffServ Per Hop Behavior.
E-MTA	Embedded MTA. A single node that contains both an MTA and a cable modem.
EO	End Office.
ESP	IPsec Encapsulating Security Payload. Protocol that provides both IP packet encryption and optional message integrity, not covering the IP packet header.
ETSI	European Telecommunications Standards Institute.
F-link	F-Links are SS7 links that directly connect two SS7 end points, such as two SSPs. 'F' stands for "Fully Associated."
FEID	Financial Entity ID.
FGD	Feature Group D signaling.
FQDN	Fully Qualified Domain Name. Refer to IETF RFC 2821 for details.
GC	Gate Controller.
GTT	Global Title Translation.

HFC	Hybrid Fiber/Coaxial. An HFC system is a broadband bi-directional shared media transmission system using fiber trunks between the headend and the fiber nodes, and coaxial distribution from the fiber nodes to the customer locations.
HMAC	Hashed Message Authentication Code. A message authentication algorithm, based on either SHA-1 or MD5 hash and defined in IETF RFC 2104.
HTTP	Hypertext Transfer Protocol. Refer to IETF RFC 1945 and RFC 2068.
IANA	Internet Assigned Numbered Authority. See www.ietf.org for details.
IC	Inter-exchange Carrier.
IETF	Internet Engineering Task Force. A body responsible, among other things, for developing standards used on the Internet. See www.ietf.org for details.
IKE	Internet Key Exchange. A key-management mechanism used to negotiate and derive keys for SAs in IPsec.
IKE–	A notation defined to refer to the use of IKE with pre-shared keys for authentication.
IKE+	A notation defined to refer to the use of IKE with X.509 certificates for authentication.
IP	Internet Protocol. An Internet network-layer protocol.
IPsec	Internet Protocol Security. A collection of Internet standards for protecting IP packets with encryption and authentication.
ISDN	Integrated Services Digital Network.
ISTP	Internet Signaling Transport Protocol.
ISUP	ISDN User Part. A protocol within the SS7 suite of protocols that is used for call signaling within an SS7 network.
ITU	International Telecommunication Union.
ITU-T	International Telecommunication Union–Telecommunication Standardization Sector.
IVR	Interactive Voice Response system.
KDC	Key Distribution Center.
LATA	Local Access and Transport Area.
LD	Long Distance.
LIDB	Line Information Database. Contains customer information required for real-time access such as calling card personal identification numbers (PINs) for real-time validation.
LLC	Logical Link Control. The Ethernet Packet header and optional 802.1P tag which may encapsulate an IP packet. A sublayer of the Data Link Layer.
LNP	Local Number Portability. Allows a customer to retain the same number when switching from one local service provider to another.
LSSGR	LATA Switching Systems Generic Requirements.
MAC	Message Authentication Code. A fixed-length data item that is sent together with a message to ensure integrity, also known as a MIC.
MAC	Media Access Control. It is a sublayer of the Data Link Layer. It normally runs directly over the physical layer.
MC	Multipoint Controller.
MCU	Multipoint Conferencing Unit.
MD5	Message Digest 5. A one-way hash algorithm that maps variable length plaintext into fixed-length (16 byte) ciphertext.

MDCP	Media Device Control Protocol. A media gateway control specification submitted to IETF by Lucent. Now called SCTP.
MDCX	Modify Connection.
MDU	Multi-Dwelling Unit. Multiple units within the same physical building. The term is usually associated with high-rise buildings.
MEGACO	Media Gateway Control IETF working group. See www.ietf.org for details.
MF	Multi-Frequency.
MG	Media Gateway. Provides the bearer circuit interfaces to the PSTN and transcodes the media stream.
MGC	Media Gateway Controller. The overall controller function of the PSTN gateway. Receives, controls and mediates call-signaling information between the PacketCable and PSTN.
MGCP	Media Gateway Control Protocol. Protocol follow-on to SGCP. Refer to IETF 2705.
MIB	Management Information Base.
MIC	Message Integrity Code. A fixed-length data item that is sent together with a message to ensure integrity, also known as a Message Authentication Code (MAC).
MMC	Multi-Point Mixing Controller. A conferencing device for mixing media streams of multiple connections.
MSB	Most Significant Bit.
MSO	Multi-System Operator. A cable company that operates many headend locations in several cities.
MSU	Message Signal Unit.
MTA	Multimedia Terminal Adapter. Contains the interface to a physical voice device, a network interface, CODECs, and all signaling and encapsulation functions required for VoIP transport, class features signaling, and QoS signaling.
MTP	The Message Transfer Part. A set of two protocols (MTP 2, MTP 3) within the SS7 suite of protocols that are used to implement physical, data link, and network-level transport facilities within an SS7 network.
MWD	Maximum Waiting Delay.
NANP	North American Numbering Plan.
NANPNAT	North American Numbering Plan Network Address Translation.
NAT Network Layer	Network Address Translation. Layer 3 in the Open System Interconnection (OSI) architecture. This layer provides services to establish a path between open systems.
NCS	Network Call Signaling.
NPA-NXX	Numbering Plan Area (more commonly known as area code) NXX (sometimes called exchange) represents the next three numbers of a traditional phone number. The N can be any number from 2-9 and the Xs can be any number. The combination of a phone number's NPA-NXX will usually indicate the physical location of the call device. The exceptions include toll-free numbers and ported number (see LNP).
NTP	Network Time Protocol. An internet standard used for synchronizing clocks of elements distributed on an IP network.
NTSC	National Television Standards Committee. Defines the analog color television broadcast standard used today in North America.
OID	Object Identification.
OSP	Operator Service Provider.

