The SIP INFO method is specified in RFC 2976. This method is a means for transferring information during an ongoing session, that is, in the middle of a call. Examples of its use could include

- The transfer of *Dual-Tone Multifrequency* (DTMF) digits
- The transfer of account balance information
- The transfer of midcall signaling information generated in another network (such as the PSTN) and passed to the IP network via a gateway

Through the use of the INFO method, application-layer information could be transferred in the middle of a call, thereby affording user-agent servers, user-agent clients, or proxies the opportunity to act upon the information. For example, a prepaid subscriber could be informed in the middle of a call if the subscriber's prepaid account is nearing zero.

The SIP INFO method is a flexible tool and, as such, does not define the information that may or may not be sent using the INFO method. Typically, the INFO method would convey the user information within a message body, with the content and structure of the message body dependent upon the type of information to be conveyed. For example, an indication that the user's prepaid account balance is near zero could be conveyed in a message body in the form of text.

SIP Responses

The start line of a SIP response is a status line. This contains a status code, which is a three-digit number indicating the outcome of the request. The start line will also contain a reason phrase, which provides a textual description of the outcome. The reason code will be interpreted and acted upon by the client software, while the reason-phrase could be presented to the human user to aid in understanding the response. The syntax of the status line is

```
status-line = SIP version SP status code SP reason-phrase CRLF
```

Status codes defined in SIP have values between 100 and 699, with the first digit of the reason code indicating the class of response as follows. Thus, all status codes between 100 and 199 belong to the same class. The different classes are as follows:

- **1XX** Provisional (for example, 181 indicates that the call is being forwarded).

- **2XX** Success (only code 200 is defined, which means that the request has been understood and has been performed. In the case of an INVITE, a response of 200 is used to indicate that the called party has accepted the call).
- **3XX** Redirection (such as 302, which indicates that the called party is not available at the addresses used in the request and that the request should be reissued to a new address included with the response).
- **4XX** Request Failure (such as 401, which indicates that the client is not authorized to make the request).
- **5XX** Server Failure (such as 505, which indicates that the server does not support the SIP version specified in the request).
- **6XX** Global Failure (such as 604, indicating that the called user does not exist anywhere).

All responses, except for 1XX responses, are considered final and should be acknowledged with an ACK message if the original message happened to be an INVITE. SIP specifies that 1XX responses are provisional and do not need to be acknowledged. Table 5-1 gives a complete listing of the status codes defined in the latest revision of SIP (as of this writing), along with proposed reason phrases.

Table 5-1
SIP status codes

Category	Status code	Reason phrase
Provisional	100	Trying
	180	Ringing
	181	Call is being forwarded
	182	Queued
	183	Session in progress
Success	200	OK
Redirection	300	Multiple choices
	301	Moved permanently
	302	Moved temporarily
	305	Use proxy
	380	Alternative service
Request Failure	400	Bad request
	401	Unauthorized
	402	Payment required
	403	Forbidden
	404	Not found
	405	Method not allowed
	406	Not acceptable

Motorola - Exhibit 1022 - Page 193

Table 5-1 cont.
SIP status codes

Category	Status code	Reason phrase
	407	Proxy authentication required
	408	Request timeout
	410	Gone
	413	Request entity too large
	414	Request-URI too long
	415	Unsupported media type
	416	Unsupported URI scheme
	420	Bad extension
	421	Extension required
	423	Interval too brief
	480	Temporarily not available
	481	Call leg/transaction does not exist
	482	Loop detected
	483	Too many hops
	484	Address incomplete
	485	Ambiguous
	486	Busy here
	487	Request terminated
	488	Not acceptable here
	491	Request pending
	493	Undecipherable
Server Failure	500	Server internal error
	501	Not implemented
	502	Bad gateway
	503	Service unavailable
	504	Server timeout
	505	SIP version not supported
	513	Message too large
Global-Failure	600	Busy everywhere
	603	Decline
	604	Does not exist anywhere
	606	Not acceptable

SIP Addressing

As with any signaling protocol, requests and responses are sent to particular addresses. In SIP, these addresses are known as SIP *Uniform Resource Indicators* (URIs). These addresses take the form of user@host, which is very similar to an e-mail address. In many cases, a given user's SIP address can, in fact, be guessed from the user's e-mail address. Although the two may look similar, however, they are different. Whereas an e-mail address uses a mailto *Uniform Resource Locator* (URL), such as mailto:collins@home.net, a SIP URI has the syntax sip:collins@home.net.

SIP deals with multimedia sessions, which could include voice. Furthermore, SIP networks need to interwork with traditional circuit-switched networks. For these reasons, SIP enables the user portion of the SIP address to be a telephone number. Thus, we could have a SIP address such as sip:3344556789@telco.net. In a given network, such a SIP address could be used to cause media to be routed to a gateway that interfaces with the traditional telephone company. A SIP URI may also be supplemented by a number of parameters that provide more information. For example, to clearly indicate that a call is to a telephone number, the URI could be supplemented by the term *user*=phone. In such a case, the URI would have the format sip:3344556789@telco.net;user=phone.

Message Headers

SIP includes a number of different message headers. These headers are items of information included in a request or response to provide further information about the message or to enable appropriate handling of the message. In that respect, message headers are akin to the message parameters or information elements included in any typical signaling protocol such as ISUP, Q.931, and so on. For example, the To: header in an INVITE message indicates the called party, and the From: header indicates the calling party. The intention in this chapter is to describe the usage of headers and to give an understanding of the most common and useful headers. For complete definitions, the user is referred to RFC 3261.

Some headers make sense only in certain requests or responses, and in some cases, the presence of a given header will depend on the context. Moreover, the presence of a given header in a response might be reasonable only if the response is issued to a specific request. Tables 5-2 and 5-3 list the various headers and specify the application of each header to each request or response. Table 5-2 provides a mapping between requests and headers. The information in Table 5-2 should be read as follows:

- M means mandatory.
- M* means that the header field should be present in the request, but a receiver should be prepared to process the request even if the header is absent.
- O means optional.
- T means that the header should be included in the request if a stream-based transport (such as TCP) is used.

- C means conditional; the presence of the header depends on the context of the message.
- N/A means not applicable; the header should not be sent in the request.

Table 5-2

A mapping between requests and headers

Header	ACK	BYE	CAN	INV	OPT	REG
Accept	N/A	O	N/A	O	M*	O
Accept-Encoding	N/A	O	N/A	O	O	O
Accept-Language	N/A	O	N/A	O	O	O
Alert-Info	N/A	N/A	N/A	O	N/A	N/A
Allow	N/A	O	N/A	O	O	O
Authentication-Info	N/A	N/A	N/A	N/A	N/A	N/A
Authorization	O	O	O	O	O	O
Call-ID	M	M	M	M	M	M
Call-Info	N/A	N/A	N/A	O	O	O
Contact	O	N/A	N/A	O	O	O
Content-Disposition	O	O	N/A	O	O	O
Content-Encoding	O	O	N/A	O	O	O
Content-Language	O	O	N/A	O	O	O
Content-Length	T	T	T	T	T	T
Content-Type*	C	C	C	C	C	C
CSeq	M	M	M	M	M	M
Date	O	O	O	O	O	O
Error-Info	N/A	N/A	N/A	N/A	N/A	N/A
Expires	N/A	N/A	N/A	O	N/A	O
From	M	M	M	M	M	M
In-Reply-to	N/A	N/A	N/A	O	N/A	N/A
Max-Forwards	M	M	M	M	M	M
Min-Expires	N/A	N/A	N/A	N/A	N/A	N/A
MIME-Version	O	O	N/A	O	O	O
Organization	N/A	N/A	N/A	O	O	O

Table 5-2 cont.

A mapping between requests and headers

Header	ACK	BYE	CAN	INV	OPT	REG
Priority	N/A	N/A	N/A	O	N/A	N/A
Proxy-Authenticate	N/A	N/A	N/A	N/A	N/A	N/A
Proxy-Authorization	O	O	N/A	O	O	O
Proxy-Require	N/A	O	N/A	O	O	O
Record-Route	O	O	O	O	O	O
Reply-to	N/A	N/A	N/A	O	N/A	N/A
Require	N/A	C	N/A	C	C	C
Retry-After	N/A	N/A	N/A	N/A	N/A	N/A
Route	C	C	C	C	C	C
Server	N/A	N/A	N/A	N/A	N/A	N/A
Subject	N/A	N/A	N/A	O	N/A	N/A
Supported	N/A	O	O	M*	O	O
Timestamp	O	O	O	O	O	O
To	M	M	M	M	M	M
Unsupported	N/A	N/A	N/A	N/A	N/A	N/A
User-Agent	O	O	O	O	O	O
Via	M	M	M	M	M	M
Warning	N/A	N/A	N/A	N/A	N/A	N/A
WWW-Authenticate	N/A	N/A	N/A	N/A	N/A	N/A

*The Content-Type header field must be included if the message contains a message body. Otherwise, the header can be omitted.

Table 5-3 provides a mapping between headers and responses. It should be noted that the inclusion of a particular header in a response is dependent upon both the status code of the response and the request that led to the response. The Status Code column indicates the status codes for which a given header may be included in the response. In some cases, a given header may be used only with certain status codes. In other cases, a given header may be used with all status codes. The six columns corresponding to the six request methods of RFC 3261 indicate whether a given header may be used in a response to that particular type of request. For example, the Allow header can be used with a 200 or a 405 status code; however, it can

only be used with a 200 response if the response is sent as a result of an OPTIONS request.

The information in Table 5-3 should be read in the same way as the information in Table 5-2, with the addition that the indication (c) means that the value of a header is copied from the request to the response. A value of N/A in the Status Code column means that the header should not be included in any response.

Table 5-3

A mapping between headers and responses

Header	Status Code	ACK	BYE	CAN	INV	OPT	REG
Accept	2XX	N/A	N/A	N/A	O	M*	O
Accept	415	N/A	C	N/A	C	C	C
Accept-Encoding	2XX	N/A	N/A	N/A	O	M*	O
Accept-Encoding	415	N/A	C	N/A	C	C	C
Accept-Language	2XX	N/A	N/A	N/A	O	M*	O
Accept-Language	415	N/A	C	N/A	C	C	C
Alert-Info	180	N/A	N/A	N/A	O	N/A	N/A
Allow	2XX	N/A	O	N/A	M*	M*	O
Allow	405	N/A	M	N/A	M	M	M
Allow	All except 2XX, 415	N/A	O	N/A	O	O	O
Authentication-Info	2XX	N/A	O	N/A	O	O	O
Authorization	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Call-ID	All (c)	M	M	M	M	M	M
Call-Info	All	N/A	N/A	N/A	O	O	O
Contact	1XX	N/A	N/A	N/A	O	N/A	N/A
Contact	2XX	N/A	N/A	N/A	M	O	O
Contact	3XX, 485	N/A	O	N/A	O	O	O
Content-Disposition	All	O	O	N/A	O	O	O
Content-Encoding	All	O	O	N/A	O	O	O
Content-Language	All	O	O	N/A	O	O	O
Content-Length	All	T	T	T	T	T	T
Content-Type*	All	C	C	N/A	C	C	C
CSeq	All (c)	M	M	M	M	M	M

Motorola - Exhibit 1022 - Page 198

Table 5-3 cont.

A mapping between headers and responses

Header	Status Code	ACK	BYE	CAN	INV	OPT	REG
Date	All	O	O	O	O	O	O
Error-Info	300–699	N/A	O	O	O	O	O
Expires	All	N/A	N/A	N/A	O	N/A	O
From	All (c)	M	M	M	M	M	M
In-Reply-to	N/A	N/A	N/A	N/A	O	N/A	N/A
Max-Forwards	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Min-Expires	423	N/A	N/A	N/A	N/A	N/A	M
MIME-Version	All	O	O	N/A	O	O	O
Organization	All	N/A	N/A	N/A	O	O	O
Priority	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Proxy-Authenticate	401	N/A	O	O	O	O	O
Proxy-Authenticate	407	N/A	M	N/A	M	M	M
Proxy-Authorization	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Proxy-Require	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Record-Route	2XX, 18X	N/A	O	O	O	O	N/A
Require	All	N/A	C	N/A	C	C	C
Retry-After	404, 413, 480, 486, 500, 503, 600, 603	N/A	O	O	O	O	O
Route	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Server	All	N/A	O	O	O	O	O
Subject	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Supported	2XX	N/A	O	O	M*	M*	O
Timestamp	All	O	O	O	O	O	O
To	All (c)	M	M	M	M	M	M
Unsupported	420	N/A	M	N/A	M	M	M
User-Agent	All	O	O	O	O	O	O
Via	All (c)	M	M	M	M	M	M
Warning	All	N/A	O	O	O	O	O
WWW-Authenticate	401	N/A	M	N/A	M	M	M
WWW-Authenticate	407	N/A	O	N/A	O	O	O

*The Content-Type header field must be included if the response contains a message body. Otherwise, the header can be omitted.

Motorola - Exhibit 1022 - Page 199

General Headers Some headers can be used in both requests and responses, and they are known as *general headers*. Such headers contain basic information needed for the handling of requests and responses. Examples include the To: header field, which indicates the recipient of the request, the From: header field, which indicates the originator of the request and the Call-ID: header field, which uniquely identifies a specific invitation to a session.

One of the most useful general headers is the Contact: header, which provides a URI for use in future communication regarding a particular session. In a SIP INVITE, the Contact: header might be different than the From: header. This means, for example, that the initiator of a SIP session need not be a participant in the session. An example of such usage would be a case where a multiparty session is organized and initiated by an administrator who does not participate in the session itself. Used in responses, the Contact: header is useful for directing further requests (such as ACK) directly to the called user when the original request passed through one or more proxies. This header can also be used to indicate a more appropriate address if an INVITE issued to a given URI failed to reach the user. An example of such usage would be with a 302 response (moved temporarily) where the Contact: header in the 302 response gives the current URI of the user.

Request Headers Request headers apply only to SIP requests. They are used to provide additional information to the server regarding the request itself or regarding the client. Examples include the Subject: header field, which can be used to provide a textual description of the topic of the session, and the Priority: header field, which is used to indicate the urgency of the request (emergency, urgent, normal, or nonurgent).

Response Headers Response header fields apply only to response (status) messages. These header fields are used to provide further information about the response that cannot be included in the status line. Examples of response header fields include the Unsupported: header field used to identify those features not supported by the server and the Retry-After header field, which can indicate when a called user will be available if the user is currently busy or unavailable.

Entity Headers In SIP, the message body is used to contain information about the session or information to be presented to the user. In the case of information regarding a session, the session description is most often specified according to SDP, in order to indicate information such as the RTP

payload type as well as an address and port to which media should be sent. The purpose of the entity headers is to indicate the type and format of information included in the message body, so that the appropriate application can be called upon to act on the information within the message body. The entity header fields are Content-Length, Content-Type, Content-Encoding, Content-Disposition, and Content-Language.

The Content-Length header field specifies the length of the message body in octets. The Content-Type header field indicates the media type of the message body. For VoIP, this header will usually indicate SDP, in which case the header field will appear as Content-Type: application/sdp.

The Content-Encoding header field is used to indicate any additional codings that have been applied to the message body and hence which decoding actions need to be taken by the recipient in order to obtain the media type indicated by the Content-Type header field. For example, it is perfectly legal to compress the content of the message body. In such a case, the Content-Encoding header field would be used to pass information related to the compression scheme used.

The Content-Disposition header field describes how the message body or, for multipart messages, a message body part should be interpreted. If the message body, or part of a message body, describes the characteristics of a session, then the associated Content-Disposition field would have the value *session*. Non-session data can be carried in a message body, however. For example, a message body might contain an image of the caller as part of an advanced Caller-ID service. In such a case, the Content-Disposition would have the value *icon*. The value *render* is used to indicate that the message body part should be displayed to the user. This value would apply, for example, if the sender included some text that he or she wanted to present to the other party. The value *alert* indicates that the message body part contains information such as an audio clip that would alert the user to the receipt of a request. For example, a given SIP user could choose his or her own personal ring tone that is played to a called party, so someone could tell who is calling based on their personal ring tone.

Examples of SIP Message Sequences

Building upon the information provided already in this chapter, a number of examples are provided to demonstrate how SIP messages might appear

in various situations. We do not explain every possible option in these examples, just some of the most common occurrences. For details of all options available, the reader is referred to RFC 3261.

Registration

The very first request issued by a client is likely to be REGISTER, because this is the request that provides the server with an address at which the user can be reached for SIP sessions. The REGISTER request is somewhat similar to the Registration Request between a terminal and a gatekeeper in H.323.

Figure 5-8 shows an example registration scenario. In this scenario, Collins has logged in to host station1.work.com. This causes a REGISTER request to be sent to the local registrar. The Via: header field contains the path taken by the request so far, which requires that the originating client insert its own address in this field. Note the format of the Via: header and in particular the fact that it specifies the transport being used. The default is the *User Datagram Protocol* (UDP). Note also that the Via: header

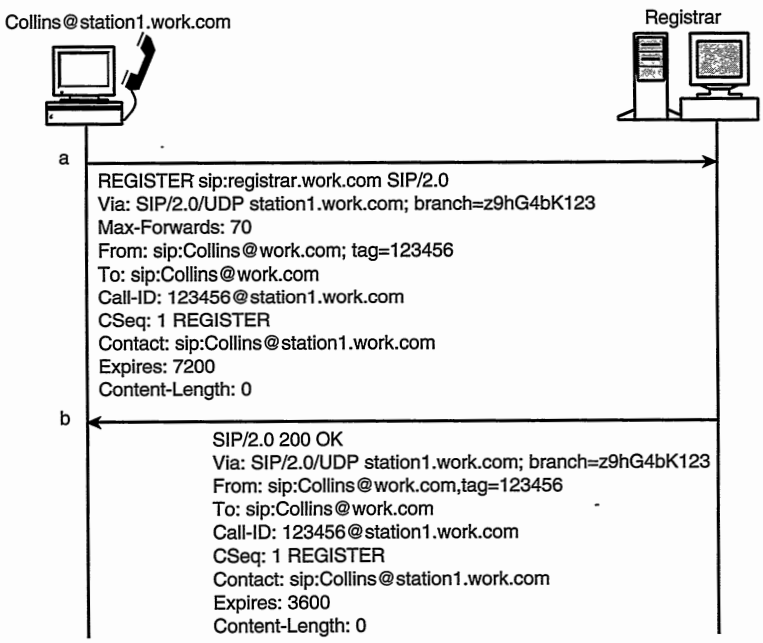


Figure 5-8
SIP registration

contains a *branch parameter*. The purpose of the branch parameter is discussed later when we describe the operation of proxy servers.

The From: header field indicates the address of the individual who has initiated the registration. The To: header field indicates the "address of record" of the user being registered and is the address that the registrar will store for that user. In other words, this is the address at which the user wants to be contactable. The From: and To: fields will be identical if a user is registering himself. The two fields need not be identical, however, which means that one individual can perform a registration on behalf of another. Note that the To: header field is not used to contain the address of the registrar. That address is indicated in the first line of the request. Note that the From: field in our example contains a tag parameter, which is a pseudorandom value generated at the sender of the request and is used for identification purposes.

The Call-ID: header field is set by the originating client. All REGISTER requests for an individual client should use the same value of Call-ID. In order to avoid the possibility that different clients might choose the same Call-ID, the recommended syntax for the Call-ID is local-id@host, thereby making the Call-ID host specific.

In Figure 5-8, Collins has inserted a Contact: header field that indicates that future SIP messages should be routed to sip:collins@station1 .work.com. In other words, messages that are addressed to sip:Collins @work.com should be forwarded to sip:Collins@station1.work.com.

The REGISTER request does not contain a message body, since the message is not used to describe a session of any kind. Therefore, the Content-Length: field is set to 0.

The response to the REGISTER request is positive, as indicted by the 200 (OK) in the response line. Note that the content of the Via: header field is copied from request to response. The content of the CSeq: (or command sequence) header field is also copied from request to response. The CSeq: header field indicates the method used in the request and an integer number. Consecutive requests for the same Call-ID must use contiguous increasing integer numbers. Although not as critical for REGISTER requests, the use of the CSeq is very important in other requests where different requests can be issued for the same Call-ID, in which case the CSeq is used to avoid ambiguity. A response to any request must use the same value of CSeq as used in the request itself.

Through the use of the Expires: header, Collins has requested that the registration be effective for 2 hours. The registrar has chosen to override this aspect of the request and has limited the duration of the request to 1 hour. A registrar can change the length of time for which a given registra-

tion is effective, but if a registrar chooses to do so, it will normally set a lower registration interval than that requested. It will not set a higher interval. The Expires: header may be specified as a duration in seconds or as a specific data and time. If it is specified as a number of seconds, the maximum value is such that a registration may be active for up to approximately 136 years, which should be long enough for anyone. If a REGISTER request contains an Expires: header with a value that is too short (the registration is to be active for too short a time), then the REGISTER request will be rejected with the status code 423 (Interval too brief). The response will include the Min-Expires: header field, which specifies the minimum registration interval that the registrar will accept.

Having registered as shown in Figure 5-8, Collins could subsequently register at another terminal. In such a case, both registrations would become active and subsequent invitations destined for Collins would be routed to both terminals. Of course, before registering at the second location, Collins could cancel the existing registration. This would be done by sending another REGISTER request for the same address of record and Contact and specifying a registration interval of 0. In this case, the REGISTER request would be identical to the first REGISTER request, with the exceptions that the CSeq integer is incremented and the Expires header field contains the value 0. If Collins wanted to cancel all existing registrations, then he would send a REGISTER message with the Expires header field set to 0 and the Contact header field populated with the wildcard character *. The Expires header value of 0 indicates the cancellation of a Registration and the Contact: header value of * indicates that the request is applicable to all contact information for Collins.

Invitation

The INVITE request is the most fundamental and important SIP request, as it is the request used to initiate a session (i.e., establish a call). Figure 5-9 shows a two-party call, where Collins initiates the session. The INVITE request is issued to manager@station2.work.com, as can be seen from the Request-URI. The reason why the address is very specific to the host where the callee is located is because this example's signaling flow does not traverse any proxies. Under normal circumstances, one might expect that the message would be addressed to sip:manager@work.com and would be forwarded by a proxy to sip:manager@station2.work.com. Since our example does not include a proxy to forward the INVITE to the specific host, the INVITE itself is addressed to the specific host. In our example, the To:

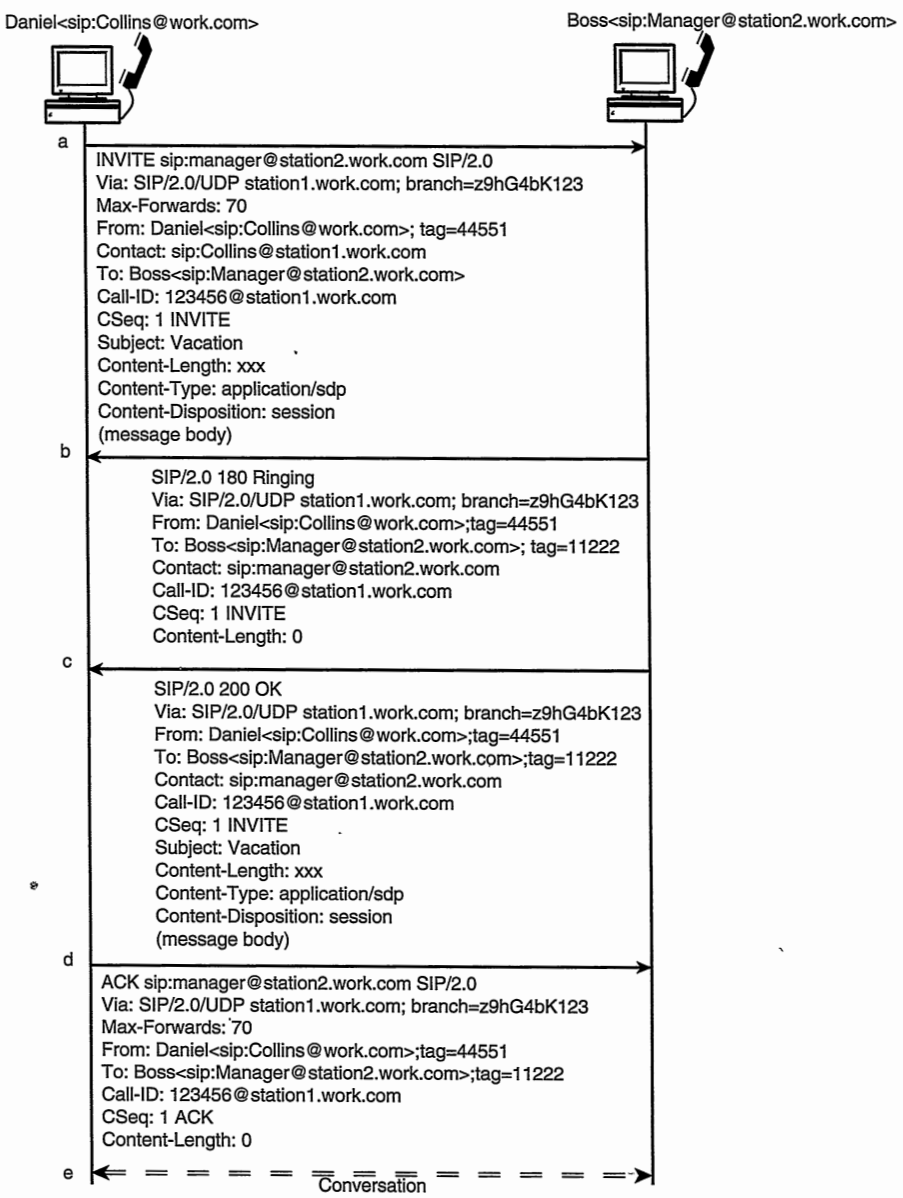


Figure 5-9
SIP establishment of a two-party call

header field has the same value as the Request-URI, because we are not traversing any proxies. (As described later in this chapter, the Request-URI can be changed when a message passes through a proxy.)

The From: header field indicates that the call is from collins@work.com. As can be seen from Figure 5-9, SIP enables a display name to be used with

the SIP URI. Thus, the called user's terminal could display the name Daniel when alerting the called user rather than displaying the SIP URI. The optional header field Subject: is used to indicate the nature of the call, and the type of media that Daniel wishes to use is described within the message body. The Content-Type: entity header field indicates that this particular message body is described according to SDP. The Content-Disposition header field indicates that the message body is to be treated as session information (as opposed to being displayed to the user, for example). In the example, the length of xxx is used just to indicate some nonzero value. The actual length will depend upon the content of the message body.

In Figure 5-9, the first response received is an indication that the user is being alerted. This is conveyed through the use of the 180 status code. Note that most of the header fields are copied from the request to the response, with the exception of the Content-Length and Content-Type, since this response does not contain a message body.

Note that the To: header field from the called party contains a tag parameter, whereas the To: field in the original invite did not contain a tag parameter. In fact, the tag parameter should be inserted only by the party that "owns" the address in question. In combination, the content of the tag in the From: field, the tag in the To: field, and the Call-ID constitute a dialog ID, which identifies a peer-to-peer relationship between two user agents. Once both parties have inserted their tag values into the To: and From: fields according to which of the fields they "own," then a dialog has been established. From that point onwards, each party should include the appropriate tag value in the To: and From: fields of every subsequent request and response until the dialog is terminated.

An important aspect of the dialog is the dialog state. If the response to an INVITE is a 1XX (provisional) response and it contains a tag parameter in the To: header field, then a session has not yet been established, but a dialog has. Such a dialog is said to be in *early* state. If a 2XX response is received with a tag in the To: header field, then the dialog is *confirmed*.

Subsequently in our example, the called user answers and a 200 (OK) response is returned. This response to an INVITE includes a number of header fields that have been copied from the original request. In addition, it contains a message body describing the media that the called party wants to use.

Finally, the caller sends an ACK to confirm receipt of the response. Note that the content of the CSeq: header field has changed to reflect the new request. Note also that, although most requests are answered with a 200 (OK) response if the request has been received and handled correctly, this is not the case for an ACK request. Once the ACK has been sent, the parties can exchange media.

Motorola - Exhibit 1022 - Page 206

Note that the requests shown in Figure 5-9 all include the Max-Forwards header field. This header should be included in all requests to ensure that a given request does not traverse too many proxies. The value of Max-Forwards is decremented at each proxy in the chain between requester and responder. Seventy is the recommended initial value for Max-Forwards. We discuss proxy servers later in this chapter.

Termination of a Call

A user agent that wants to terminate a call does so through the issuance of a BYE request. Any party to a call can issue this request. Upon receipt of a BYE request, the recipient of the request should immediately stop transmitting all media directed at the party who has issued the BYE request.

Figure 5-10 shows a call termination corresponding to the call origination shown in Figure 5-9. As can be seen, the BYE request from Daniel has many of the same header values as in the original INVITE request, with the important exception that the CSeq has been changed to reflect the BYE request, as opposed to the INVITE request. As with any successfully completed request (besides ACK), a 200 response is returned.

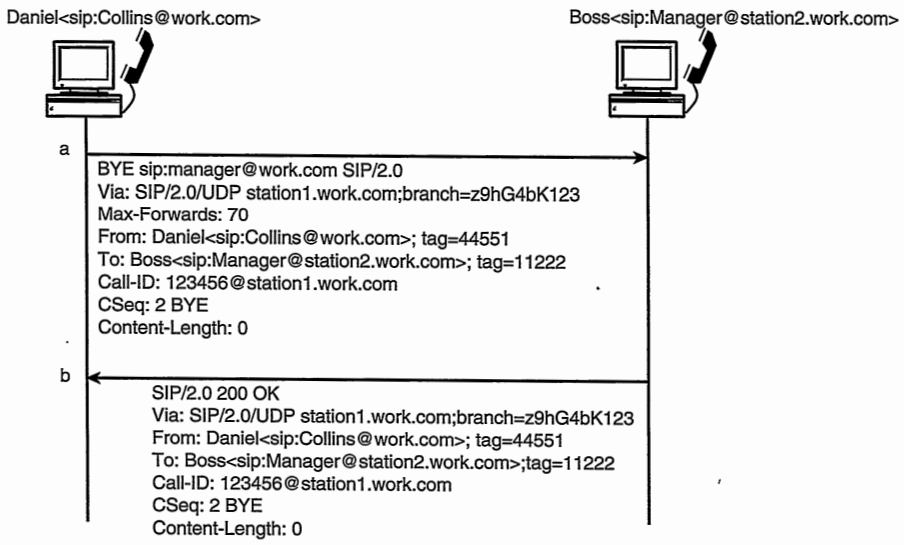


Figure 5-10
Call termination

the SIP URI. Thus, the called user's terminal could display the name Daniel when alerting the called user rather than displaying the SIP URI. The optional header field Subject: is used to indicate the nature of the call, and the type of media that Daniel wishes to use is described within the message body. The Content-Type: entity header field indicates that this particular message body is described according to SDP. The Content-Disposition header field indicates that the message body is to be treated as session information (as opposed to being displayed to the user, for example). In the example, the length of xxx is used just to indicate some nonzero value. The actual length will depend upon the content of the message body.

In Figure 5-9, the first response received is an indication that the user is being alerted. This is conveyed through the use of the 180 status code. Note that most of the header fields are copied from the request to the response, with the exception of the Content-Length and Content-Type, since this response does not contain a message body.

Note that the To: header field from the called party contains a tag parameter, whereas the To: field in the original invite did not contain a tag parameter. In fact, the tag parameter should be inserted only by the party that "owns" the address in question. In combination, the content of the tag in the From: field, the tag in the To: field, and the Call-ID constitute a dialog ID, which identifies a peer-to-peer relationship between two user agents. Once both parties have inserted their tag values into the To: and From: fields according to which of the fields they "own," then a dialog has been established. From that point onwards, each party should include the appropriate tag value in the To: and From: fields of every subsequent request and response until the dialog is terminated.

An important aspect of the dialog is the dialog state. If the response to an INVITE is a 1XX (provisional) response and it contains a tag parameter in the To: header field, then a session has not yet been established, but a dialog has. Such a dialog is said to be in *early* state. If a 2XX response is received with a tag in the To: header field, then the dialog is *confirmed*.

Subsequently in our example, the called user answers and a 200 (OK) response is returned. This response to an INVITE includes a number of header fields that have been copied from the original request. In addition, it contains a message body describing the media that the called party wants to use.

Finally, the caller sends an ACK to confirm receipt of the response. Note that the content of the CSeq: header field has changed to reflect the new request. Note also that, although most requests are answered with a 200 (OK) response if the request has been received and handled correctly, this is not the case for an ACK request. Once the ACK has been sent, the parties can exchange media.

Note that the requests shown in Figure 5-9 all include the Max-Forwards header field. This header should be included in all requests to ensure that a given request does not traverse too many proxies. The value of Max-Forwards is decremented at each proxy in the chain between requester and responder. Seventy is the recommended initial value for Max-Forwards. We discuss proxy servers later in this chapter.

Termination of a Call

A user agent that wants to terminate a call does so through the issuance of a BYE request. Any party to a call can issue this request. Upon receipt of a BYE request, the recipient of the request should immediately stop transmitting all media directed at the party who has issued the BYE request.

Figure 5-10 shows a call termination corresponding to the call origination shown in Figure 5-9. As can be seen, the BYE request from Daniel has many of the same header values as in the original INVITE request, with the important exception that the CSeq has been changed to reflect the BYE request, as opposed to the INVITE request. As with any successfully completed request (besides ACK), a 200 response is returned.

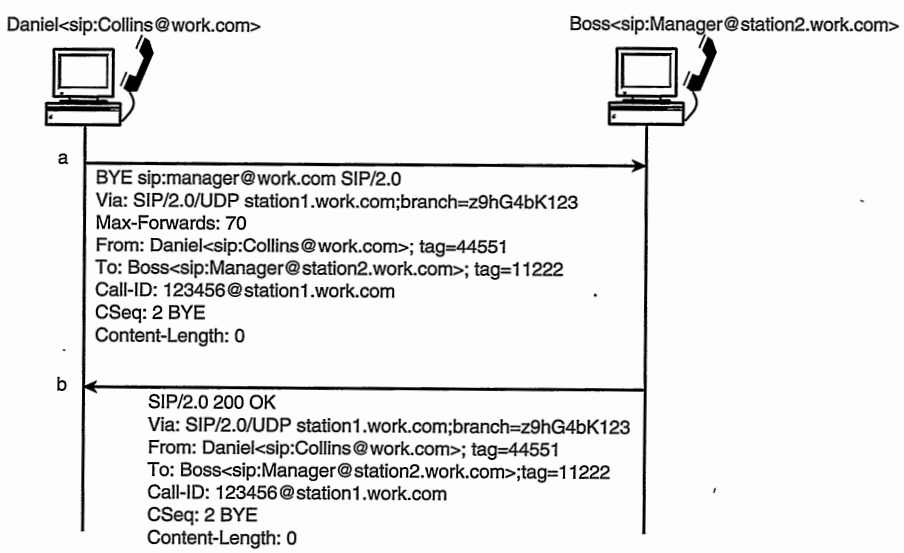


Figure 5-10
Call termination

Redirect and Proxy Servers

The examples shown so far have involved one user agent communicating directly with another. Although that is certainly possible, a more likely scenario will involve the use of one or more proxy or redirect servers.

Redirect Servers

A redirect server normally responds to a request with an alternative address to which the request should be directed. The exceptions are cases where the server has received a request that cannot be supported, in which case it will return a 4XX, 5XX, or 6XX response, or when the server receives a CANCEL request, to which it should respond with a 200 response.

Consider again the example call from Daniel to Boss as depicted in Figure 5-9. Instead of the call-completion scenario depicted in Figure 5-9, let us assume that Boss is out of the office and has registered at a different location. Using a redirect server, the call scenario would be as shown in Figure 5-11.

The initial INVITE is sent to Boss at the redirect server. In this case, the Request-URI and the To: address do not include the specific host where Boss has logged in. That is likely to be the most common situation. After all, a caller is unlikely to know exactly where a given user has logged in.

The server responds with the status code 302 (moved temporarily) and within the response includes the Contact: header field. The content of this field is the address that the caller should try as an alternative.

The client then creates a new INVITE. This new INVITE uses the address received from the redirect server as the Request-URI while the To: header remains the same. The call-ID also remains the same, but the CSeq number is incremented. This number is always incremented when a second request of the same type is made for a given call. In this case, this is the second INVITE for the call. The fact that the To: header is the same allows Boss (at home) to recognize that the call was originally made to his or her address at work. Based on this information (plus the identity of the caller and the subject of the call), Boss might or might not decide to accept the call. This is another example of how SIP provides information to enable a user to make intelligent call-handling decisions.

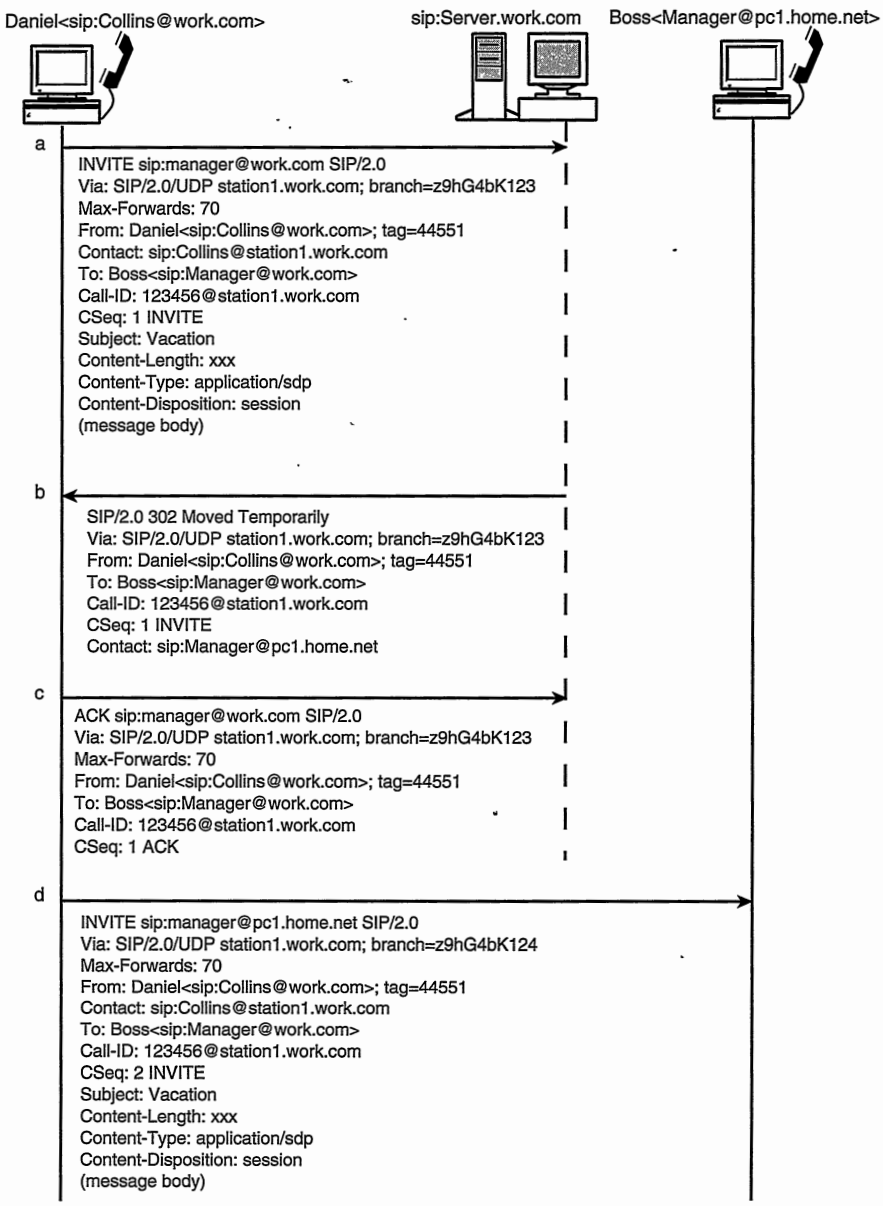


Figure 5-11 Application of a redirect server

Proxy Servers

A proxy server sits between a user-agent client and the far-end user-agent server. A proxy accepts requests from the client and forwards them

onwards, perhaps after some translation. Numerous proxies can reside in a chain between the caller and callee. The most common scenario will have at least two proxies: one at the caller end and one at the callee end.

Let's assume that Boss (at home) calls Collins (at work). If we assume that the call will pass via a proxy, as might be expected, then the sequence of messages could appear as in Figure 5-12. To avoid clutter, the Subject and Message Body headers have been omitted. Those headers would normally be included.

The proxy receives the INVITE and forwards it onwards. In doing so, the proxy can change the Request-URI. The proxy will do so if it knows that the Request-URI should be mapped to a different address. For example, if a proxy is also a registrar and is aware of a Contact address for a given address of record, then the proxy will change the Request-URI of an INVITE to the appropriate contact address. In the example of Figure 5-12, the proxy is aware that the address sip:Collins@work.com should be mapped to sip:Collins@station1.work.com, and it uses sip:Collins@station1.work.com as the Request-URI when it forwards the request. In many cases, however, there might be several proxies in the chain from caller to callee. It is likely that only the last proxy in the chain is aware of where a user has logged in, and it is only that proxy that would map the Request-URI to a different value. The other proxies in the chain would simply use the domain part of the received Request-URI as input to a location function (such as a *Domain Name System* [DNS]) to determine the next hop in the path from caller to callee, but they would not change the value of the Request-URI.

In Figure 5-12, the proxy returns a provisional response of 100 (trying) to the initial INVITE, and the proxy does this in parallel with forwarding the INVITE towards the called party. The requirement to issue the provisional 100 response is mandatory in some cases and optional in others. In general, it is a good idea to provide the provisional response in all cases.

Of major importance for proxy servers is the Via: header field. The Via: header is used to indicate the path taken by a request so far. When a request is generated, the originating client inserts its own address in a Via: header field. Each proxy along the way also inserts its address in a new Via: header field, placed in front of any existing Via: headers, as can be seen in Figure 5-12. Therefore, the collection of Via: headers provides a map of the path taken through the network by a given request.

When a proxy receives a request, it checks to see if its own address is already included in one of the Via: header fields. If so, then the request has already passed through the proxy and a loop condition might exist. A loop condition will exist if those header fields that affect the routing of the request (most notably the Request-URI) are the same as a request that has

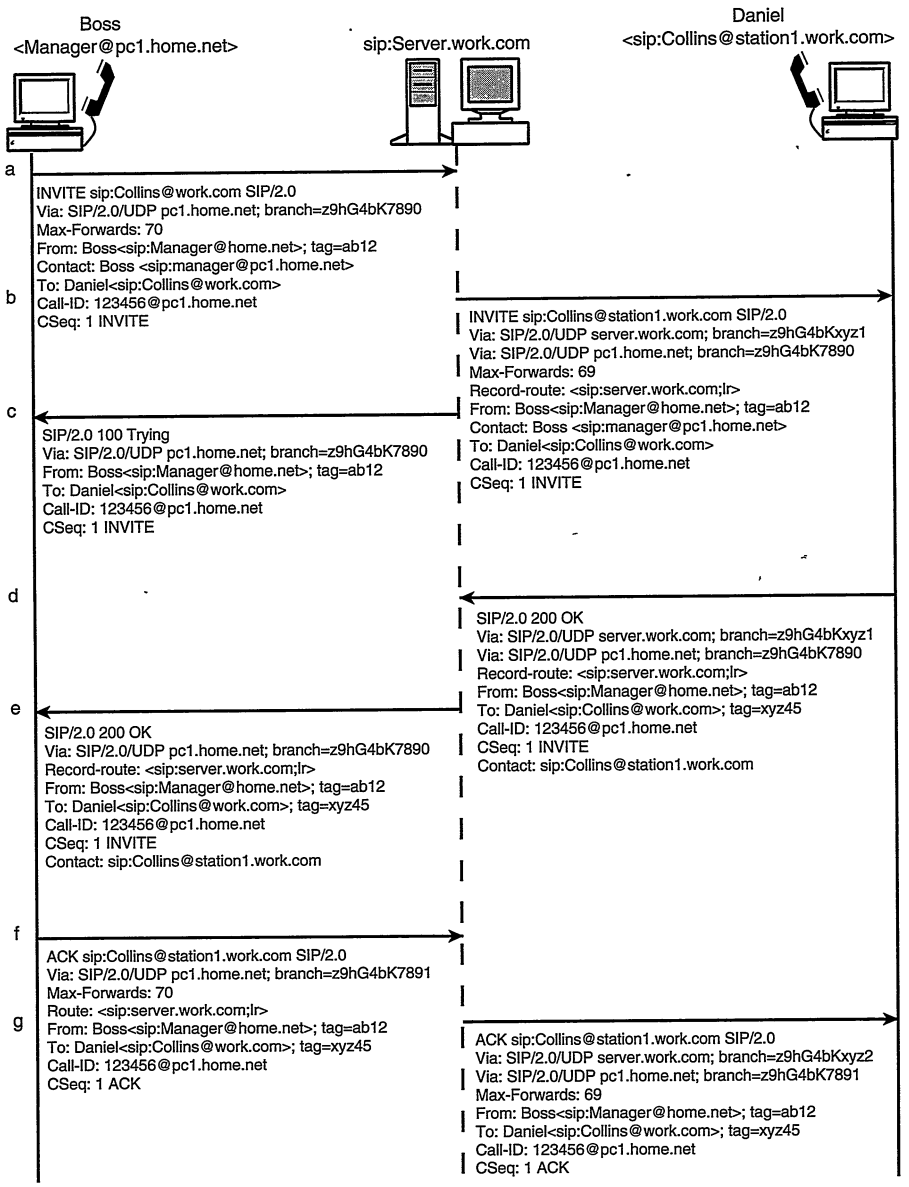


Figure 5-12
Application of a proxy server

already been processed at the proxy. In other words, the fact that a proxy has received a request with a Via: header field indicating the proxy itself does not necessarily mean that a loop condition does exist, just that it *might* exist. Examination of the REQUEST-URI and the other header fields is needed to be certain.