OSS	Operations Systems Support. The back-office software used for configuration, performance, fault, accounting, and security management.
OSS-D	OSS Default. Network Provider Provisioning Server.
PAL	Phase Alternate Line. The European color television format that evolved from the American NTSC standard.
PCES	PacketCable Electronic Surveillance.
PCM	Pulse Code Modulation. A commonly employed algorithm to digitize an analog signal (such as a human voice) into a digital bit stream using simple analog-to-digital conversion techniques.
PDU	Protocol Data Unit.
PHS	Payload Header Suppression. A DOCSIS technique for compressing the Ethernet, IP, and UDP headers of RTP packets.
PKCROSS	Public-Key Cryptography for Cross-Realm Authentication. Utilizes PKINIT for establishing the inter-realm keys and associated inter-realm policies to be applied in issuing cross-realm service tickets between realms and domains in support of Intradomain and Interdomain CMS-to-CMS signaling (CMSS).
PKCS	Public-Key Cryptography Standards. Published by RSA Data Security Inc. These Standards describe how to use public key cryptography in a reliable, secure and interoperable way.
PKI	Public-Key Infrastructure. A process for issuing public key certificates, which includes standards, Certification Authorities, communication between authorities and protocols for managing certification processes.
PKINIT	Public-Key Cryptography for Initial Authentication. The extension to the Kerberos protocol that provides a method for using public-key cryptography during initial authentication.
PSC	Payload Service Class Table, a MIB table that maps RTP payload Type to a Service Class Name.
PSFR	Provisioned Service Flow Reference. An SFR that appears in the DOCSIS configuration file.
PSTN	Public Switched Telephone Network.
QCIF	Quarter Common Intermediate Format.
QoS	Quality of Service. Guarantees network bandwidth and availability for applications.
RADIUS	Remote Authentication Dial-In User Service. An internet protocol (IETF RFC 2865 and RFC 2866) originally designed for allowing users dial-in access to the internet through remote servers. Its flexible design has allowed it to be extended well beyond its original intended use.
RAS	Registration, Admissions and Status. RAS Channel is an unreliable channel used to convey the RAS messages and bandwidth changes between two H.323 entities.
RC4	Rivest Cipher 4. A variable length stream cipher. Optionally used to encrypt the media traffic in PacketCable.
RFC	Request for Comments. Technical policy documents approved by the IETF which are available on the World Wide Web at http://www.ietf.cnri.reston.va.us/rfc.html.
RFI	The DOCSIS Radio Frequency Interface specification.
RJ-11	Registered Jack-11. A standard 4-pin modular connector commonly used in the United States for connecting a phone unit into a wall jack.