For example, assume a request had been processed at a proxy and that request led to a certain routing that ultimately led back to the proxy. A loop condition would exist here only if the proxy would forward the request the same way both the first time and the second time. On the other hand, if the request had been modified such that the proxy would route the request differently the second time it arrives, then we have a spiral condition and not a loop condition. A spiral condition is not an error. In the event that a loop exists, the proxy can respond to the request with status code 482 (loop detected). Otherwise, the proxy will pass the request onwards after inserting its own address in a new Via: header field.

Responses also include Via: header fields, which are used to send a response back through the network along the same path that the request used (in reverse, of course). When a server responds to a request, the response contains the list of Via: headers exactly as received in the request. When a proxy receives a response, the first Via: header should refer to itself. If it does not, a problem has occurred and the message is discarded. Assuming that the first Via: header field indicates the proxy itself, then the proxy removes the header and checks to see if a second Via: header exists. If not, then the message is destined for the proxy itself. If there is a second Via: header field, then the proxy passes the response to the address in that field. In this way, the response finds its way back to the originator of the request along the path that was originally taken by the request.

Note that the Via: header in the example of Figure 5-12 contains a branch parameter. In the original RFC 2543, the primary objective of the branch parameter was to distinguish between multiple responses to the same request. As we shall describe shortly in a discussion regarding *forking*, a proxy might forward a single request to multiple destinations. In such an event, multiple responses might be received and those need to be distinguished.

In the latest SIP specification (as of this writing), greater significance is applied to the branch parameter. Specifically, the branch parameter is used to uniquely identify a transaction created at a given entity (such as a proxy) and assists in the recognition of loop conditions. Thus, the branch parameter must be unique. Moreover, the latest SIP specification requires that the branch parameter begin with the characters z9hG4bK, which is known as the *magic cookie*. This strange set of characters at the beginning of the branch parameter indicates that a client or server complies with the latest SIP specification, rather than the base RFC 2543. In the base RFC 2543 specification, the branch parameter is optional and does not uniquely identify a transaction. Consequently, if a proxy receives a request with a Via: header and the branch parameter, it will only assume that the branch

parameter is globally unique if it begins with the magic cookie. The chances of a system that complies only with RFC 2543 selecting a branch parameter that begins with the magic cookie are miniscule.

The branch parameter is unique for every request issued by a client, with the exception of CANCEL and ACK for non-2XX responses. The branch parameter for a CANCEL or ACK for a non-2XX response should be the same as the branch parameters for the original request (the request being cancelled, or the request for which the non-2XX response was received).

Proxy State A proxy can be either stateless or stateful. If stateless, then the proxy takes an incoming request, performs whatever translation is necessary, forwards the corresponding outgoing request, and forgets that anything ever happened. If a message is retransmitted through a stateless proxy, then it must be forwarded in exactly the same manner as the original request. This action requires that the choice of Via: header (specifically the tag value) be computed based on header values in a request that do not change on retransmission.

If a proxy is stateful, then the proxy remembers incoming requests and corresponding outgoing requests, and it is able to act more intelligently on subsequent requests and responses related to the same session.

Two important headers in Figure 5-12 are the Record-Route: and the Route: headers. In the example, all messages and responses pass through the same proxy. This need not necessarily be the case, however. Although the response to a given request must return along the same route as used by the request, two consecutive requests need not follow the same path. Given that an initial INVITE will contain a Contact: header and responses can also contain a Contact: header, the two ends of the call might have the addressing information necessary to communicate directly. Therefore, after the initial INVITE request and response, subsequent requests and responses might be sent end to end. A proxy might require that it remain in the signaling path for all subsequent requests, however. For example, a proxy might need to remain in the signaling path in order to provide some advanced service. A proxy can ensure that it remains in the signaling path through the use of the Record-Route: header.

Each proxy that wants to remain in the signaling path for future requests inserts its address in the Record-Route: header field of the INVITE, such that the Record-Route: header field ultimately contains a list of proxy addresses, each of the form server.domain. Each proxy places its address at the head of the list such that, at the end of the signaling path, the first entry in the list is that of the last proxy used.

A 200 response to the INVITE will include the Record-Route: header as it was received by the ultimate destination of the INVITE. Hence, this response contains an ordered list of all the proxies used in the INVITE. The content of the header is propagated back through the network along with the 200 response. When the 200 response is received at the client that originated the INVITE, the information contained in the Record-Route: header is used in subsequent requests (such as ACK or BYE) related to the same call. The client takes the information in the Record-Route: header and places it in a Route: header in reverse order so that the Route: header contains a list of proxies in the direction from calling client to called server. The client uses the first entry in the Route: header as the destination to send the next request (such as ACK or BYE). The proxy that receives that request removes the first entry in the Route: header field (which will indicate the proxy itself) and forwards the message to the next proxy in the chain, or to the user-agent server if the proxy in question is the last proxy in the chain. The result is that each request follows the same path as the original INVITE, thereby allowing each proxy in the chain to be aware of each request and response.

As can be seen from Figure 5-12, the Record-Route: and Route: header fields contain the *loose routing* (lr) parameter. Systems compliant with the latest SIP specification should insert this parameter in the Record-Route: header. The initial SIP specification did not require the presence of the lr parameter, and its absence indicates strict routing. The difference between the loose and strict routing is in how the Route: header is populated and in the value that is placed in the Request-URI.

In Figure 5-12, lr is depicted. A client or proxy that supports loose routing uses the content of the Route: header field as the address to which a message is forwarded, but uses the ultimate destination (as specified in a Contact: header) as the Request-URI. In Figure 5-12, we can see that Boss sends an ACK message to the proxy, but uses Collins@station1.work.com as the Request-URI (not the SIP address of the proxy). Moreover, the ACK from Boss contains the Route: header value of sip:server.work.com. If strict routing were used, then the ACK from Boss would have a Request-URI of sip:server.work.com, and the Route: header would not include the proxy server's SIP address; rather, it would contain Daniel's Contact: header information (sip:Collins@station1.work.com).

Forking Proxy A proxy can "fork" requests. This would happen if a particular user is registered at several locations. An incoming INVITE for the user would be sent from the proxy to each of the registered locations. If one

of the locations answered, then the proxy could issue a CANCEL to the other locations so that they do not continue "ringing" when it is known that the called user has answered elsewhere. In order to handle such forking, a proxy must be stateful.

Figure 5-13 gives an example of a forked request. For the sake of clarity, the Subject: and message body header fields have been omitted. In this scenario, Boss (at home) calls Collins (at work). However, Collins happens to be logged in to two different work stations: pc1.work.com and pc2.work.com. When the proxy receives the INVITE, it immediately returns a 100 (trying) response. The proxy then forwards the INVITE request to the two locations in parallel. Not only does the proxy add a Via: header to each outgoing INVITE, but it ensures that the branch parameter is different in each of the outgoing INVITE requests, which enables the proxy to distinguish between the responses to those requests.

Collins answers the incoming call at pc2.work.com. Therefore, a 200 OK response is returned from pc2.work.com. Upon receiving the response, the proxy immediately issues a CANCEL to pc1.work.com so that any resources allocated by that machine may be released. Note that the CANCEL contains only a single Via: header, that of the server. Since a CANCEL request is always sent hop by hop, a Via: header identifying the caller is not included in the CANCEL request.

The response from pc2.work.com is then relayed back to the originator of the INVITE request. The remainder of the call (ACK, BYE, and so on) will proceed as normal. Meanwhile, however, pc1.work.com still owes the proxy server two responses: one for the initial INVITE and one for the CANCEL. Upon receipt of the CANCEL request, pc1.work.com responds to the INVITE with the status code 487 (request cancelled). It will also respond to the CANCEL request with 200 OK. These two responses are not shown in Figure 5-13.

The Session Description Protocol (SDP)

The examples so far have focused on various SIP requests, responses, and header fields. Although some mention has been made of the message body and entity header fields, the message body has not been described in any detail. Although a SIP message body can carry many different types of information (such as an image or text to display), the most common mes-

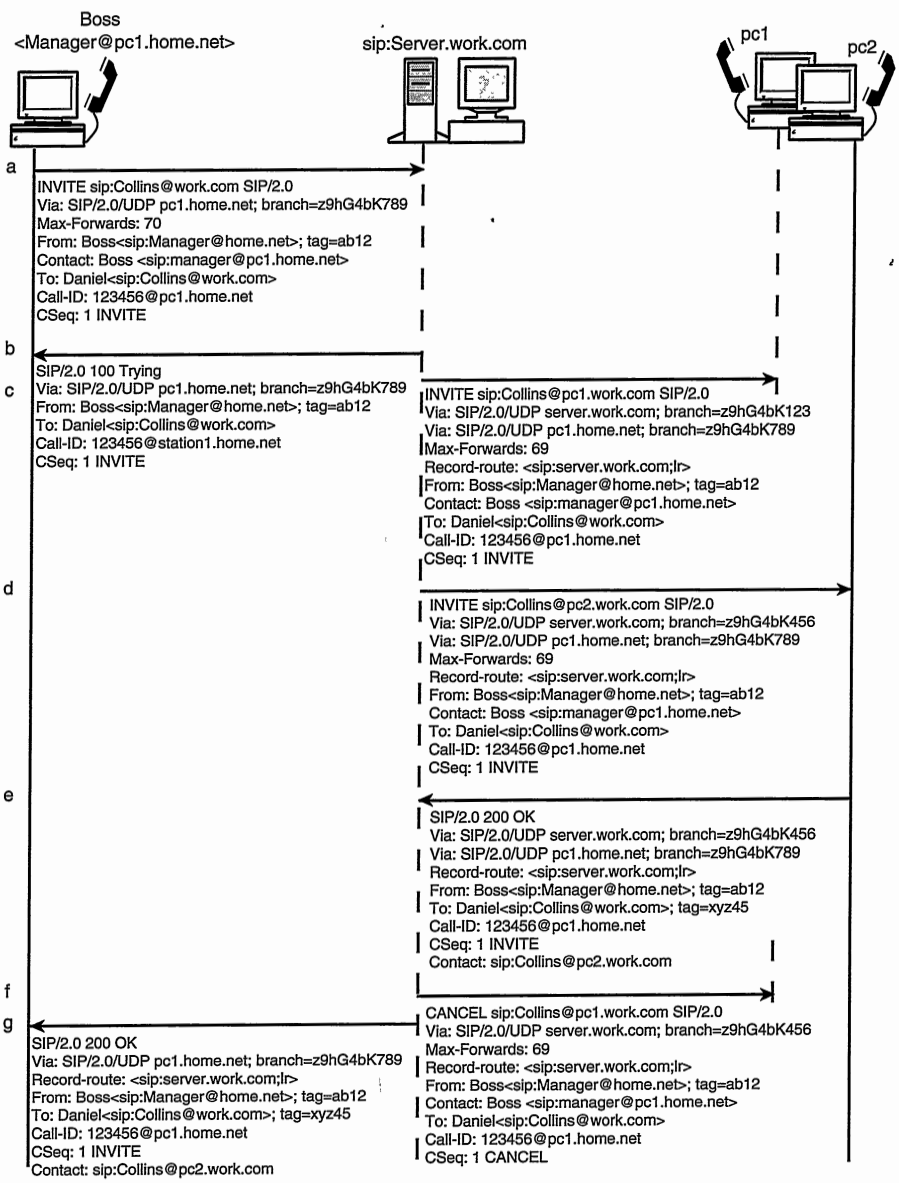


Figure 5-13
A forking proxy

sage body will be session information describing the media to be exchanged between the parties. The session description will include media information such as the RTP payload type, addresses, and ports. The format of that description will normally be according to SDP.

SIP uses SDP in an answer/offer mode. A caller sends an INVITE with an SDP description that describes the set of media formats, addresses, and ports that the caller is willing to use. This set of media formats comprises an offer by the caller. The called party responds with an SDP description that aligns with the offered SDP description, but which includes an acceptance or rejection of each of the offered media formats. The result of this exchange is an agreement between the two parties as to the types of media they are willing to share.

The Structure of SDP

SDP is specified in RFC 2327. Since the initial publication of RFC 2327, however, a number of modifications to the protocol have been suggested and a draft update to the specification has been prepared within the IETF. In addition, RFC 3264 entitled "An Offer/Answer Model with SDP" has been prepared to describe how SDP and SIP should be used together.

SDP simply provides a format for describing session information to potential session participants. Basically, a session is comprised of a number of media streams. Therefore, the description of a session involves the specification of a number of parameters related to each of the media streams. Information also exists that is common to the session as a whole, however. Therefore, we have session-level parameters and media-level parameters. Session-level parameters include information such as the name of the session, the originator of the session, and the time(s) that the session is to be active. Media-level information includes the media type, port number, transport protocol, and media format. Figure 5-14 illustrates this general structure.

Because SDP simply provides session descriptions and does not provide a means for transporting or advertising the sessions to potential participants, it must be used in conjunction with other protocols (such as SIP). For example, SIP carries SDP information within the SIP message body.

Similar to SIP, SDP is a text-based protocol, utilizing the ISO 10646 character set in *Unicode Standard Transmission Format* (UTF)-8 encoding (RFC 2044). This coding enables the usage of multiple languages and includes US-ASCII as a subset. Although SDP field names use only US-ASCII, textual information may be passed in any language. For example, the field name that indicates the name of a session must be in US-ASCII, but the actual string of characters that comprises the session name may be in any language.

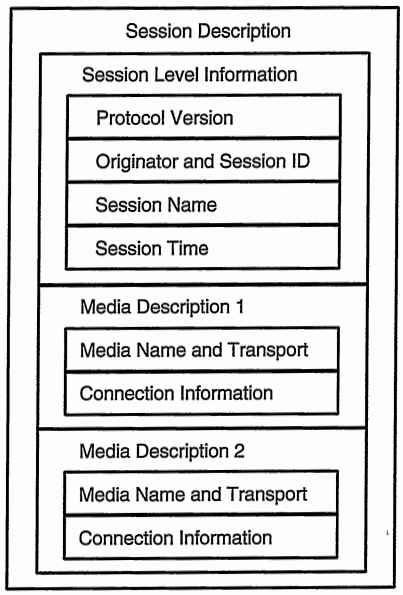


Figure 5-14
SDP session description structure

Although the use of ASCII coding in SDP, as opposed to binary coding, is a little bandwidth greedy, SDP is written in a compact form to counteract bandwidth inefficiency. Thus, where English words might have been used for field names, single characters are used instead. For example, *v* is version, *s* is session name, and *b* is bandwidth information.

SDP Syntax

SDP conveys session information using a number of lines of text. Each line uses the format field=value, where field is exactly one character (case-significant) and value is dependent on the field in question. In some cases, value may comprise a number of separate pieces of information, separated by spaces. No spaces are allowed between field and the = sign or between the = sign and value.

Session-level fields must be included first, followed by media-level fields. The boundary between session-level data and media-level data occurs at the first occurrence of the media description field (m=). Each subsequent occurrence of the media description field marks the beginning of data related to another media stream in the session.

Mandatory Fields SDP includes mandatory fields, which must be included in any session description, and optional fields, which may be omitted. The following fields are mandatory:

- **v=** Meaning the protocol version, this field marks the start of a session description and the end of any previous session description. When SDP is used as part of a SIP message body, only one SDP description should be present, which means that the v= field should appear only once in an SDP-based SIP message body.
- **o=** The session origin or creator and session identifier.
- **s=** Standing for the session name, this is a text string that could be displayed to potential session participants. The use of the session name is quite appropriate for a multicast session, but not for a unicast session such as a simple call between two parties. Therefore, when SDP is used in a SIP message body, the session name should be just a single space. SDP does not permit the s= line to be absent, nor does it permit the session name to be empty.
- **t=** This provides the start and stop times for the session. Unlike a pre-arranged multicast conference, however, sessions established by SIP are typically established in real time and can be of any duration. Consequently, the notion of predefined start and stop times does not apply to a SIP-established session. For that reason, an SDP description in a SIP message body should have a t= line of the form t=0 0. Strictly speaking, SDP interprets such time values to mean that the session is permanent. Such an interpretation does not matter in a SIP-established session, however, since SIP signaling will be responsible for the setup and teardown of the session, regardless of the time values indicated in the SDP description.
- **m=** This field indicates the media type, the transport port to which the data should be sent, the transport protocol (such RTP), and the media format (typically an RTP payload format). The appearance of this field also marks the boundary between session-level and media-level information or between different media descriptions.

Optional Fields Of the optional fields in SDP, some apply only at the session level or the media level, while some optional fields can be applied at both. In such cases, the value applied at the media level overrides the session-level value for that media instance. Thus, if a given field has value *X* for the session and value *Y* for media type 1, then value *X* applies to all media except for media type 1, where value *Y* applies. The following are the optional fields:

- **i=** Representing the session information, this is a text description of the session topic, meant to provide greater detail than the session name. This field may be specified both at the session and media levels. When SDP is used in a SIP message body, the i= field could be included, but would be somewhat superfluous, since SIP already supports the Subject: header field to convey session topic/subject information.
- **u=** This is a URI (such as a web address) where further session information may be obtained. For example, a session agenda might be posted on a web page, in which case the URI of the web page would be specified. Only one URI may be specified per session. Unlike pre-arranged conferences, SIP sessions are typically established on demand. Therefore, one is unlikely to find the u= line in an SDP description within a SIP message body.
- **e=** This is the e-mail address of the individual responsible for the session. Several instances of this field may exist and would be used if several individuals could be contacted for more information. This field applies only at the session level.
- **p=** This is a phone number for the individual responsible for the session. As for the e-mail address field, several instances of this field may be included. This field applies only at the session level. Similar to several of the other optional fields, this field is unlikely to be found in an SDP-based SIP message body.
- **c=** This field provides connection data, including the connection type, network type, and connection address. This field can be applied either at the session or media level. Although we are listing the c= field as optional here, it is more correct to say the field is conditional. Because this field defines the network address to which media should be sent, it must be included somewhere in the session description either at the session level or at the media level, or at both. If the field appears at the session level, then it is optional at the media level and vice versa.
- **b=** This optional field specifies the bandwidth required in kilobits per second. It may be specified at either the session or media level, or both. b= indicates the bandwidth required either by the whole conference or by a given media type. When used at the conference level, it can be considered analogous to a traditional dial-in conference where a maximum number of lines are available.
- **r=** In the case of a regularly scheduled session, this field specifies how often and how many times a session is to be repeated. For a SIP-initiated session, this field should not be included.

- **z=** Indicating the timezone adjustments, this field is used with regularly scheduled sessions. In such cases, the schedule of sessions may span a change from standard time to daylight savings time or vice versa. Since such changes occur at different times in different parts of the world, confusion might exist as to the actual start time of a specific session, particularly if it is to occur in the spring or fall. This field enables the session creator to specify up front when such timing changes will occur. As with other time-related fields, this field should not be included when SDP is used as a SIP message body.
- **k=** This field provides an encryption key or specifies a mechanism by which a key may be obtained for the purposes of encrypting and decrypting the media. The field may be applied either at the session or media level or both.
- **a=** This field is used to describe additional attributes to be applied to the session or to individual media.

Ordering of Fields Because certain fields can appear both at the session and media levels, the ordering of fields is important to avoid ambiguity. SDP demands that the following order be used.

Session Level

- Protocol version (v)
- Origin (o)
- Session name (s)
- Session information (i) (optional)
- URI (u) (optional)
- E-mail address (e) (optional)
- Phone number (p) (optional)
- Connection information (c) (conditional, depending on whether specified at the media level)
- Bandwidth information (b) (optional)
- Time description (t) (should be set to "t=0 0" when SDP is used with SIP)
- Repeat information (r) (optional) (should be omitted when SDP is used with SIP)
- Time zone adjustments (z) (optional) (should be omitted when SDP is used with SIP)

- Encryption key (k) (optional)
- Attributes (a) (optional)

Media Level

- Media description (m)
- Media information (i) (optional)
- Connection information (c) (optional if specified at session level)
- Bandwidth information (b) (optional)
- Encryption key (k) (optional)
- Attributes (a) (optional)

Subfields Of the various fields specified in SDP, a number are structured in several subfields. In such cases, the value of the field is a combination of several values, separated by spaces. The SDP specification's structure is as follows:

```
Field=<value of subfield 1> <value of subfield 2> <value of
subfield 3>.
```

A number of the most important fields are described here. The reader is referred to the SDP specification for a complete definition of all fields.

Origin Origin (o) has six subfields—username, session ID, version, network type, address type, and address.

The username is the originator's log-in identity at the host machine or is "-" if no log-in identity applies.

The session ID is a unique ID for the session, most likely created by the host machine of the originator. In order to ensure that the ID is unique, the SDP specification recommends that the ID makes use of a *Network Time Protocol* (NTP) timestamp.

The version is a version number for this particular session. When a session is advertised, the version number is important for the purpose of distinguishing a revised version of the session from an earlier version.

The network type is a text string indicating the type of network. The string IN refers to Internet.

The address type is the type of address in the network. SDP defines IP4 and IP6 to denote IP version 4 and IP version 6 respectively.

The address is the network address of the machine where the session was created. This address can be a fully qualified domain name or the actual IP address. The IP address can be the actual IP address only if the

IP address is unique as far as other session participants are concerned, which might not be the case if it is a local address, such as an IP address on a LAN behind a proxy.

Connection Data Connection Data (c) has three subfields: network type, address type, and connection address. Although some of these have the same names as the subfields defined for the origin, they have different meanings. They refer to the network and address at which media data are to be received, as opposed to the network and address at which the session was created.

Network type indicates the type of network in question. Currently, only the value IN is defined, meaning Internet.

Address type will have either the value IP4 or IP6 referring to IP version 4 or IP version 6.

The Connection address is the address where data should be sent. Although it may be a dotted decimal IP address, a fully qualified domain name is also possible and can be a better choice as it enables more flexibility and less ambiguity.

Media Information Media Information (m) has four subfields: media type, port, transport, and format. The media type may be audio, video, application, data, or control. For voice, the media type will be audio.

The port subfield indicates the port number to which media should be sent. The port number depends on the type of connection in question and the transport protocol. However, for VoIP, the media will normally be carried using RTP over UDP. Thus, the port number will be an even value between 1,024 and 65,535.

The format subfield lists the various types of media format that can be supported. If a given user can support voice coded in one of several ways, each supported format is listed with the preferred format listed first. Normally, the format will be an RTP payload format, with a corresponding payload type. In this case, it is necessary only to specify that the media is according to an RTP audio/video profile and specify the payload type.

If we have a system that wants to receive voice on port number 45678 and can only handle speech coded according to G.711 μ-law, then RTP payload type 0 applies and the media information is as follows:

```
m=audio 45678 RTP/AVP 0
```

If we have a system that wants to receive voice on port 45678 and can handle speech coded according to G.728 (payload type 15), *Global System*

for Mobile Communications (GSM) (payload type 3), or G.711 μ-law (payload type 0) and would prefer to use G.728, then the media information is as follows:

```
m=audio 45678 RTP/AVP 15 3 0
```

Attributes SDP includes a field called attributes (a), which enables additional information to be included. Attributes may be specified at the session or media level or both. Furthermore, multiple attributes may be specified for the session as a whole and for a given media type. Therefore, numerous attribute fields may be present in a single session description, with the meaning and significance of the field dependent upon its position within the session description. If an attribute is listed prior to the first media information field, then it is a session-level attribute. If it is listed after a given media information field (m), then it applies to that media type.

Attributes can have two forms. The first form is a property attribute, where the attribute specifies that a session or media type has a specific characteristic. The second form is a value attribute, which is used to specify that a session or media type has a particular characteristic of a particular value. The SDP specification describes a number of suggested attributes.

Examples of property attributes in SDP are "sendonly" and "recvonly." The first of these specifies that the sender of the session description wants to send data but does not want to receive data. If RTP is being used, the entity in question will not receive an RTP stream, but should still receive RTCP messages. In the case of recvonly, the entity in question will not send any RTP streams, but should still send RTCP data.

An example of a value attribute would be "orient," used in a shared whiteboard session and indicating whether the whiteboard has a portrait or landscape orientation.

These examples would be included in SDP using the following syntax:

```
a=sendonly
a=recvonly
a=orient:landscape.
```

One important use of attributes in a VoIP application is provided by the *rtpmap* attribute. This attribute can be applied to a media stream and is particularly useful when the media format is not a static RTP payload type. rtpmap has the format

```
a=rtpmap:<payload type> <encoding name>/<clock rate>[/<encoding
parameters>].
```

The payload type corresponds to the RTP payload type. The encoding name is a text string. RTP profiles that specify the use of dynamic payload types must define the set of valid encoding names and/or a means to register encoding names if that profile is to be used with SDP. The clock rate is the rate in Hz. The optional encoding parameters may be used to indicate the number of audio channels.

Strictly speaking, the rtpmap attribute is only necessary for dynamic payload types. For example, standard G.711 speech is a static RTP payload type and would be fully described by a media field such as

```
m=audio 45678 RTP/AVP 0
```

A dynamic RTP payload type needs greater information for the remote end to fully understand the media coding, however. An example of such a dynamic payload type is 16-bit linear encoded stereo (2 channels) audio sampled at 16 kHz. If we want to use dynamic RTP/AVP payload type value 98 for such a stream, then the media description must be followed by an rtpmap attribute as follows:

```
m=video 45678 RTP/AVP 98
a=rtpmap 98 L16/16000/2
```

If multiple payload types are indicated by the media description, then a separate rtpmap attribute may be specified for each.

Although the rtpmap attribute is not necessary for static RTP payload types, SIP recommends that the attribute always be included when SDP is used for a SIP message body, even in the case of static payload types.

Usage of SDP with SIP

SIP and SDP make a wonderful partnership for the transmission of session information. SIP provides the messaging mechanisms for the establishment of multimedia sessions and SDP provides a structured language for describing those sessions. The message body in SIP, identified by the entity headers, provides a neat slot where SDP can be used. It is this usage that this section will discuss.

SIP uses SDP in an offer/answer model. The initiator of a session offers a selection of media formats to be used in the session. The receiver of the offer either rejects the offer completely or selects those media formats that have been offered and which the responder is willing to accept. If the

responder agrees to participate in the session, then the response to the initial offer (a SIP INVITE) will include a message body of the same form as the received message body, with an indication as to the acceptance or rejection of each of the offered media types.

If we take the example outlined in Figure 5-9 and assume that Collins can support voice coded according to G.726, G.723.1, or G.728 and we assume that Boss can only support voice coded according to G.728, then the messages would appear as shown in Figure 5-15. For avoidance of clutter, a number of header fields (such as Via:) have been omitted, as has the 180 (Ringing) response. These would be included according to Figure 5-9.

In Figure 5-15, the message body of the INVITE lists the individual media formats separately, port numbers at which Daniel wants to receive each of the formats, and an rtpmap attribute exists for each media format. The message body of the 200 (OK) response also lists the media formats individually in the same order as received in the INVITE. In this example, Boss can support only G.728 and is rejecting the other two media formats. This rejection is indicated by specifying port number zero for each rejected media format. The fact that Boss is willing to accept G.728 is indicated by the fact that the associated port number (6666 in our example) is nonzero. The 6666 port number is the port at which Boss is willing to receive the G.728 RTP media stream. Since both parties can support the same media format, and each knows the port number at which the other party wishes to receive the media, media exchange (conversation) can take place.

In Figure 5-15, each of the offered media formats was listed in a separate line with a separate port number associated with each format. If Daniel wants to receive only one of the offered media formats and to reserve just a single number (port number 4444, for example), then the media part of the offer could be as follows:

```
m=audio 4444 RTP/AVP 2 4 15
a=rtpmap 2 G726-32/8000
a=rtpmap 4 G723/8000
a=rtpmap 15 G728/8000
```

This syntax indicates that Daniel wants to use G.726 (2), G.723 (4), or G.728 (15) in that order of preference, and he wants to receive the media stream at port number 4444.

If Boss is to respond to such an offer, then the response would have the following format (again assuming that Boss chooses only G.728):

```
m=audio 6666 RTP/AVP 15
a=rtpmap 15 G728/8000
```

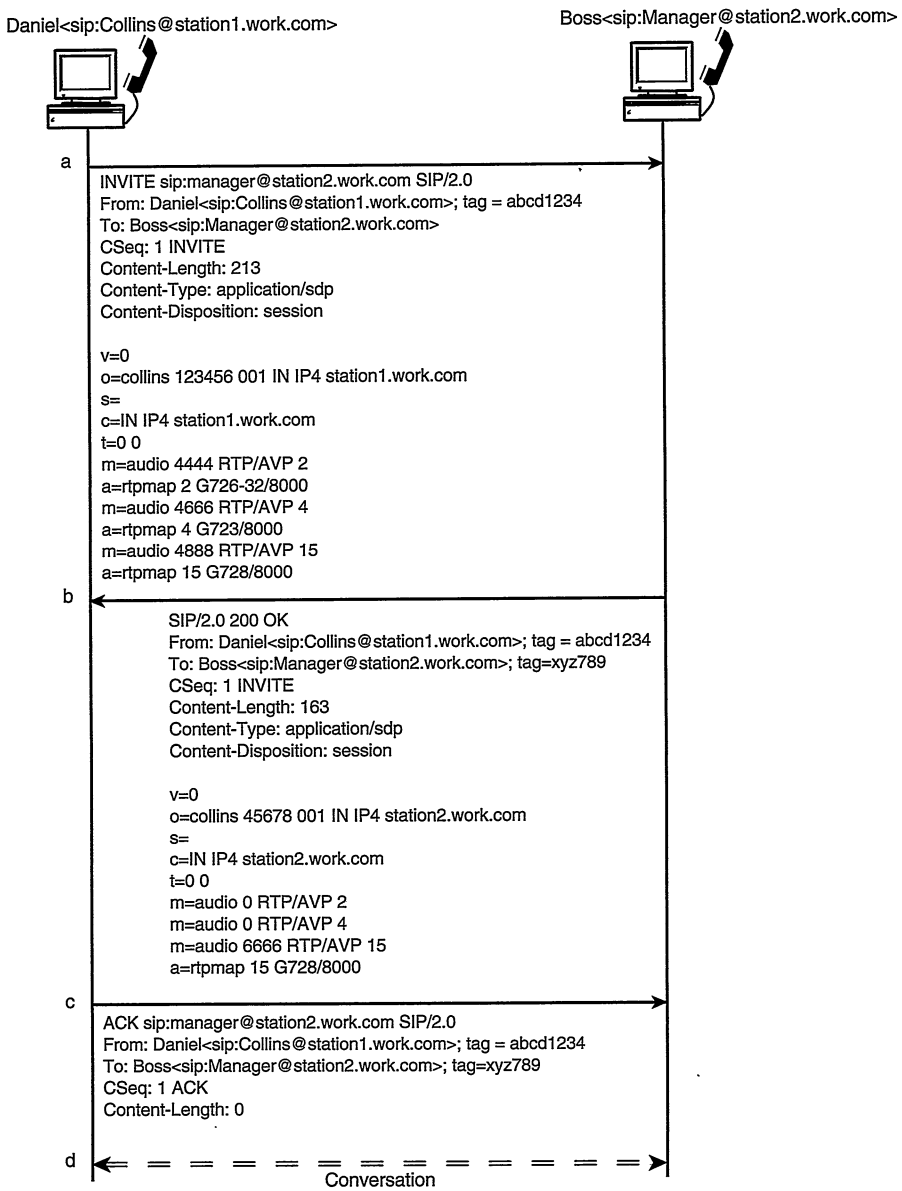


Figure 5-15
SDP inclusion in SIP messages

Negotiation of Media

Situations can arise where one party offers a number of media types and the responding party can support several of the offered media types. In the

example of Figure 5-15, we assume that Boss supports only G.728. If Boss were to indicate that both G.723 and G.728 can be supported, then we have reduced the selection of *coder/decoders* (codecs) from three to two, but we have still not finalized exactly which codec will be used. Ideally, Boss's device should select just a single codec from the offered set. Such a selection would eliminate any confusion as to the codec to be used.

If, however, Boss were to reply with more than one codec, then the initiator of the session should choose which of the two to use and should generate a new offer that includes just a single codec. For SIP, this reoffer would mean the issuance of a new INVITE request with a message body that includes just the single selected codec. This new INVITE would exist within an already established dialog and, as such, it would use the same dialog identifier (From and To headers, including tag values), Call-ID, and Request-URI as would be used by any other request (such as INFO or BYE) within an established dialog. This scenario would appear as shown in Figure 5-16.

In Figure 5-16, we note that the initial INVITE specifies the offered media formats, but it uses the inactive attribute to specify that the media formats are set such that the media in question will neither be sent nor received (at least not for the present). As in Figure 5-15, Daniel offers the three coding schemes G.726, G.723, and G.728. In this case, Boss replies with G.723 and G.728. Collins acknowledges the response and issues a re-INVITE with G.728 as the only coding choice, and this time the inactive attribute is not present. The o= line of the re-INVITE indicates the same sender and session number, but the session version is increased by 1.

Although an ACK can carry a message body, we cannot use the ACK request in Figure 5-16 to carry an updated version of the offer that was in the initial INVITE. In SIP, two mechanisms are used for the answer/offer exchange. In the first case, an offer is made within an INVITE and is answered with a 2XX response. In the second case, an offer is made within a 2XX response and the ACK carries the answer. The second case would apply if the initial INVITE does not contain an offer (a valid situation). If the first INVITE does contain an offer in the form of an SDP description, however, the ACK cannot be used to modify that offer.

Not only can there be overlap between the supported media formats of the two parties in a session, but it is quite possible that the receiver of an offer (the receiver of an INVITE that contains an SDP description) cannot support any of the media formats proposed in the offer. If, for example, in Figure 5-15, Boss could only support voice coded according to G.711 μ-law (RTP payload type 0), then there would be a mismatch between the capabilities of the called party and those of the calling party. In such a situation,

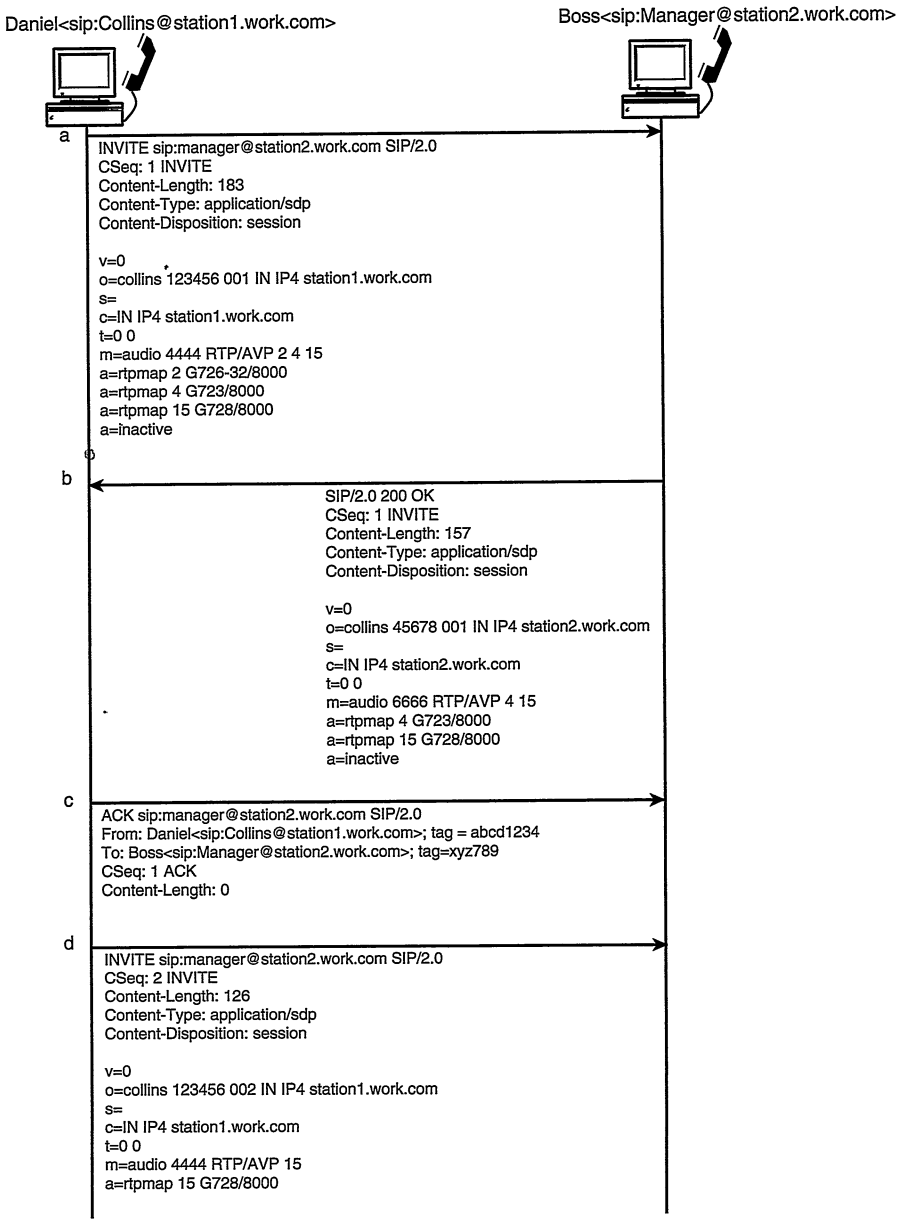


Figure 5-16
SIP and SDP offer/answer model

the response to the INVITE should contain a 488 (Not Acceptable) or a 606 (Not Acceptable) status code. In addition, the response should contain a Warning: header field, with the warning code of 304 (media type not available) or 305 (incompatible media type). In that case, the caller could choose to issue a new INVITE request.

The fact that VoIP enables the use of a range of voice-coding techniques means that there can be a mismatch between the abilities of different SIP users. When some doubt exists about a party's ability to handle a particular media type, then the OPTIONS method provides a useful mechanism for finding out in advance. This avoids the partial establishment of a session that is doomed to fail.

OPTIONS Method A potential caller can use the OPTIONS method to determine the abilities of a potential called party. The recipient of the OPTIONS request should respond with the abilities supported.

Figure 5-17 shows an example of the OPTIONS request. In this case, Collins queries the abilities of Boss, who responds with an indication that he can accept speech coded according to G.723 and G.728 (RTP payload formats 4 and 15 respectively).

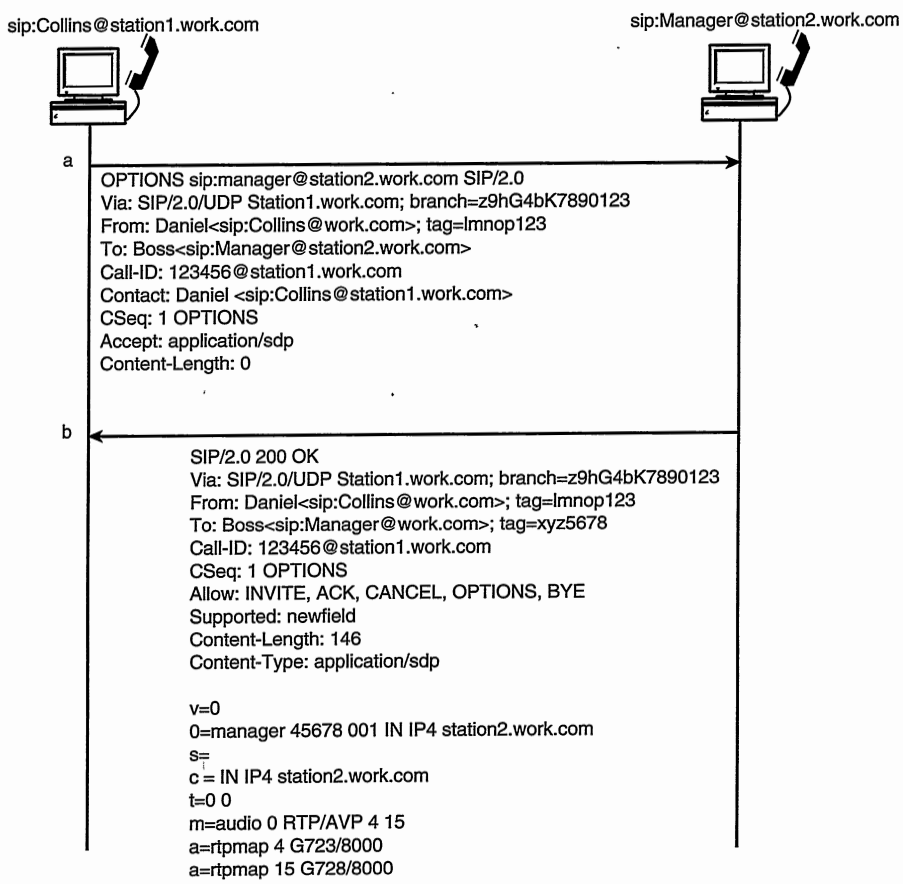


Figure 5-17
Usage of the OPTIONS method

We see the Accept: header field in the OPTIONS request. This field indicates the type of information that the sender is hoping to receive in the request. By specifying the value "application/sdp" in the Accept: header field, Collins is requesting that Boss respond with an SDP description for the type of SDP sessions that Boss can accept. If the OPTIONS request does not include the Accept: header, the application/sdp is used as a default.

In the response, we see the use of the Allow: header field. This field indicates those SIP methods that Boss can handle. We also see the Supported: header field, which indicates any extensions to SIP that Boss is able to support. For example purposes, Figure 5-17 shows that Boss can support an extension called "newfield." The Supported: header field provides a means for different parties to indicate those SIP extensions that they support. If both parties support the same extensions, then they can use them to provide additional services or features. SIP extensions are discussed later in this chapter.

SIP Extensions and Enhancements

Since RFC 2543 was released in March 1999 as an Internet standards track protocol, SIP has attracted enormous interest. Many companies, including those from the traditional telecommunications environment and others such as cable TV providers and *Internet Service Providers* (ISPs), have recognized that SIP offers great potential. In the process, however, several individuals and companies have found that SIP would be better if some enhancements or extensions were added. A large number of extensions to SIP have been proposed and are being handled in the SIP working group of the IETF.

As of this writing, numerous revisions have been made to the SIP specification as part of its progress along the standards track from proposed standard to standard. In addition, a number of Internet drafts have been prepared that we can expect to be included in the SIP specification in the future or that may become RFCs in their own right. The following descriptions address some of the most significant extensions to SIP.

One example of an extension that has been included within the revised SIP specification is the 183 (session progress) response code, which is typically used to open a one-way audio path from called end to calling end. The 183 response is often used when interworking with the SS7 network, where the Address Complete Message is used to open such a one-way path so that ring-back tone can be relayed from the called end to the calling end.

Motorola - Exhibit 1022 - Page 233

Another addition to the latest SIP specification (RFC 3261) is the addition of the Supported: header field. The base RFC 2543 specification includes the Require: header field, which enables a client to specify those extensions or options that a server must support in order to process a request. RFC 2543 does not, however, enable a server to independently indicate what extensions or options that it supports. Thus, if a client wants to know whether a server supports a given extension, it specifies that extension within a Require: header of a request (such as INVITE) and waits to see if an acceptable response is returned. If a 420 (bad extension) response is returned with an Unsupported: header field, then the client will know that the server does not understand the extension or option in question. Clearly, this is a cumbersome way of determining what extensions a server does or does not support. The Supported: header removes this shortcoming and provides a more elegant means for a client to determine what extensions a server supports. If a client issues an OPTIONS request to a server, then the server should include a Supported: header to indicate what extensions it has available.

The Supported: header can also be used in requests and should be used in requests from a client that supports SIP extensions. Associated with the Supported: header is the 421 (extension required) response. This response is returned by a server when it requires a particular extension in order to process a request and the request itself did not indicate that the client supports the extension.

The SIP INFO Method

The SIP INFO method is specified in RFC 2976. This method is a means for transferring information during an ongoing session, such as in the middle of a call. Examples of its use could include

- Transferring DTMF digits
- Transferring account balance information
- Transferring midcall signaling information generated in another network (such as the PSTN) and passed to the IP network via a gateway

Through the use of the INFO method, application-layer information could be transferred in the middle of a call, thereby affording user-agent servers, user-agent clients, or proxies the opportunity to act upon the information. For example, a prepaid subscriber could be informed in the middle of a call if the subscriber's prepaid account is nearing zero.

The SIP INFO method is a flexible tool and, as such, does not define the information that may or may not be sent using the INFO method. Typically,

the INFO method would convey the user information within a message body, with the content and structure of the message body dependent upon the type of information to be conveyed. For example, an indication that the user's prepaid account balance is near zero could be conveyed in a message body in the form of text.

SIP Event Notification

Several SIP-based applications have been devised based on the concept of a user being informed of some event or events. For example, significant work has been performed related to use of SIP for instant messaging, and an important part of instant messaging is presence, that is, knowing when a member of a buddy list is available. The concept of event notification, however, can be applied in numerous applications. For example, a user might want to be informed of any schedule changes for a flight or of a change in a stock price.