RKS	Record Keeping Server. The device, which collects and correlates the various Event Messages.
RSA	A public-key, or asymmetric, cryptographic algorithm used to provide authentication and encryption services. RSA stands for the three inventors of the algorithm; Rivest, Shamir, Adleman.
RSA Key Pair	A public/private key pair created for use with the RSA cryptographic algorithm.
RSVP	Resource Reservation Protocol.
RTCP	Real-Time Control Protocol.
RTO	Retransmission Timeout.
RTP	Real-time Transport Protocol. A protocol for encapsulating encoded voice and video streams. Refer to IETF RFC 1889.
SA	Security Association. A one-way relationship between sender and receiver offering security services on the communication flow.
SAID	Security Association Identifier. Uniquely identifies SAs in the DOCSIS Baseline Privacy Plus Interface (BPI+) security protocol.
SCCP	Signaling Connection Control Part. A protocol within the SS7 suite of protocols that provides two functions in addition to those provided within MTP. The first function is the ability to address applications within a signaling point. The second function is Global Title Translation.
SCP	Service Control Point. A Signaling Point within the SS7 network, identifiable by a Destination Point Code that provides database services to the network.
SCTP	Stream Control Transmission Protocol.
SDP	Session Description Protocol.
SDU	Service Data Unit. Information delivered as a unit between peer service access points.
SF	Service Flow. A unidirectional flow of packets on the RF interface of a DOCSIS system.
SFID	Service Flow ID. A 32-bit integer assigned by the CMTS to each DOCSIS Service Flow defined within a DOCSIS RF MAC domain. SFIDs are considered to be in either the upstream direction (USFID) or downstream direction (DSFID). Upstream Service Flow IDs and Downstream Service Flow IDs are allocated from the same SFID number space.
SFR	Service Flow Reference. A 16-bit message element used within the DOCSIS TLV parameters of Configuration Files and Dynamic Service messages to temporarily identify a defined Service Flow. The CMTS assigns a permanent SFID to each SFR of a message.
SG	Signaling Gateway. An SG is a signaling agent that receives/sends SCN native signaling at the edge of the IP network. In particular, the SS7 SG function translates variants ISUP and TCAP in an SS7-Internet Gateway to a common version of ISUP and TCAP.
SGCP	Simple Gateway Control Protocol. Earlier draft of MGCP.
SHA – 1	Secure Hash Algorithm 1. A one-way hash algorithm.
SID	Service ID. A 14-bit number assigned by a CMTS to identify an upstream virtual circuit. Each SID separately requests and is granted the right to use upstream bandwidth.
SIP	Session Initiation Protocol. An application-layer control (signaling) protocol for creating, modifying, and terminating sessions with one or more participants.

SIP+	Session Initiation Protocol Plus. An extension to SIP.
S-MTA	Standalone MTA. A single node that contains an MTA and a non-DOCSIS MAC (e.g., ethernet).
SNMP	Simple Network Management Protocol.
SOHO	Small Office/Home Office.
SS7	Signaling System number 7. An architecture and set of protocols for performing out-of-band call signaling with a telephone network.
SSP	Service Switching Point. SSPs are points within the SS7 network that terminate SS7 signaling links and also originate, terminate, or tandem switch calls.
STP	Signal Transfer Point. A node within an SS7 network that routes signaling messages based on their destination address. This is essentially a packet switch for SS7. It may also perform additional routing services such as Global Title Translation.
TCAP	Transaction Capabilities Application Protocol. A protocol within the SS7 stack that is used for performing remote database transactions with a Signaling Control Point.
TCP	Transmission Control Protocol.
TD	Timeout for Disconnect.
TFTP	Trivial File Transfer Protocol.
TFTP-D	Default – Trivial File Transfer Protocol.
TGS	Ticket Granting Server. A sub-system of the KDC used to grant Kerberos tickets.
TGW	Telephony Gateway.
TIPHON	Telecommunications and Internet Protocol Harmonization Over Network.
TLV	Type-Length-Value. A tuple within a DOCSIS configuration file.
TN	Telephone Number.
ToD	Time-of-Day Server.
TOS	Type of Service. An 8-bit field of every IP version 4 packet. In a DiffServ domain, the TOS byte is treated as the DiffServ Code Point, or DSCP.
TSG	Trunk Subgroup.
UDP	User Datagram Protocol. A connectionless protocol built upon Internet Protocol (IP).
VAD	Voice Activity Detection.
VBR	Variable Bit Rate.
VoIP	Voice-over-IP.

Appendix I. Revision History

The following Engineering Change Notices were incorporated into PKT-SP-EC-MGCP-I03-010620:

ID	ECN Date
ncs-n-00038	1/22/01
mgcp-n-00053	9/22/00
mgcp-n-00054	6/9/00
mgcp-n-00055	6/9/00
mgcp-n-00056	6/9/00
mgcp-n-00057	6/9/00
mgcp-n-00058	6/9/00
mgcp-n-00059	6/9/00
mgcp-n-00060	6/9/00
mgcp-n-00070	9/11/00
mgcp-n-00071	9/11/00
mgcp-n-00072	9/11/00
mgcp-n-00074	9/11/00
mgcp-n-00075	9/11/00
mgcp-n-00076	9/11/00
mgcp-n-00102	1/21/01
mgcp-n-00127-v2	2/26/01
mgcp-n-00132	2/12/01
mgcp-n-00155-v2	3/5/01
mgcp-n-01010-v2	3/26/01
mgcp-n-01020	3/26/01
mgcp-n-01027-v3	5/21/01
mgcp-n 01036	5/7/01
mgcp-n-01043	5/7/01
mgcp-n-01054	5/14/01
mgcp-n-01055	5/21/01
mgcp-n-01057	5/21/01