To address these issues, RFC 3265 has been prepared to address the issue of event notification. That RFC specifies a general mechanism for event notification, but it does not specify the specific events. Any application that would want to avail itself of event notification would require a separate specification of the details of the events in question.

SIP-based event notification is achieved through the use of two new SIP methods: SUBSCRIBE and NOTIFY. The SUBSCRIBE method enables a user to subscribe to certain events, which means that the user should be informed when such events occur. The NOTIFY method is used to inform the user of the occurrence of a subscribed event. Figure 5-18 shows a basic example of the use of these two SIP methods.

The user who wants to subscribe to certain events sends a SUBSCRIBE method to a server that is aware of the events in question. The SUBSCRIBE request must include a new header field called Event:, which is used to identify an event package including those events of interest to the subscriber. The content of the Event: header field might explicitly or implicitly indicate the events of interest, or the content of the Event: header field might indicate that the details of the events of interest are carried within the SUBSCRIBE message body.

For example, if a user wants to know about the presence and state of Instant Messaging buddy list members, then it would send a SUBSCRIBE request to the Instant Messaging server. The Event: header would simply need to indicate that the user wants to be informed about his buddy list. The server would know of the buddy list contents and, provided that the

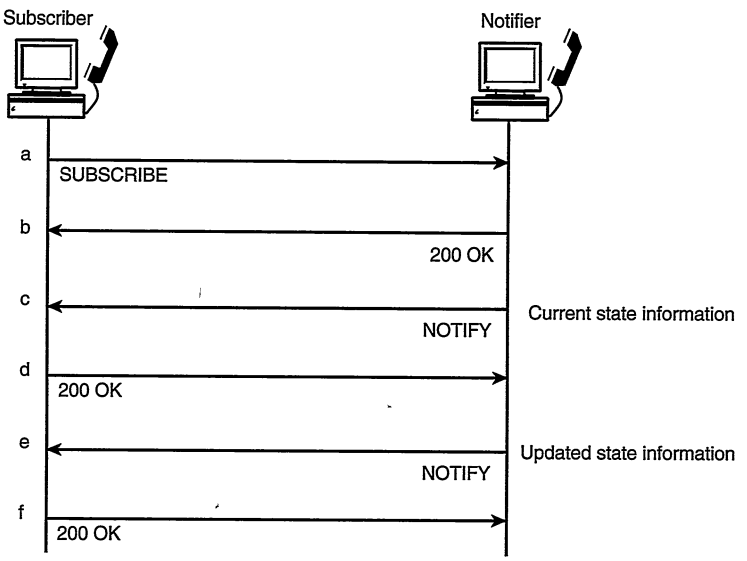


Figure 5-18
SIP event notification

subscriber is authenticated, would have enough information in the Event: header to be able to send the required notifications to the subscriber. On the other hand, if a subscriber wants to be informed of stock prices for an arbitrary set of stocks, then the list of stock ticker symbols might be carried as text within the message body, and the Event: header would simply provide information to enable the server to understand the meaning of the message body.

Upon receipt of a SUBSCRIBE request and assuming that the subscriber is authorized to receive the requested information, the server will respond with a 200 (OK) response. The server will then send a NOTIFY message to indicate the current state of affairs for the requested information (the current prices for the stocks in question, for example). The server will then send a new NOTIFY when state changes occur, such as if a stock price changes or if a buddy list member becomes active. The NOTIFY request is likely to include the requested information within a message body. Provided that no errors occur in a NOTIFY message (such as information about an unsubscribed event), then the subscriber will issue a 200 (OK) response to each NOTIFY.

As mentioned, the SUBSCRIBE-NOTIFY mechanism is a framework for enabling users to subscribe to specific information. Those who want to use the mechanism in support of event notification for some specific application must develop detailed specifications for the manner in which the mechanism is to be used. Such detailed specifications must include the content of

the Event: header and the format of the message bodies to be used in the SUBSCRIBE and NOTIFY requests. One example of such specification details can be found in an Internet draft that details how the SUBSCRIBE-NOTIFY technique can be used for presence information.

SIP for Instant Messaging

We have already discussed the usage of SIP event notifications in relation to presence information for instant messaging applications. In fact, SIP has been proposed by many as a unifying protocol to replace the several incompatible protocols that have existed to date. To that end, a separate IETF working group has been established to develop a specification for the use of SIP for instant messaging. That working group is titled *SIP for Instant Messaging and Presence Leveraging Extensions* (SIMPLE).

The SIMPLE working group has already developed an Internet draft for presence functions, an important aspect of instant messaging. Another important aspect is the mechanism by which the instant messages themselves are passed between users. One solution is a new SIP method, the MESSAGE request, which is proposed in an Internet draft.

Instant messaging typically involves the exchange of content between a set of participants in near real time. Generally, the content is short text messages, although that need not be the case, and other content (such as images or other binary data) can also be transferred. Generally, the messages that are exchanged are not stored, but this also need not be the case. Instant messaging differs from e-mail in common usage in that instant messages are usually grouped together into brief live conversations. Although some instant messaging protocols do establish a session between users, most often individual messages are stand-alone, and no explicit relationship exists between messages. Although there can be the appearance of a session between users, that appearance is often a function of the client-user interface or just a function of the conversation that takes place.

SIP is well suited to a session-based instant messaging solution, where the session would be instigated with an INVITE request and terminated with a BYE request. Supporting an instant messaging regime where each message is stand-alone, however, needs an extension to the basic SIP protocol. The proposed extension is the new MESSAGE method.

A user that wants to send an instant message to another user does so through the issuance of a MESSAGE request. This request carries the actual message in a message body. The message body will typically be of the form *text/plain* or may be in the form *message/cpim*, where cpim stands for

common presence and instant message format, an IETF format designed for instant message data.

Unlike an INVITE, a MESSAGE request does not establish a SIP dialog. MESSAGE requests can, however, be associated with an existing SIP dialog. An instant message sent between two parties during an existing session would be associated with the dialog of that session. In many cases, however, a MESSAGE request will simply be sent between two users and will not be associated with any session. Therefore, if a MESSAGE request is sent outside an existing dialog, it is not required that any tag value be included in the To: header.

Figure 5-19 shows an example of the MESSAGE method as a means for passing instant messages. Two messages are shown. In the first case, Boss sends a message to Daniel via a proxy. When the message is delivered, a 200 (OK) response is returned. Subsequently, Daniel wants to respond to the instant message and constructs a MESSAGE request of his own. This request is also sent via a proxy and a 200 (OK) response is returned. Note that both of the messages are sent via the same proxy in Figure 5-19. From a protocol perspective, the two MESSAGE requests are totally independent. Therefore, the MESSAGE request from Daniel to Boss might traverse a different proxy (or proxies) than the MESSAGE request from Boss to Daniel. In Figure 5-19, we see that none of the Contact:, Record-Route:, or Route: headers are included. The Contact: header is strictly forbidden with the MESSAGE method. The Record-Route: and Route: headers are not relevant to the MESSAGE method because those headers are related to a specific dialog, and the MESSAGE method does not establish a SIP dialog.

The SIP REFER Method

Like many of the proposed SIP extensions, the REFER method is currently documented within the IETF in the form of an Internet draft. This particular SIP extension, however, is extremely popular and has been implemented in many SIP devices.

A REFER request enables the sender of the request to instruct the receiver to contact a third party, with the contact details for the third party included within the REFER request. A common application of the REFER method is in Call Transfer applications.

Imagine, for example, that Joe and Mary are having a conversation. Mary determines that Joe really needs to talk to Susan and decides to transfer the call to Susan. Mary would send a REFER message to Joe, with Susan's contact details contained with a new Refer-to: header field, and Joe

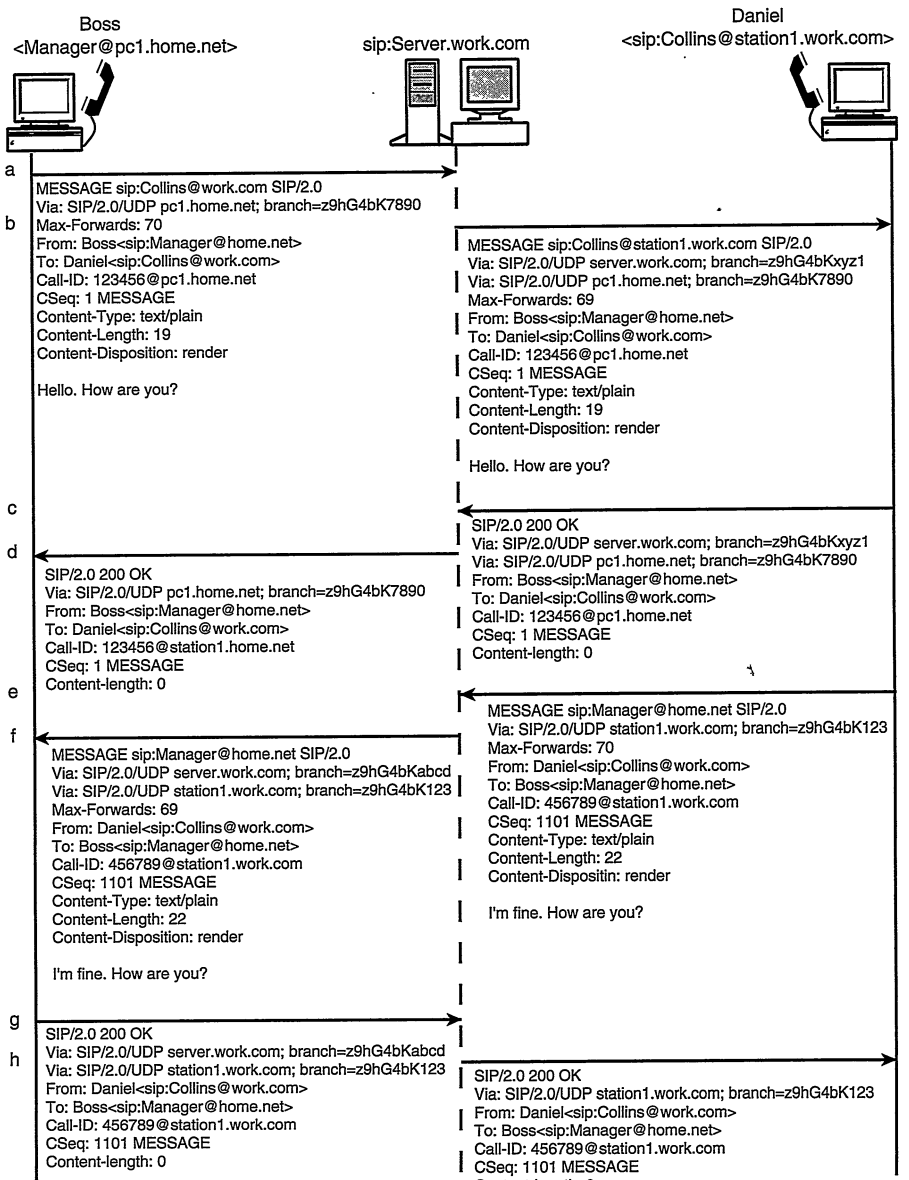


Figure 5-19
SIP MESSAGE method for instant messaging

can then contact Susan. Moreover, the REFER method can be interpreted as an implicit SUBSCRIBE request, so that Joe can subsequently send a NOTIFY request to Mary to indicate whether he was able to successfully contact Susan. Figure 5-20 shows an example of the use of the REFER method.

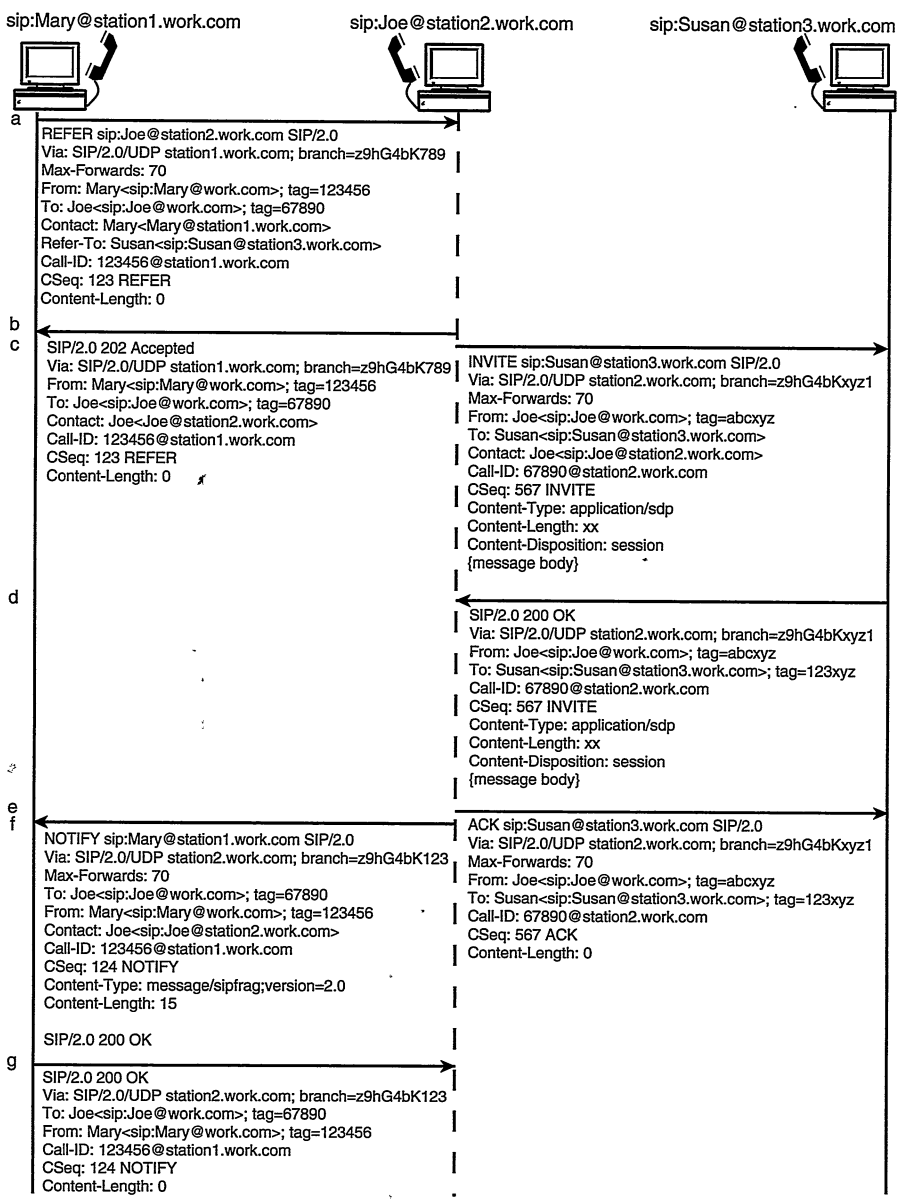


Figure 5-20
Usage of the SIP REFER method

In Figure 5-20, the REFER and NOTIFY requests between Mary and Joe are sent within an established dialog. Once Joe receives the REFER request from Mary, he issues a 202 (accepted) response. This is a new status code that is not defined in the base SIP specification, but it is specified in several

SIP extensions. After sending the 202 response, Joe proceeds to send an INVITE request to Susan.

Once Susan accepts the INVITE request, Joe sends an ACK as normal. He (or his device) also sends a NOTIFY request to Mary to indicate that the actions performed as a result of the referral have been successful. The NOTIFY message has a message body that contains part of the SIP response received from Susan. The inclusion of a SIP message fragment is indicated by the "Content-Type: message/sipfrag; version=2.0" line of the NOTIFY request. This form of message content type is defined within a separate Internet draft, and it enables a SIP message part to be tunneled within a SIP message. If the content type had been "Content-Type: message/sip, version=2.0," then the whole of the 200 (OK) response would be included in the message body.

Although not shown in Figure 5-20, yet another Internet draft defines the Referred-By: header, which can be included in a REFER message and can optionally be included in an INVITE that occurs as a result of a REFER request. In the example of Figure 5-20, the Referred-By: header would enable Joe to indicate to Susan that he is contacting her at Mary's suggestion (which might make Susan more likely to accept his call).

In Figure 5-20, the dialog between Mary and Joe remains established, and Joe could return to the dialog after consultation with Susan. If either Mary or Joe wants to terminate the dialog, then they can do so in the normal manner through the use of the BYE request.

Reliability of Provisional Responses

A SIP session is instigated with an INVITE message. One or more provisional responses might occur, such as 100 (trying) or 183 (session progress) prior to a final response to the INVITE. SIP can use UDP transport, which is unreliable. Therefore, it is possible that messages sent from one end might never reach the other end. SIP addresses some of these issues by specifying rules for the retransmission of a request or a response. Moreover, SIP includes the ACK as a means to confirm that a final response has been received. Provisional responses do not generate an ACK, however. Therefore, a provisional response could be lost forever.

Imagine, for example, an INVITE sent from A to B. B responds with the provisional response such as 180 (ringing), but this response is not received at A. The called user then picks up the phone and B issues a final response of 200 (OK), which is received at A prior to expiry of the retransmission timer for the INVITE. A is happy that a final response has been received

and the call is established. However, there might be a problem, particularly in the case of interworking with other networks.

In the case of interworking, the 180 (ringing) response or the 183 (session progress) response can be mapped to the Q.931 Alerting message or the ISUP *Address Complete message* (ACM). External network elements might be expecting such messages/responses to drive state machines, or certain applications might be relying upon guaranteed delivery to ensure that a particular feature or service operates correctly. A simple example would be a call to an unassigned number. The SS7 ACM message is used to open a one-way path from the called end to the calling end in order to relay an announcement such as "The number you have called has been changed." If the ACM is mapped to a 180 or 183 response and that response is not delivered to the calling end, then the caller might be left in the dark and not understand why the call did not connect. Therefore, a SIP extension has been specified to provide reliability for SIP provisional responses.

The document describing this SIP extension is titled "Reliability of Provisional Responses in SIP" (RFC 3262). The extension involves the addition of two new headers: *response sequence* (RSeq) and *response ACK* (RAck). RAck is a request header field, and RSeq is a response header field. The extension also defines an option tag to be used with the "supported" SIP header. This option tag is 100rel. The tag may also be included in the "unsupported" header to indicate that a server does not understand the option. The extension also defines a new SIP method, *Provisional Response ACK* (PRACK).

The extension works according to the following example. A client that supports the reliable delivery of provisional responses sends an INVITE, which includes the supported header with the option tag 100rel and the Require: header with the option tag 100rel. When a server receives the INVITE and determines that a provisional response should be delivered reliably, it sends the response and includes the RSeq header. It then waits for the client to send a PRACK message to confirm that the response has been received. If a PRACK message is received, then it is certain that the provisional response was received. Upon reception of the PRACK, the server returns 200 (OK) to indicate that the PRACK has been received. If the server does not receive the PRACK message, then it shall retransmit the response T1 seconds later (the default T1 value is 0.5 seconds or 500 milliseconds). If the server still does not receive a PRACK, then it retransmits the response every T1 seconds, up to a maximum of 64 times, after which the server assumes that a network failure has occurred.

The initial provisional response contains the RSeq header, which is defined to carry a 32-bit integer value between 1 and 4,294,967,295. The

initial value of RSeq chosen the first time that a given response is returned is a random number between 1 and 2,147,483,647. Each time the response is retransmitted, the value of RSeq is incremented by 1. The provisional response also includes the Require: header containing the 100rel option tag.

The PRACK sent by the client contains the RAck header, which contains two components. The first component is the value of the RSeq copied from the provisional response for which the PRACK is being sent. The second is a CSeq number and method token copied from the response for which the PRACK is being issued.

The process is shown in Figure 5-21. In this example, the calling client uses the Supported: header to indicate in the INVITE that it supports the acknowledgment of provisional responses. The first 180 (ringing) response to the INVITE is lost. After a timeout, the server retransmits the response. Note that both instances of the response include the RSeq: header, but that the value has been incremented in the second instance of the response.

Upon receipt of the retransmitted response, the calling client acknowledges the response through the use of the PRACK method. This contains the RAck: header field. The content of the field is the value received in the RSeq: header of the response, plus the content of the CSeq: header in the response. Finally, the server confirms receipt of the PRACK by issuing a 200 (OK) response. Note that this 200 (OK) response relates to the PRACK and not to the initial INVITE.

Although reliability may be applied to provisional responses, it must not be applied to the 100 (trying) response and the RSeq: header should not be included in such a response. The 100 (trying) response is sent on a hop-by-hop basis, whereas the reliability mechanism described in this extension is end to end and does not apply to the 100 (trying) response.

The SIP UPDATE Method

As already described in this chapter, a SIP session is established with an INVITE. The far-end response to the INVITE will include a tag parameter and will lead to the establishment of a dialog. Depending upon whether the response is a provisional (1XX) response or a final (2XX) response, the dialog is either early or confirmed.

If something about the session needs to change (the codecs being used, for example), then a re-INVITE can be issued. One problem with this approach, however, is that the reissuance of an INVITE changes the state of the dialog. A re-INVITE within an established session (one that has a confirmed dialog) will generally work well as a means of changing session

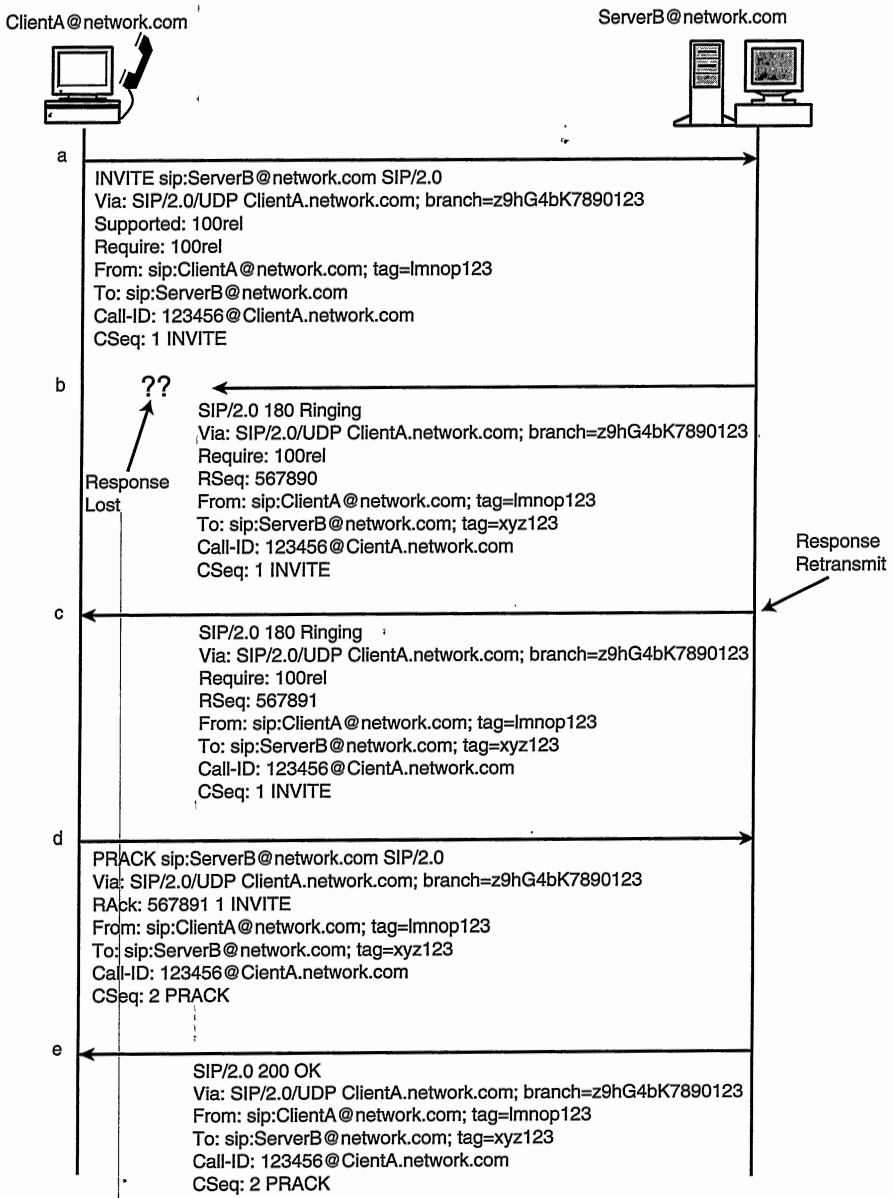


Figure 5-21
Reliability for provisional response

data such as media parameters. A re-INVITE for an early dialog, however, cannot be used for the same purpose. Consider, for example, an INVITE that receives a 183 (session progress) response that includes a message body. That message body might establish a media stream from callee to

caller for sending a ring tone or music while the called party is alerted. At this stage, the dialog is in the early state. If the caller wants to change any media parameters, such as the codec to be used, or wants to put the early media on hold, then a re-INVITE will not fill the need. Something else is required.

The solution is the new UPDATE method, which is specifically designed to enable the modification of session information before a final response to an INVITE is received. Operation of this extension is simple. The caller begins with an INVITE transaction, which proceeds normally. Once a dialog is established, either early or confirmed, the caller can send an UPDATE request method that contains an SDP offer for the purposes of updating the session. The response to the UPDATE method contains the answer.

Similarly, once a dialog is established, the callee can send an UPDATE with an offer, and the caller places its answer in the 2XX response to the UPDATE. The Allow header (in an INVITE, for example) can indicate support for the UPDATE method, so that one party can know if the other party supports the method.

One important usage of the UPDATE method is when reserving network resources as part of a SIP session establishment. We address this issue next.

Integration of SIP Signaling and Resource Management

Ensuring that sufficient resources are available to handle a media stream is a very important aspect of providing a high-quality service and is therefore a necessity for a carrier-grade network. For example, when a called party answers the phone and attempts to speak, it is imperative that the necessary resources are available to carry that speech back to the caller. Equally, resources must be available in the direction from caller to callee. Although customers may expect to experience occasional congestion, that experience should only happen while attempting to establish the call. Once the far end phone rings and is answered, a customer expects that conversation can take place. Anything else is considered defective operation.

In our discussion so far, we have mainly focused on the signaling related to call setup and teardown. The bandwidth required for signaling is miniscule compared to that required for handling the media to be exchanged, however. Furthermore, the signaling may take a very different path from the media. Therefore, we cannot assume that the successful transfer of sig-

naling messages will automatically lead to a successful transfer of media. If carrier-grade operation is desired, the signaling must accommodate some mechanism to ensure that resources are available in the network for the transport of the required media.

As we shall describe in Chapter 8, "Quality of Service," resource reservation can be achieved either on a per-session basis or on an aggregate basis. On a per-session basis, end-to-end network resources are reserved in real time as part of session establishment. On an aggregate basis, however, a certain amount of network resources are reserved in advance for a certain type of usage, and network resource utilization is constantly monitored. If a session is to be established under such circumstances, it is simply necessary to check that the resource demands of the new session will not cause the total usage for sessions of that type to exceed any pre-arranged reservation.

For example, if an IP network were used to carry voice, user data, and management traffic, then it would be common to reserve a certain percentage of resources for voice, a certain percentage of resources for user data, and a certain percentage of resources for management traffic. The management of access to those resources takes place at the edge of the network. Therefore, if a new voice session is to be established, policing functions at the edge of the network are invoked to verify that the new session would not cause the total usage to exceed any prereserved maximum.

Since two main approaches to network resource reservation exist, any SIP solution for resource reservation must be compatible with both approaches. In other words, SIP-related resource reservation must work in a regime where resources are reserved on a session-by-session basis and must also work in a regime where resources are reserved in advance and where new sessions are allowed only if total usage does not exceed a prereserved maximum. The remainder of this section addresses a SIP extension that supports both regimes.

An Internet draft was prepared in 1999 describing a means for reserving network resources in advance of alerting the called user. The draft specified the use of a new SIP header to be used in the INVITE, indicating to the called end that resource reservation is needed and that the user should not yet be alerted. After successful resource reservation, a second INVITE would be issued to set up the call. This approach, however, had some incompatibilities with standard SIP. If new headers were not supported by the called end, a call could be established without sufficient resources, since SIP states that new unrecognized headers should be ignored. Consequently, the extension was rejected. The basic need was recognized, however, and several groups have worked for some time to define a new Internet draft to address the issue in a different way. That draft is titled "Integration of

Resource Management and SIP." As for all Internet drafts, this specification is subject to change.

The new approach utilizes reliable signaling for provisional responses and also utilizes the UPDATE method described previously. The technique also involves extensions to SDP. In fact, the details of the session preconditions are specified through extensions to SDP. This approach is reasonable, since SDP describes the media to be exchanged. Given that the *quality of service* (QoS) requirements are related to the media, it is logical that they should be included as part of the session description rather than within SIP itself. Moreover, if a given session is to include multiple media streams, then different QoS requirements might be applied to the different streams. For example, a session that includes both audio and video might have stronger QoS requirements for the audio in the event that it is considered more important. If different resource requirements are to be applied to different media streams, then there is no choice but to specify those requirements within the message body, rather than within SIP.

Of course, since SIP is used to carry the session description, it is not possible to completely divorce the SIP signaling from the media to be exchanged. In fact, resource reservation is one of the main reasons for the introduction of the UPDATE method described previously. SIP, however, should not care what the resource requirements happen to be. By specifying the requirements at the media level, it is possible to define any number of requirements in SDP in the future without having to revise SIP every time.

The basic methodology is depicted in Figure 5-22. An INVITE carries a session description that describes the offered media and preconditions to be met. The called entity returns a 183 (session progress) response containing a session description that requests the caller to reserve resources in the caller-to-callee direction. This response is sent using reliable signaling (and therefore causes a PRACK message from the caller to callee) because it contains a message body requesting resource reservation and we cannot take a risk that the response might be lost. Once the 183 (session progress) message has been sent, both caller and callee reserve resources in their respective directions. Once the calling entity has reserved resources, it sends an UPDATE request to the called end. The UPDATE indicates that resources have been reserved from the calling end to the called end. The called entity responds to the UPDATE to indicate that resources have been reserved from the called end to the calling end. The called user is now alerted and the call proceeds as normal.

As mentioned, the preconditions/requirements are specified in the SDP description. Let's take a closer look at how those preconditions are specified. A given precondition is defined to have three statuses: desired, current, and

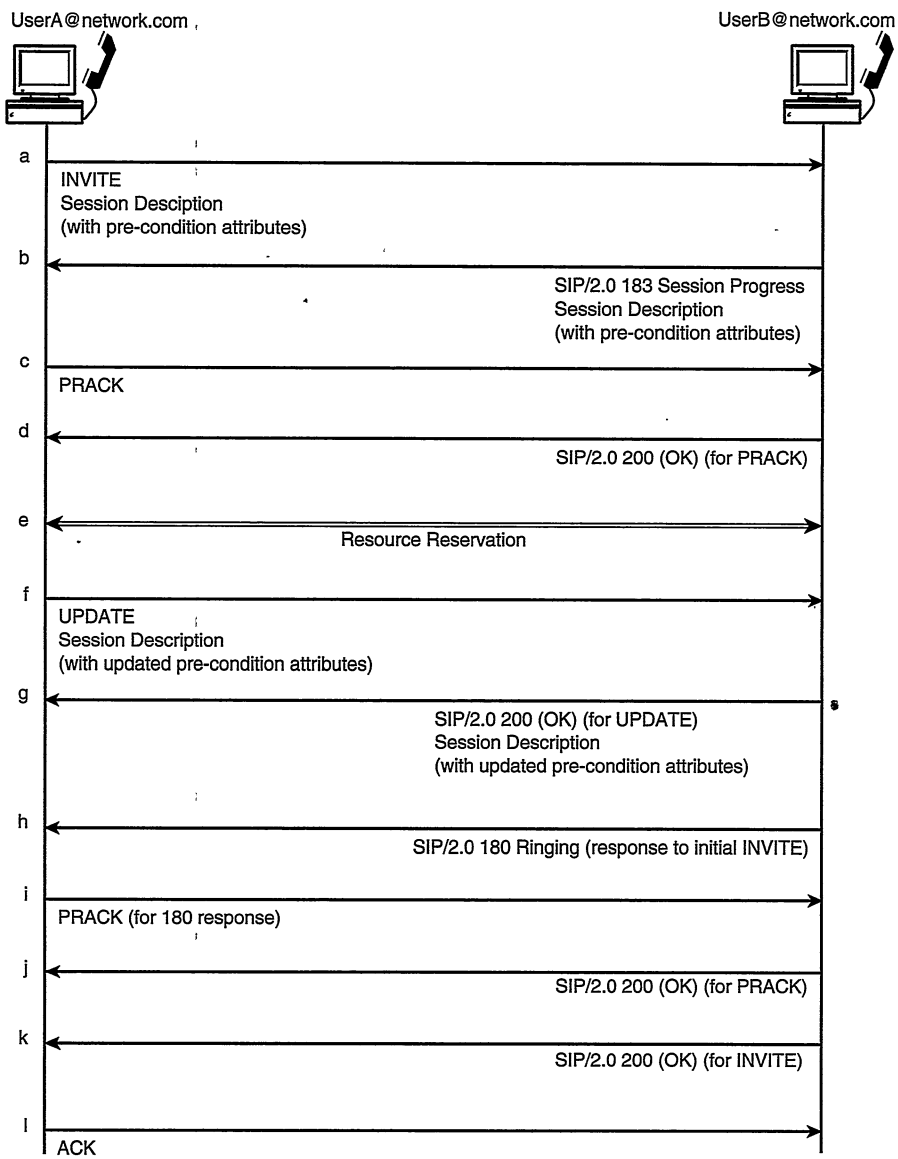


Figure 5-22
Session establishment with end-to-end resource reservation

confirmed. The current status reflects the existing state of affairs for a given resource reservation (which might be that no reservation has yet taken place). The desired status reflects a minimum threshold that must be achieved for a call to proceed. The confirm status indicates that the minimum threshold has been reached.

For each of the status types, we can indicate whether the status relates to the *end-to-end* (e2e) resource reservation, local resource reservation, or remote resource reservation. Recall that resource reservation can be on a session-by-session basis or on an aggregate basis. In the case of per-session resource reservation, resources must be reserved end to end as part of the session establishment. In the case of aggregate reservation, each end must check that it is allowed to access the reserved aggregate resources at its end (local). Moreover, each end must ensure that the far end has also been granted access (remote). In addition to the status type (e2e, local, or remote), we must also specify whether the reservation is for sending media, receiving media, or both. Finally, when requesting that the far end reserve resources, we can use a strength tag to indicate whether the reservation is optional or mandatory.

Example SDP Description for the Initial INVITE Referring again to Figure 5-22, let's look at examples of the media portions of the SDP descriptions that would be associated with the signaling exchange. Here is an example of an SDP description for the initial INVITE:

```
v=0
0=userA 45678 001 IN IP4 stationA.network.com
s=
c = IN IP4 stationA.network.com
t=0 0
m=audio 4444 RTP/AVP 0
a=curr: qos e2e none
a=des: qos mandatory e2e sendrecv
```

This SDP description indicates that User A wants to use G.711 μ-law voice. The description includes QoS-related preconditions, which specify that the current QoS is such that no end-to-end reservation has taken place. The desired precondition is that mandatory end-to-end resource reservation be in place for both the sending and receiving of media.

Example SDP Description for the 183 (Session Progress) Response This description is almost identical to that of the initial INVITE, with the exception of User-B-specific address and port information as well as the addition of the confirmation attribute. The confirmation attribute indicates that User B wants User A to reserve resources in the direction from User A to User B (which is the receive direction from User B's perspective). User B will separately reserve resources in the User-B-to-User-A direction.

```
v=0
0=userB 12345 001 IN IP4 stationB.network.com
```

```
s=
c = IN IP4 stationB.network.com
t=0 0
m=audio 6666 RTP/AVP 0
a=curr: qos e2e none
a=des: qos mandatory e2e sendrecv
a=conf: qos e2e recv
```

Example SDP Description for the UPDATE Request This SDP description indicates that User A has performed his part of the task; resources are reserved in the direction from User A to User B.

```
v=0
0=userA 45678 001 IN IP4 stationA.network.com
s=
c = IN IP4 stationA.network.com
t=0 0
m=audio 4444 RTP/AVP 0
a=curr: qos e2e send
a=des: qos mandatory e2e sendrecv
```

Example SDP Description for the 200 (OK) Response to the UPDATE Request This SDP description indicates that the current QoS status and desired QoS status are the same. In other words, resource reservation has taken place in both directions. The call can now proceed.

```
v=0
0=userB 12345 001 IN IP4 stationB.network.com
s=
c = IN IP4 stationB.network.com
t=0 0
m=audio 6666 RTP/AVP 0
a=curr: qos e2e sendrecv
a=des: qos mandatory e2e sendrecv
```

The foregoing description looks at resource reservation on an end-to-end basis. On the other hand, network resources might have already been reserved on an aggregate basis. In such a case, each end simply needs to verify with a local policing function that it is allowed access to available resources. The approach is to use status types that are segmented (local and remote) as opposed to end to end, which means that each participant deals with network access permission at its own end. The signaling flow for such a scenario would be as shown in Figure 5-23.

In Figure 5-23, User A requests access to network resources in advance of sending the INVITE. Assuming that access permission has been granted, then the INVITE message body indicates that local send and receive

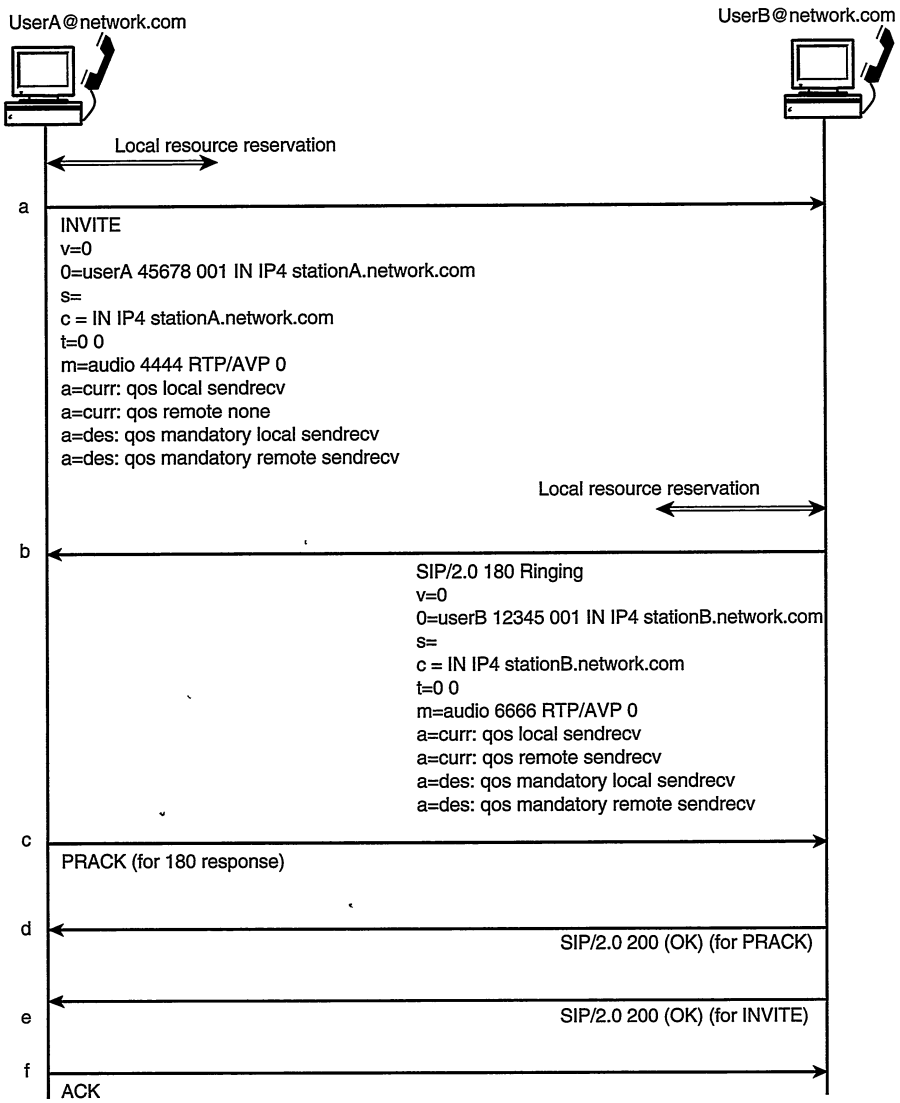


Figure 5-23
Session establishment with local and remote resource verification

resources are available, but that remote resources are not yet known to be available. Upon receipt of the INVITE, User B requests access to resources at the called end. Once access is granted, User B can proceed to alert the called party and returns a 180 (ringing) response to User A. That response contains a message body that indicates that the necessary resources are available at User B's end. The 180 (ringing) response is sent reliably. Since both ends have verified access to network resources, the call can proceed.

In the foregoing descriptions, mandatory strength means that the session cannot continue unless the required resources are definitely available. Optional strength means that the receiver of the session description should reserve resources and proceed with the call, but the resource reservation is not an absolute requirement. Strength of *none* is the initial situation and indicates that no effort to reserve resources has yet taken place. The strength value of *failure* also exists and would be included in a message body in the event that resource reservation did not succeed. This value would be returned in a 580 (precondition failure) response, which is a new response that is included as part of the resource reservation extension to SIP.

Note that, in the event that resources cannot be reserved, the decision not to complete a call might be made only as a last resort, even if the reservation is mandatory. If sufficient resources cannot be used, the option to select a lower-bandwidth codec to reduce the resource requirements is always available. This avenue should be explored before refusing to establish a call. However, it can only be explored if each end supports a lower-bandwidth codec than their first choice. Moreover, a lower-bandwidth codec often means lower speech quality, something that needs to be seriously considered before assuming that a lower-bandwidth codec is really an option.

4.3 Usage of SIP for Features and Services

A number of SIP messaging scenarios have already been described in this chapter, and most of them have described fairly straightforward call scenarios. We have also seen a number of extensions to SIP. In fact, one of the great advantages of SIP is the fact that it enables the easy addition of backwards-compatible extensions. Another great advantage of SIP is the fact that it can be used to provide a number of advanced features and services. For example, we have already seen how the REFER method can be used for a call-transfer application. We have also seen that SIP inherently supports personal mobility through the use of registration. We have seen how a forking proxy can enable a one-number service and we have also seen how the Retry-After: header can be used to support call-completion services. In many ways, these are just the tip of the iceberg when it comes to the features that SIP can offer.

It is possible for SIP messages to carry MIME content as well as an SDP description. Therefore, a response to an INVITE could include a piece of

text, an HTML document, an image, and so on. Equally, because of the fact that a SIP address is a URL, it can easily be included in web content for click-to-call applications. Basically, SIP is well suited to integration with existing IP-based applications and can leverage those applications in the creation of new services. SIP not only supports new and exciting services, but it can also be used to implement the existing supplementary (CLASS) services that exist in traditional telephony today, including call waiting, call forwarding, multiparty calling, call screening, and so on.

In many cases, the signaling for a SIP call will be routed through a proxy. This approach is useful in many ways because it enables the proxy to invoke various types of advanced feature logic. The feature logic may be resident at the proxy or located in a separate feature server or database. In fact, the proxy may also have access to many other functions such as a policy server, an authentication server, and so on, as depicted in Figure 5-24.

The policy server could hold call-routing information and/or QoS-related information, while the feature server would hold subscriber-specific feature data, such as screening lists, forwarding information, and so on. It is perfectly possible for the proxy to use the services of an *Intelligent Network* (IN) *Service Control Point* (SCP), with which communication might be done over the *IN Application Part* (INAP), as described in Chapter 7, "VoIP and SS7," or the network might use the Parlay *Open Service Access* (OSA) approach, utilizing *application programming interfaces* (APIs) between the nodes. Although a distributed architecture has several advantages, it is also

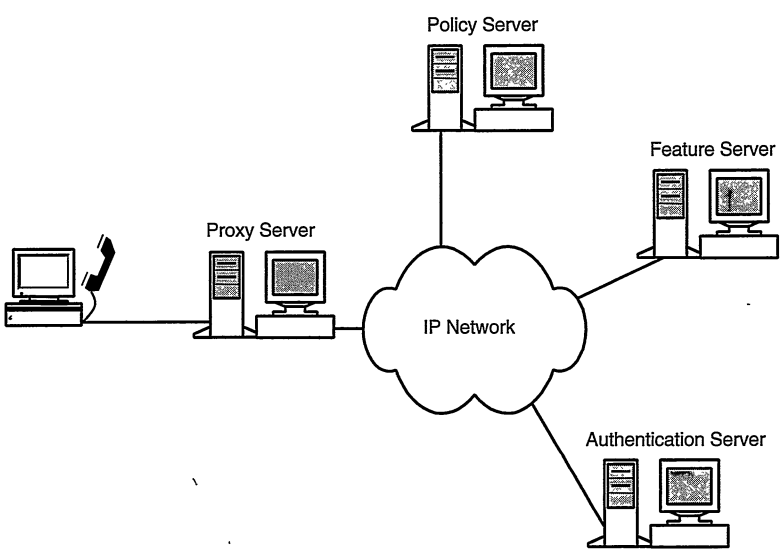


Figure 5-24
SIP service architecture

possible, of course, for the various functions depicted in Figure 5-24 to be integrated in the proxy in one monolithic implementation.

The following section describes just a few examples of calling features using SIP. These are simple examples and are certainly not meant as a complete picture of the features that SIP can enable.

Call Forwarding

Figure 5-25 depicts a call-forwarding-on-busy scenario. Note that, in order to avoid unnecessary clutter, the message body and many of the message headers have been omitted from the figure. These items would, of course, be included in a real signaling implementation.