The following Engineering Change Notices were incorporated into PKT-SP-EC-MGCP-I04-011221:

ID	ECN Date	Problem Description
mgcp-n-01093-v2	8/20/01	Ambiguity in existing text. Unclear when MTA is detecting digits.
mgcp-n-01105	37123	Clarification of Name parameter of Caller ID
Mgcp-n-01058-v2	6/18/01	Provide for use of RTP Named Telephony Events, as defined in RFC2833. Also allows for use of RFC2833 and other media types.
mgcp-n-01063-v2	7/23/01	Clarify whether a new transaction ID or the old transaction ID should be used.
mgcp-n-01064	6/11/01	Clarify whether the last sentence means to implement transaction atomicity (i.e. all or nothing) or it means to say previous success should be kept.
mgcp-n-01068	6/11/01	There is no explicit guideline a gateway MUST follow when doing endpoint assignment on its own.
mgcp-n-01074	6/18/01	The current text on retransmission of piggy-backed datagrams contains race conditions and may lead to infinitely sized datagrams that will also end up violating the maximum message lifetime
mgcp-n-01081	7/2/01	Clarify whether a new transaction ID should be used to re-initiate disconnect procedure
mgcp-n-01086	7/23/01	The current text (incl. modifications provided in MGCP-n-01063) does not clearly describe, that a new RestartInProgress command is only required to be sent when an error response is received for the "restart" or "disconnected" restartMethods.
mgcp-n-01095	8/6/01	The handling of NTFY's and "notification state" when becoming disconnected is not defined.
tgcp*-n-00033-v2	10/10/00	This ECR affects both TGCP and NCS. It permits the cancellation of a graceful shutdown of a gateway when maintenance personnel choose to do so.
mgcp-n-01067-v2	6/18/01	The specification requires an endpoint to generate an error when it is queried about a capability it does not understand. However, there is no NCS message encoding rule that supports specification of different options within “Capabilities” parameter.
mgcp-n-01089	7/16/01	The two separate signals (ois and oil) have been merged into a single timeout signal, osi.
mgcp-n-01094	7/30/01	Discrepancies between text in Section 4.4.2 and accompanying flowchart.
mgcp-n-01120-v3	10/22/01	The minimum digit map size that all an end-point has to support is specified as 2048 Bytes.
mgcp-n-01124	9/10/01	Clarification on reporting capabilities via response to AUEP
mgcp-n-01125	9/10/01	The dynamic payload type for a given codec is expected to be the same in the send and receive direction, however this is not stated explicitly anywhere.
mgcp-n-01166	10/22/01	Default value of Type of Service field ignores that this field may be provisioned.

ID	ECN Date	Problem Description
mgcp-n-01161	10/15/01	'Netwloop' and 'Netwtest' connection modes require sending of packets even when RemoteConnectionDescriptor has not been received. However, in this case the upstream resources are not committed and packets cannot be sent.
mgcp-n-01194	11/19/01	The IO3 NCS Specification needs to be updated to be consistent with latest Packet Cable Security Spec.
mgcp-n-01195	11/19/01	Need ability to specify more than one packetization period in Local Connection Options
mgcp-n-01196	11/19/01	We suggest an optional extension to NCS as per the MGCP 1.05 spec to accept the 'Q: loop' QuarantineHandling parameter which allows the MTA to generate multiple event notifications (NTFY) under one Request ID.
mgcp-n-01199	11/19/01	The current specification of the Audit Endpoint message does not allow the Call Agent to query the reason why an endpoint was taken out of service
mgcp-n-01155	12/10/01	The current MGCP specification does not say anything about the UDP packetsize. The revision specifies that a Call Agent has to support the maximum datagram size as 4K bytes.
sec-n-00049-v3	2/26/01	The current RTP key distribution over NCS signaling messages assumes that each endpoint knows if it is the initiator of a call or a target of a call. This assumption may be false in some cases.