User 1 calls User 2 via a proxy. User 2 wants calls to be forwarded to User 3 whenever User 2 is busy. Since User 2 happens to be busy, a 486 (busy here) response is returned. Upon receiving the response, the proxy forwards the call to User 3. Note that the INVITE request from the proxy to User 3 contains the same To: field as in the INVITE to User 2. This enables User 3 to recognize that this call is a forwarded call, originally sent to User 2. The 200 (OK) response from User 3 contains a Contact: header, indicating User 3. This enables subsequent messages for this call to pass directly from User 1 to User 3. The proxy could, however, have chosen to insert a Record-route: header in the INVITE. This would ensure that future messages and responses are sent through the proxy.

A call-forwarding-on-no-answer scenario would be quite similar to the call-forwarding-on-busy scenario. The difference is that the INVITE from the proxy to User 2 would timeout. The proxy would then send a CANCEL to User 2 and then forward the call by sending an INVITE to User 3.

The proxy would independently take care of a call-forwarding unconditional service. The proxy would not bother to send an INVITE to User 2 at all. It would simply proceed to send an INVITE to User 3. The To: field of the INVITE to user 3 would still indicate user 2, as is the case for all call-forwarding scenarios.

Consultation Hold

Figure 5-26 depicts a scenario where User B calls User A and, after speaking for a while, User B places User A on hold and places a call to User C. Once the call to User C is finished, User B reopens communication with User A. Again, for the avoidance of clutter, many of the SIP headers have been omitted.

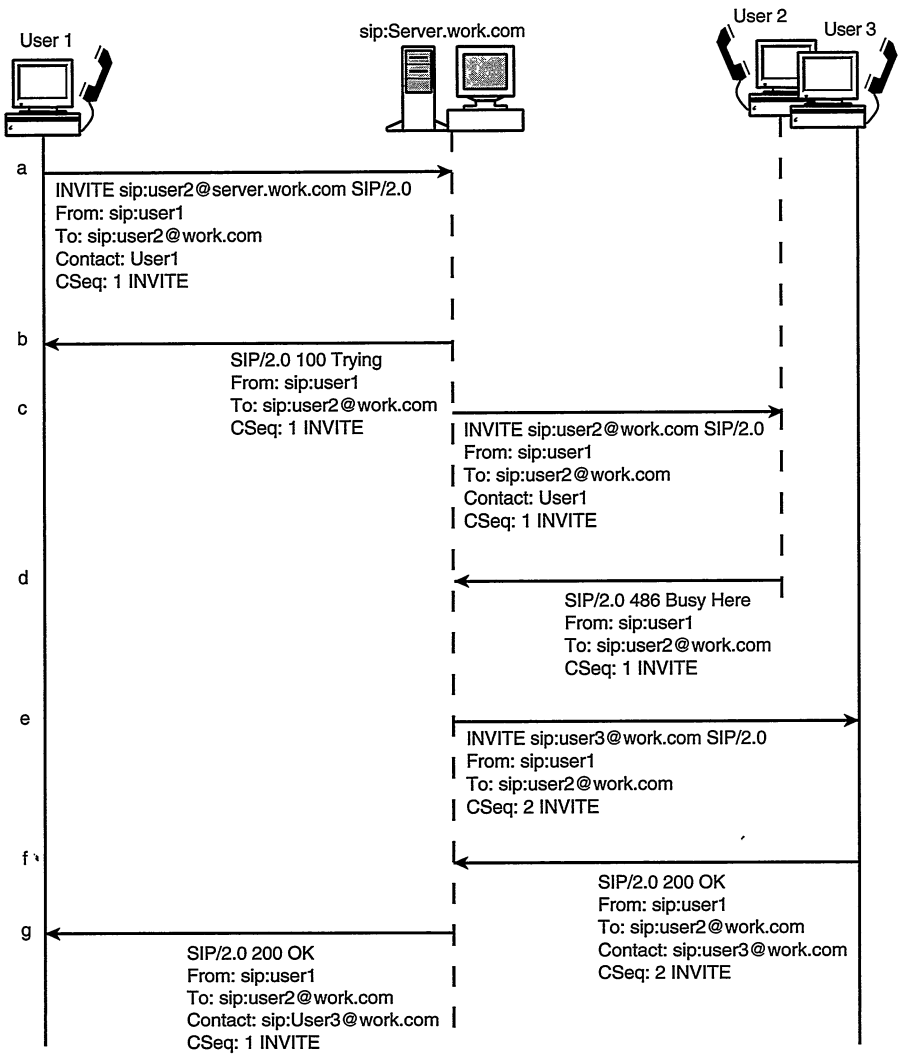


Figure 5-25
Call-forwarding-on-busy scenario

Note the manner in which a call is placed on hold. A SIP UPDATE is issued with the same Call-ID and other parameters as the original INVITE, but the SDP description includes a media attribute that specifies inactive (neither send nor receive). Depending on the situation, the issuer of the UPDATE could specify either inactive or sendonly. When sendonly is used, it means that the receiver of the UPDATE should send audio, but not receive any. The issuer of the UPDATE might also locally mute any received audio stream. The 200 OK response to the UPDATE can also include an updated session description, in accordance with the offer/answer model of SIP.

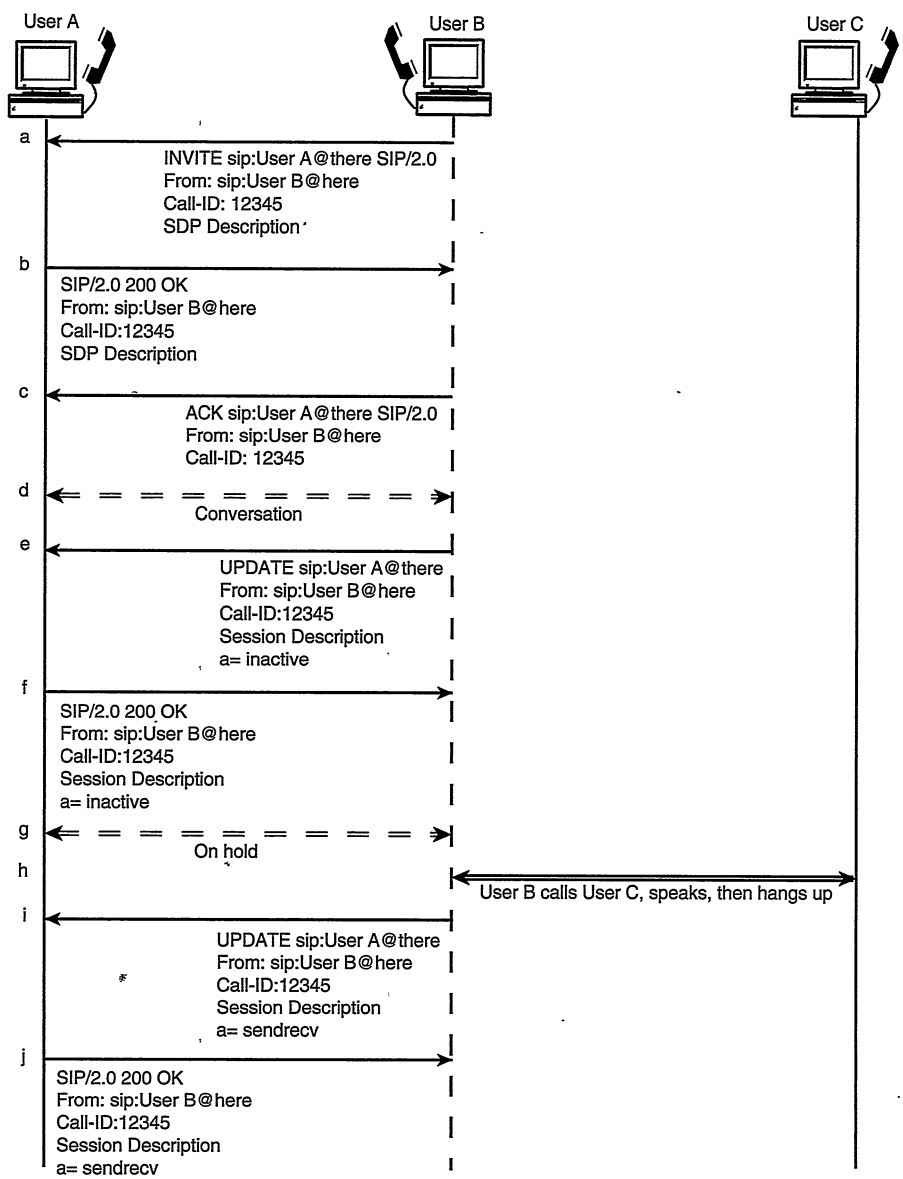


Figure 5-26
Usage of the UPDATE method to place a call on hold

In the example, User B places a call to User C in a normal manner. After that call is finished, User B reopens communication with User A by issuing another UPDATE, again with the same Call-ID as the original call (message i in the figure). The message body in this UPDATE has attribute of sendrecv. Equally, the 200 (OK) response to the UPDATE (message j) has a sendrecv

attribute in the SDP description. The 200 response is acknowledged (not shown) and communication between User A and User B is open again.

A real-life scenario for this type of signaling sequence is one where User A asks User B a question, and User B needs to check with User C for the correct answer. In such a situation, User C might need to talk to User A directly, in which case User B could use the REFER method to transfer the call to User C.

Interworking

Obviously, SIP-based networks will never be the only types of networks in existence. To begin with, circuit-switched networks will continue to be with us for a very long time. Therefore, an obvious need exists for SIP-based networks to be able to interwork with the circuit-switched networks of the PSTN. Furthermore, although SIP is viewed by many as the future of IP telephony, an embedded base of H.323 systems already exists, and more H.323 systems are being deployed. Therefore, a need also exists for SIP-based networks to interwork with H.323-based networks. The SIP specification, however, does not specifically address how such interworking should be achieved.

PSTN Interworking

For interworking with the PSTN, gateways will be required to provide the conversion from circuit-switched media to the packet and vice versa. In fact, we have already seen that the H.323 architecture places great emphasis on the role of gateways. Not only must there be media interworking, but there must also be signaling interworking. After all, SIP is a signaling protocol. If calls are to be established between a SIP-based network and the PSTN, then the SIP network must be able to communicate with the PSTN according to the signaling protocol used in the PSTN. In most cases, this protocol is SS7. For the establishment, maintenance, and teardown of speech calls, the applicable SS7 protocol is the *ISDN User Part* (ISUP).

The usage of gateways and the control of those gateways in a SIP-based network are discussed in some detail in Chapter 6. Furthermore, interworking with the PSTN through the use of signaling gateways is discussed in detail in Chapter 7. For the moment, however, it is sufficient to assume

that a gateway entity exists on the network for interworking between SIP and the PSTN. Let us call that entity a *network gateway* (NGW). In such a case, let us consider a call from Manager (at work) to Collins (at home), where Collins has a standard residential phone connected to a PSTN switch. Such a scenario would appear as in Figure 5-27. Note that most of the fields within the SIP messages and responses are omitted to avoid clutter.

Recall that SIP enables a SIP URI to have the form of a telephone number. The INVITE includes such a URI in the To: field, which is mapped to the calling party number of the *initial address message* (IAM). The From: field of the INVITE could also have a URI in the form of a telephone for the calling party. This would enable the called party to still avail itself of the PSTN Caller-ID service.

The PSTN switch responds with an ACM, which is mapped to the SIP 183 (session progress) response. This response contains a session description and enables in-band information to be returned from the called switch to the SIP caller. Once the call is answered, an ISUP ANM is returned, which is mapped to the 200 (OK) response. This response will have an updated session description, since the session description in the 183 response is considered temporary and is part of an early dialog.

If the call were to be in the opposite direction, from the PSTN to a SIP user, then it would appear as in Figure 5-28. In this case, the 180 ringing response is used instead of the 183 session progress response, but this is for example purposes only. In this example, the 180 ringing response is used by the NGW to trigger a ringing tone towards the calling party locally. Alternatively, the called SIP user agent might return a 183 (session progress) response, with an early media description (such as a ringing tone) to be played to the caller. As is the case in the previous example, not every field is shown for the SIP messages and responses in Figure 5-28. Furthermore, for the avoidance of clutter, the usage of reliable signaling for SIP provisional responses is not shown in either example. It can be assumed that the reliability of these responses would be applied, so that PRACK methods and corresponding 200 (OK) responses would occur.

Although these two interworking examples might seem quite straightforward, seamless interworking between two different protocols is not often quite that easy, because rarely does a one-to-one mapping of messages and parameters take place between one protocol and the other. Sometimes the mapping from protocol A to B means that what is sent in protocol B is only an approximation of what was received in protocol A. If dual interworking is used (from protocol A to protocol B and back to protocol A), what comes out at one end may not be the same as what went in at the other end. Given

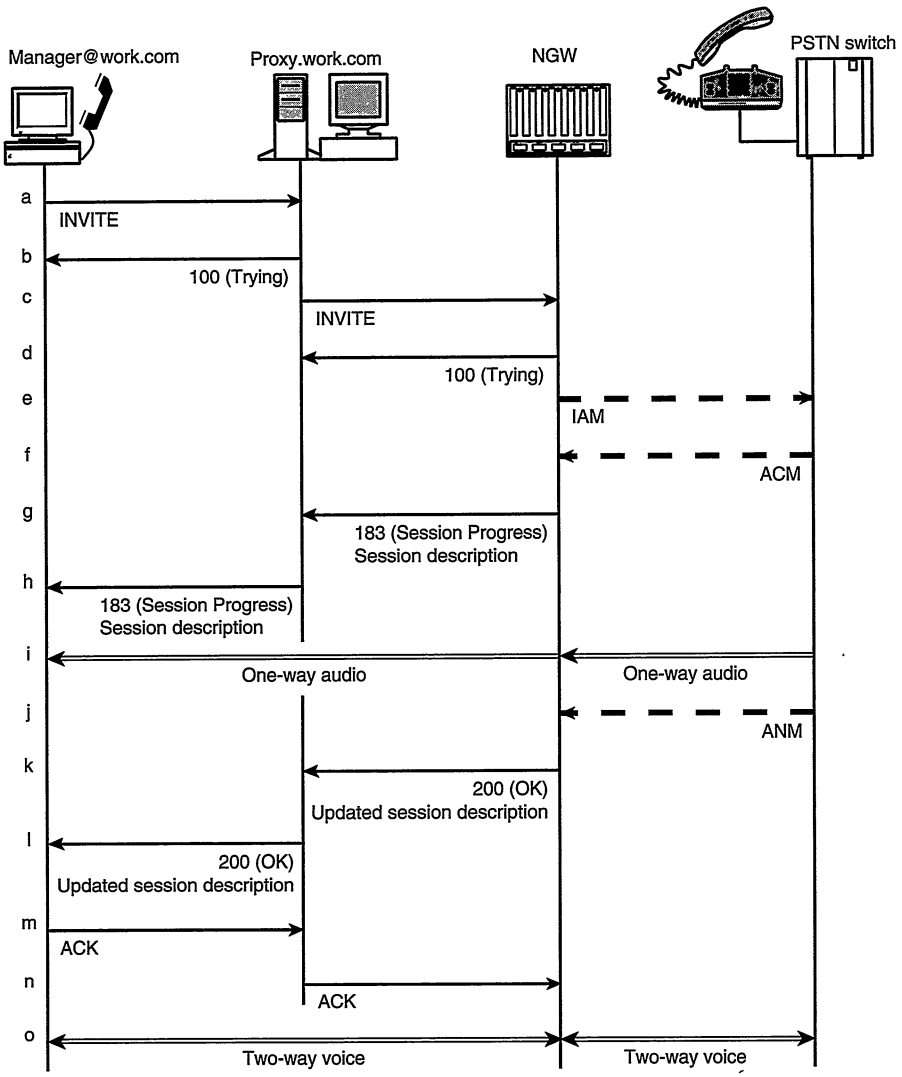
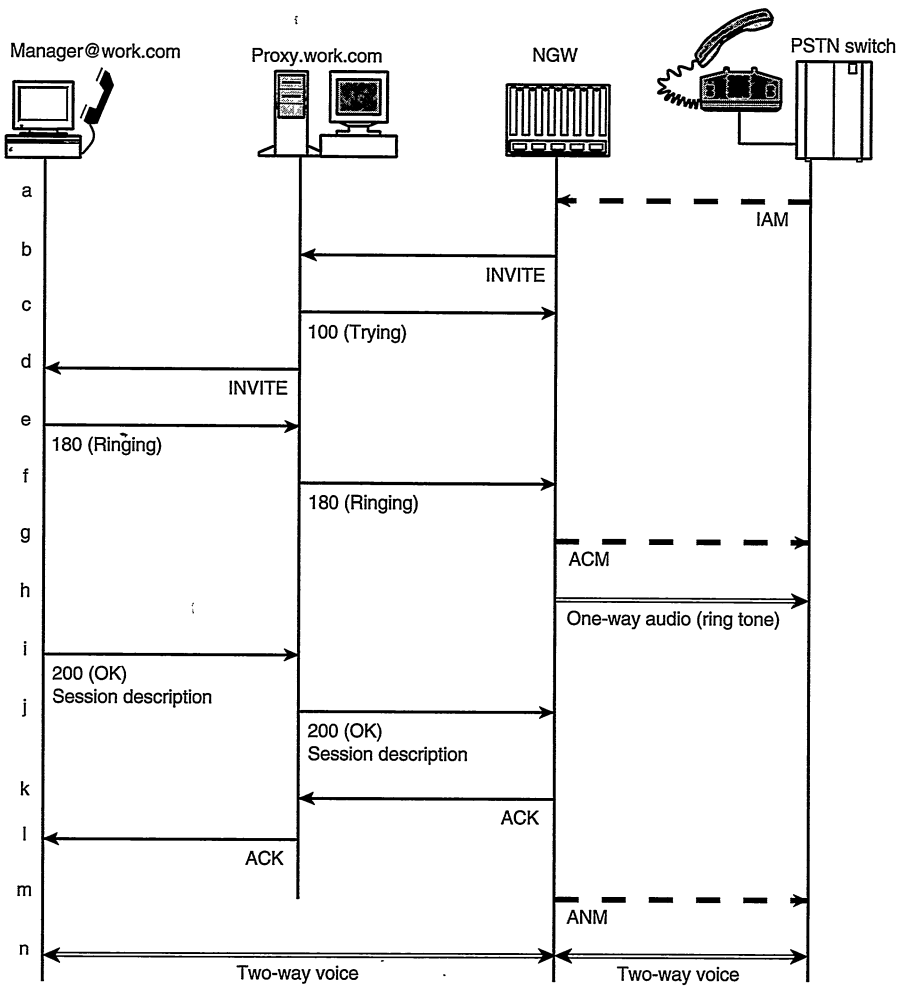


Figure 5-27
SIP to PSTN call

that certain VoIP networks provide a transit service from PSTN at one end to PSTN at the other end, this lack of transparency is a concern.

The solution to this issue is known as *SIP for Telephony* (SIP-T), which is documented in an Internet draft. SIP-T is not so much an extension to SIP, but rather it is a specification of how best to use SIP when interworking with traditional telephony networks. As such, the SIP-T draft is categorized as a *Best Current Practice* (BCP) rather than a potential technical standard.

Figure 5-28
PSTN to SIP call



A significant aspect of SIP-T is the capability to carry ISUP or QSIG message contents within a SIP message body. This capability is described in RFC 3204 (MIME media types for ISUP and QSIG objects). The approach is to carry an ISUP or QSIG message within a SIP message body in hex format. In such a case, the SIP message body could be a multipart message body, including both an SDP session description as one part and an ISUP or QSIG message as another. The SIP INFO method is another component of SIP-T, as it provides for the transfer of midcall information within a SIP session. Such midcall information would typically be generated within the PSTN.

The whole issue of interworking with SS7 is fundamental to the success of VoIP in the real world. The issue is a major concern not just to enable SIP-based networks to interwork seamlessly with the PSTN for regular phone calls, but also to enable SIP-based systems to avail themselves of services in the PSTN that rely on SS7 signaling, including IN services such as toll-free calling. Therefore, Chapter 7 of this book is devoted to the topic and addresses the transit issue just described as well as many other aspects.

Interworking with H.323

Interworking with H.323 is also a major consideration, particularly because H.323 is already widely deployed. Several Internet drafts have been prepared to address the issue. One in particular, titled "SIP-H.323 Interworking Requirements," is under consideration by the IESG as a potential informational RFC.

The proposed approach for interworking between a SIP network and an H.323 network involves the use of a gateway, as depicted in Figure 5-29. Depending on the function to be performed, the gateway may appear to the SIP network as a user-agent client or user-agent server. To the H.323 network, the gateway will appear as an H.323 endpoint. On the SIP side, the gateway might also include a SIP registrar function and/or proxy function, and on the H.323 side, it could contain a gatekeeper function.

Figure 5-30 depicts an example scenario of a call from a SIP caller to an H.323 terminal. The H.323 Fast Connect procedure is used, and we assume that a gatekeeper is not involved in the call. If a gatekeeper were to be used, then the RAS signaling would occur on the H.323 side of the gateway and would not be seen by the SIP network in any case. The SDP information in the SIP INVITE is mapped to the logical channel information included in

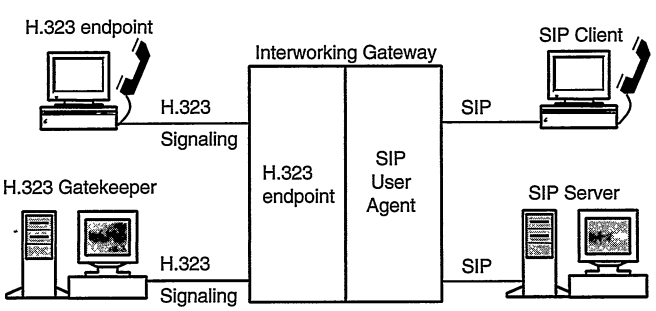


Figure 5-29 SIP-H.323 interworking gateway

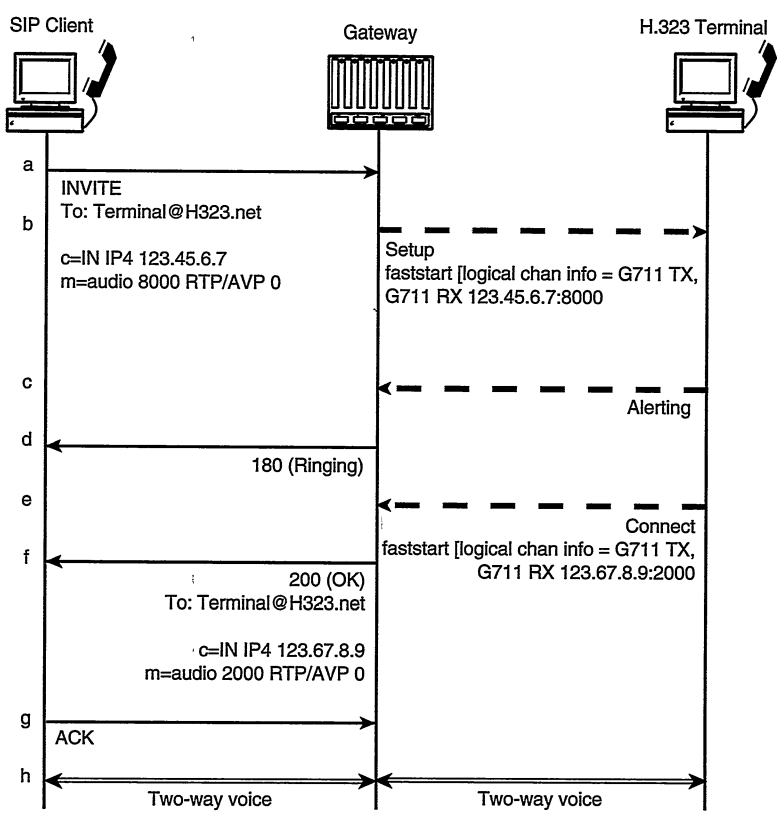


Figure 5-30
A call from SIP to H.323 using H.323 Fast Connect

the faststart element of the Q.931 SETUP message from the gateway to the H.323 terminal. In the example, a faststart element is conveniently returned from the called terminal in the Connect message rather than the Alerting. In this case, the logical channel information within the Connect message can be mapped directly to the session description within the SIP 200 (OK) response.