The following ECNs were incorporated into PKT-SP-EC-MGCP-I05-021018:

EC ID	ECN Date	Problem Description:
mgcp-n-01227	6/24/02	The current specification only has vague references to the PacketCable Codec specification for defining the proper formatting of the literal names of vocoder algorithms.
mgcp-n-02015	6/24/02	The current specification indicates that some of the ringing cadences can be provisioned when in fact, the intention is that some of the ring cadences MUST be provisionable, while others need not be provisionable.
mgcp-n-02016	6/24/02	Spec does not clearly describe how to fill out the bandwidth parameter (b) of the SDP for use within the PacketCable system; add a reference
mgcp-n-02018	7/1/02	Clarify what action is to be taken by MTA when RTP/RTCP ciphersuite negotiation fails for any reason
mgcp-n-02024	6/24/02	This change simply clarifies the encoding of Reason Codes.
mgcp-n-02029	6/24/02	Need to more clearly define requirements of MTA for generating multiple signal requests simultaneously.
mgcp-n-02037	6/24/02	MTA MUST use Resource ID if provided by MTA and MUST return if provided by CMTS.
mgcp-n-02038	6/24/02	Clarification on the use of Embedded Requests.
mgcp-n-02039	6/24/02	How to calculate default timeout for call waiting tones.
mgcp-n-02089	7/1/02	Need to add back in clarification of disconnect timer handling steps that we inadvertently removed in the I04 specification.
mgcp-n-02099	7/1/02	Clarification of the meaning of an absent SignalRequests, RequestedEvents parameters.
mgcp-n-02101-	7/29/02	Current definition of packetization interval in the SDP (RFC-2327) is inadequate for PacketCable use, a new attribute is defined to alleviate this problem.
mgcp-n-02108	7/1/02	Change support of Q:loop from optional to mandatory.
mgcp-n-02117	7/1/02	Enhance the connection parameters to allow for remote endpoint statistics derived from RTCP information.
mgcp-n-02139	7/29/02	Provisional response timing requirement is too vague.

The following ECNs were incorporated into PKT-SP-EC-MGCP-I06-021127:

ID	ECN Date	Problem Description:
mgcp-n-02181	11/11/02	Expected MTA behavior in certain codec negotiation scenarios requires clarification.
mgcp-n-02184	11/14/02	The Notified Entity parameter syntax is not clear.
mgcp-n-02208	11/19/02	The ECR contains a number of editorial and minor technical clarifications.
mgcp-n-02209	11/18/02	CMSs and MTAs MUST ignore any SDP parameters, attributes, or fields it doesn't understand.

The following ECNs were incorporated into PKT-SP-EC-MGCP-I07-03415:

ID	ECN Date	Problem Description
mgcp-n-02218	12/30/02	The CMS MUST send a Final Response ACK to the MTA endpoint when a Final Response is received regardless of the "receipt" of a provisional response.
mgcp-n-02220	1/13/03	Pre-existing footnote disagrees with new ECN's and needs to be cleaned up.
mgcp-n-02224	2/10/03	Clarify that the CMS should silently ignore any unknown parameters found in a capabilities audit response.
mgcp-n-03008	2/24/03	Clarifies that On/Off signal state is not affected by empty SignalRequest; corrects errors in Appendix B and 4.3.1; replaces reference to RFC821 with RFC2821.
mgcp-n-03029	3/10/03	Addition of footnote to clarify TOS usage for media stream packets.
pkt-n-03006	3/3/03	Change definition of SFIDs in glossary.

The following ECNs were incorporated into PKT-SP-EC-MGCP-I08-030728:

ID	ECN Date	Problem Description
mgcp-n-03016	6/9/03	Clarifies how dynamic payload types are to be used in PacketCable.
mgcp-n-03035	5/5/03	Provides numbers and captions for figures and tables and inserts refs to labels in text.
mgcp-n-03055	6/30/03	Add mandatory requirements in Appendix C to clarify necessary MTA behavior.

The following ECNs were incorporated into PKT-SP-EC-MGCP-I09-040113:

ID	ECN Date	Problem Description
mgcp-n-03065	11/3/03	Clarify proper MTA behavior for DSX messaging when flow settings or GateId change as a result of a MDCX.
mgcp-n-03070	11/17/03	Clarification of several implied reciprocal requirements in the NCS specification.
mgcp-n-03109	12/1/03	Behavior of MTAs during shutdown procedures.
mgcp-n-03111	12/1/03	Add requirements to CMS not to violate SDP
mgcp-n-03119	12/1/03	The specification should be more clear about how Audit commands are responded to.

The following ECNs were incorporated into PKT-SP-EC-MGCP-I10-040402:

ID	ECN Date	Problem Description
EC-MGCP-N-04.0136-5	3/8/04	MGCP Grammar.
mgcp-n-03071-v15	3/11/04	Align NCS with MGCP Informational RFC 3435.