If the faststart element had been included in the Alerting message instead or some other messages (such as Call Proceeding), then the gateway would simply have had to store the information until it was ready to send it to the SIP client in the 200 (OK) response. Note that the text included with the messages in the figure is for explanatory purposes only and does not match the actual syntax of the messages as they should appear in a real implementation.

If we assume that the called H.323 terminal network does not support the Fast Connect procedure, then the sequence becomes more long-winded,

as shown in Figure 5-31. In this case, the gateway supports Fast Connect, but the called terminal does not. Thus, the Setup message includes the fast-start element, but the messages returned from the called H.323 terminal do not. Therefore, upon receipt of the Connect message, the gateway must perform H.245 logical channel signaling to and from the called terminal before returning the 200 (OK) to the calling SIP client. The H.245 messaging includes capability exchange and the establishment of logical channels. For

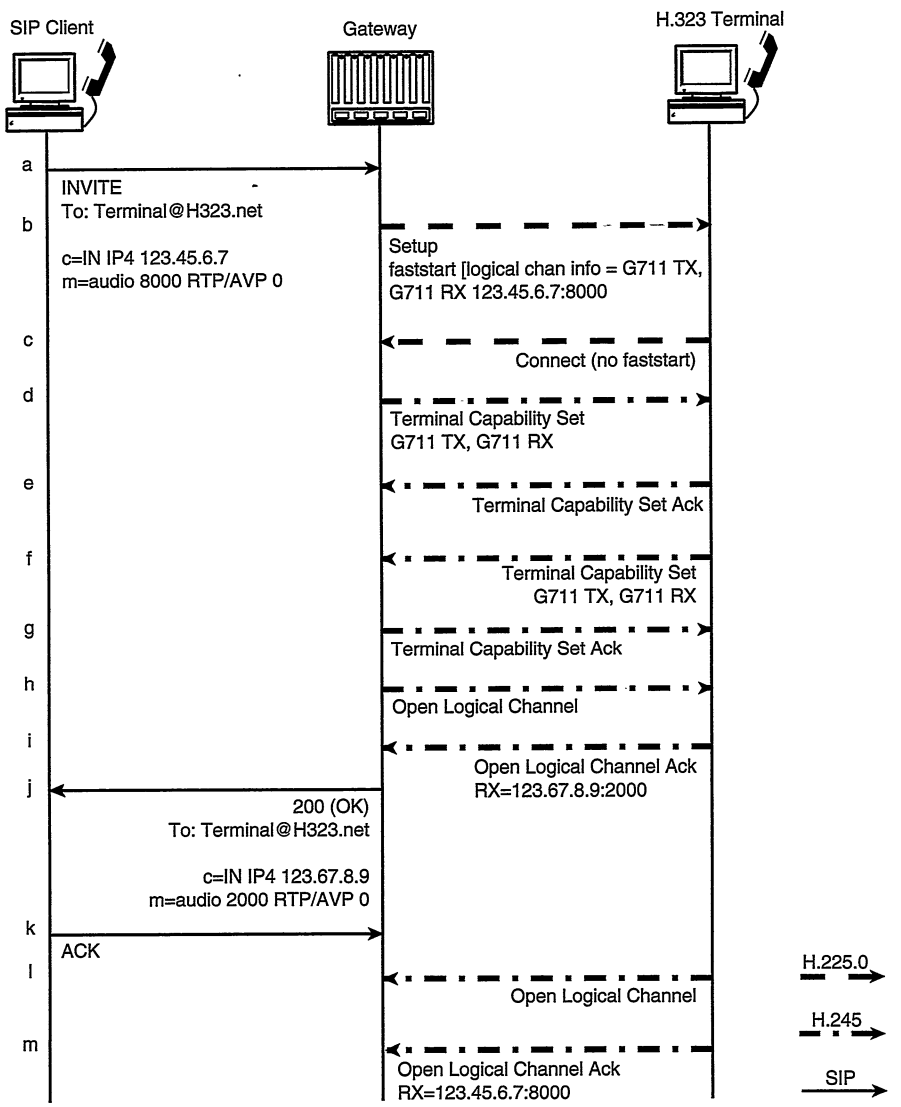


Figure 5-31
A call from SIP to H.323 without H.323 Fast Connect at the destination

clarity, interim responses from the called terminal (Call Progress, Alerting, and so on) are not shown in the figure.

A call in the opposite direction, from H.323 to SIP, is relatively straightforward if the H.323 terminal and the gateway both support the H.323 Fast Connect procedure. Such a scenario is depicted in Figure 5-32. In this case, the session information is included in the H.225/Q.931 SETUP message and this can be mapped to the SDP information within the SIP INVITE. Equally, the session description accompanying the 200 (OK) response from the called SIP user agent can be mapped to the information within the faststart element of the Connect message back to the H.323 terminal.

If, on the other hand, the calling H.323 terminal does not support the Fast Connect procedure, then the interworking becomes more complex for two reasons. First, the fact that Fast Connect is not used means that separate H.245 logical-channel signaling is required on the H.323 side of the gateway. Second, no logical channel information is carried within the Setup message. Therefore, nothing can be mapped to a session description in the SIP INVITE.

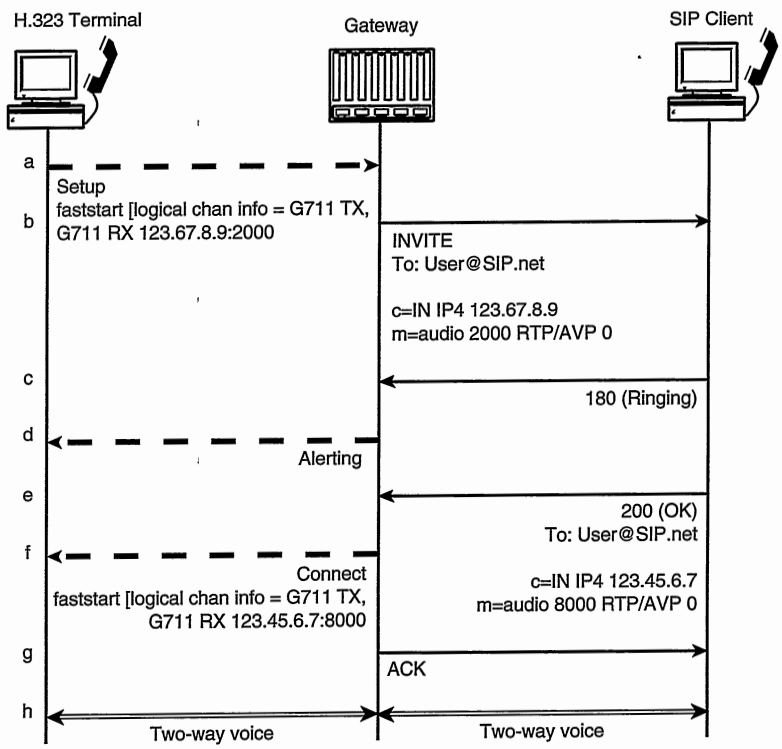


Figure 5-32
A call from H.323 to SIP using H.323 Fast Connect

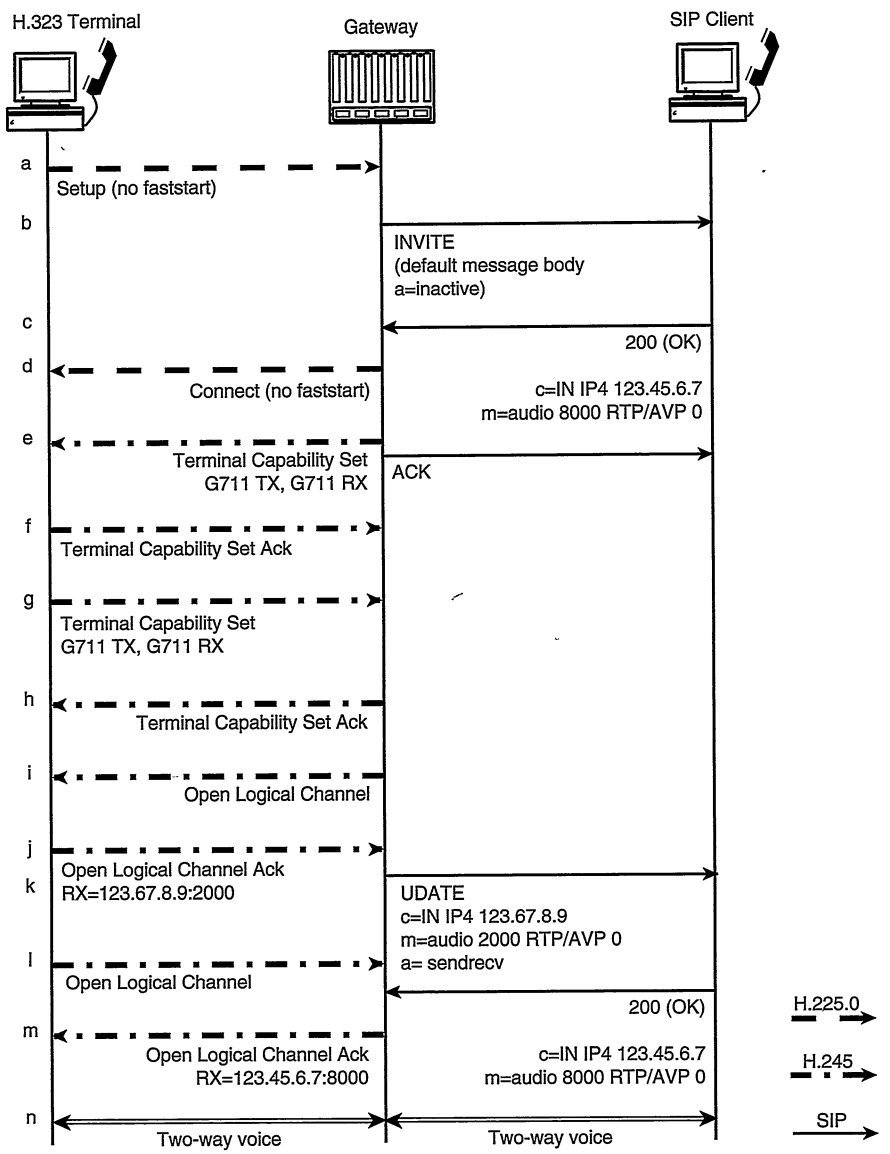


Figure 5-33
A call from H.323 to SIP without H.323 Fast Connect

In that event, the INVITE could contain a default session description. The IP address in the connection (c=) line of the session description could contain a default value, while the media (m) line would indicate a port number of 0 and an attribute of inactive, as shown in Figure 5-33. Once the appropriate H.245 signaling has taken place between the gateway and the

H.323 terminal, the gateway can issue a SIP UPDATE to the SIP client, including a revised session description.

Summary

Although quite new, SIP has already established itself as the way of the future for signaling in VoIP networks. Its strengths include the fact that it is far simpler than H.323 and is therefore easier to implement. Its simplicity, however, belies its flexibility and power. It can support all the features that are already supported by traditional telephony networks and more besides. In fact, SIP is the protocol of choice for the evolution of third-generation wireless networks, where it is expected that SIP-based mobile devices will become available and SIP-based network elements will be introduced within mobile networks.

As we shall see in the next chapter, SIP also fits very well with those protocols used for media gateway control and, as such, forms part of the overall architecture known as softswitch.

Motorola - Exhibit 1022 - Page 266

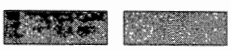
CHAPTER 6

Media Gateway Control and the Softswitch Architecture

Motorola - Exhibit 1022 - Page 268

Previous chapters described some of the advantages that *Voice over IP* (VoIP) can offer over traditional circuit-switched telephony, such as lower cost of network implementation, integration of voice and data applications, new features, and potentially reduced bandwidth for voice calls. It would be nice if all telephony were to be carried over the *Internet Protocol* (IP) so that these advantages could be made available globally. Unfortunately, that day will not arrive for a long time. Replacing all traditional circuit-switched networks is not feasible, and perhaps not even desirable. Among other reasons, the cost would simply be exorbitant. Rather, one can expect a gradual evolution from circuit-switched to IP-based networks such that both types of networks will exist side by side for a very long time.

Given that VoIP networks and traditional circuit-switched networks will have to coexist for many years, they will need to interwork as seamlessly as possible. In other words, users of existing circuit-switched systems should be able to place calls to VoIP users, and vice versa. In each case, the user should not have to do anything radically different from simply calling another user on the same type of network. The challenge therefore is to develop solutions that make such seamless interworking possible.



Separation of Media and Call Control

Previous chapters discussed the interworking between VoIP networks and circuit-switched networks through the use of gateways. The primary function of such gateways is to make the VoIP network appear to the circuit-switched network as a native circuit-switched system and vice versa. In other words, VoIP networks must accept signaling and media from circuit-switched networks in native format and convert both to the format used within the IP network.

We also saw in Chapter 4, "H.323," that the signaling involved in a VoIP call can take a very different path from the media path itself. The media might go directly from end to end, while the signaling might pass through one or more intermediate entities, such as gatekeepers or *Session Initiation Protocol* (SIP) proxies. In fact, the logical separation of signaling and media is not a new concept; it is something that has existed for many years in the *Signaling System 7* (SS7) architecture.

Putting these two considerations together means that a network gateway has two related but separate functions: signaling conversion and media

conversion. The signaling aspect is coupled to whatever call-control functions are operating on the network. In fact, signaling can be considered the language that call-control entities use to communicate. The media conversion and transport can be considered a slave function, invoked and manipulated to meet the needs dictated by call control and signaling.

In terms of network architecture, this leads to a configuration of the kind depicted in Figure 6-1. This figure shows a scenario where a VoIP network sits between two external networks, as might be the case for a VoIP network that provides long-distance service. Figure 6-1 illustrates the separation of call control and signaling from the media path. Moreover, it illustrates the master-slave relationship between call control and media, even within the network gateway itself. Even if the gateway is just a single box handling both signaling and media, some internal control system will be used between the call-control functions and the media-handling functions.

Given that such a logical separation exists, there is no compelling reason why the separation could not also be made physical. In fact, such a physical separation has some significant advantages. First, a physical separation enables media conversion to take place as close as possible to the traffic source or sink within the circuit-switched network while centralizing the call-handling functions. Second, a smaller number of gateway controllers or call agents located more centrally can control multiple gateways placed in various locations. Such a network architecture leads to another benefit, which is that new features could be rolled out more quickly as they would need to be implemented only in the centralized call-control nodes, rather than at every node in the network. All of this leads us to the type of architecture shown in Figure 6-2.

Of course, the physical separation that could bring about these benefits requires a well-standardized control protocol between the *media gateway*

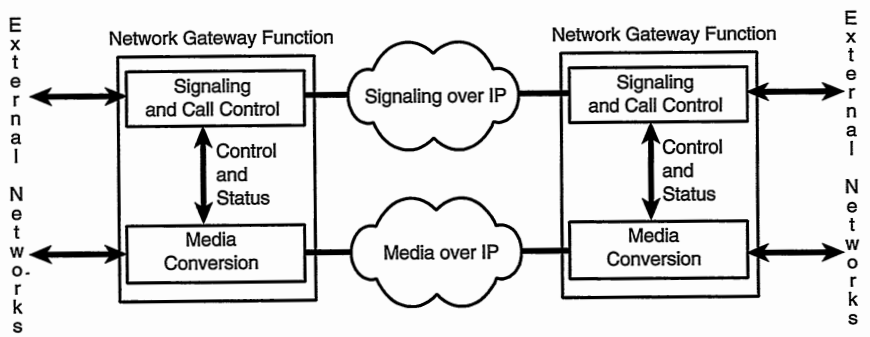


Figure 6-1 VoIP network gateways in tandem between external networks

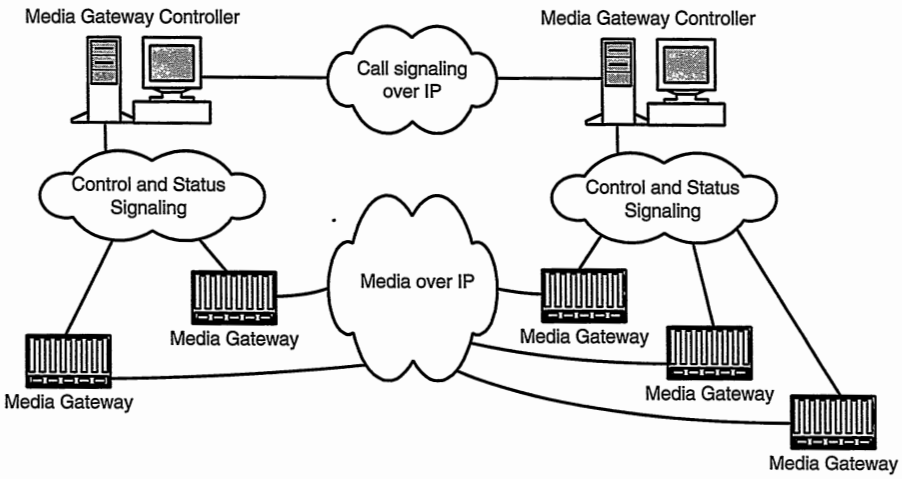


Figure 6-2
Distributed system architecture

controller (MGC) or the call agent and *media gateway* (MG). Numerous such standardized protocols have been developed. The two most common protocols are the *Media Gateway Control Protocol* (MGCP or MEGACO) and MEGACO/H.248. By using one or another of these protocols for media gateway control and SIP as the call-signaling protocol, the architecture shown in Figure 6-2 is a reality and is known as the *softswitch architecture*.

Softswitch Architecture

The softswitch architecture involves the separation of the media path and media-conversion functions from the call control and signaling functions, as depicted in Figure 6-2. The entities that handle call control are known as call agents or MGCs, while the entities that perform the media conversion are known as MGs. The name softswitch is used because many of the switching functions traditionally handled by large monolithic systems in the circuit-switched world are instead emulated by software systems. In fact, in many quarters, the term softswitch is used to refer to a call agent or MGC as opposed to the overall architecture.

The softswitch architecture has many supporters within the VoIP industry. In fact, many of the leaders in the industry have come together in an organization known as the *International Softswitch Consortium*. This organization promotes the softswitch concept and related technologies, it allows

Motorola - Exhibit 1022 - Page 271

interested parties to share ideas and cooperate in the ongoing development and deployment of the technology, and it facilitates interwork testing between different implementations.

One of the reasons why the softswitch approach is so popular is the fact that it is a distributed architecture. For a network operator, it is possible to use different network components from different vendors, such that the best in each class may be chosen in each area. For equipment vendors, it is possible to focus efforts on one area and not have to develop or acquire expertise in all areas. For example, a company can focus on the development of an MGC without having expertise in the voice-coding techniques and related *digital signal processors* (DSPs) used within MGs. Equally, a company that does have the necessary voice-coding expertise can develop an MG without having the call management expertise needed to develop a call agent. Another distinct advantage is the rapid rollout of new features described previously.

In the softswitch architecture, SIP is often used as the signaling protocol between the MGCs, so that an MGC appears as a SIP user agent in the SIP network. Therefore, when interworking between SIP and the *Public Switched Telephone Network* (PSTN), the MGC handles the functional interworking between SIP and PSTN signaling such as *ISDN User Part* (ISUP), while the gateways take care of the media conversion functions, as shown in Figure 6-3.

As mentioned, two popular protocols exist for media gateway control. MGCP was developed within the *Internet Engineering Task Force* (IETF) and MEGACO/H.248 has been developed as a cooperative effort between

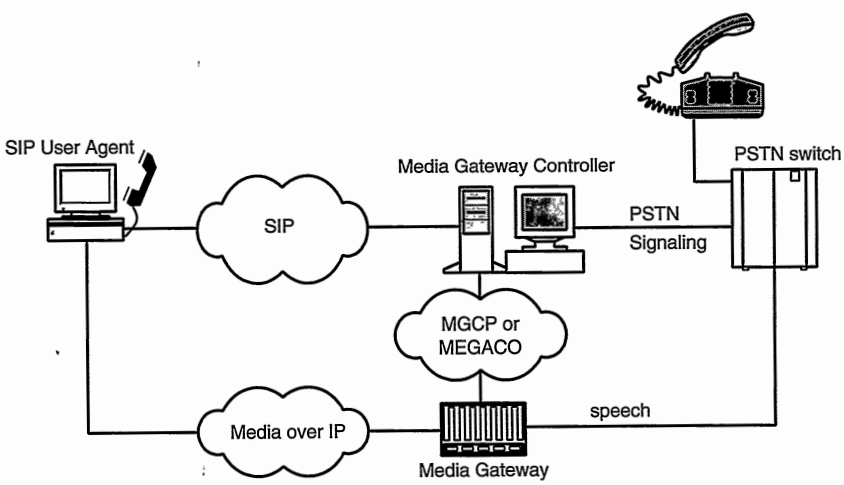


Figure 6-3
Softswitch/PSTN interworking

the IETF and the *International Telecommunication Union Telecommunications Standardization Sector* (ITU-T) Study Group 16. This reason is why MEGACO/H.248 has such a strange name. Within the IETF, it is known as MEGACO (for media gateway control protocol) and within the ITU it is known as H.248. Although MGCP is still popular in many implementations, MEGACO/H.248 is the media gateway control protocol of the future, and can be considered the successor to MGCP. Most new implementations use MEGACO/H.248.

Requirements for Media Gateway Control

Before delving deeply into the details of MGCP or MEGACO, it is first worth considering the functions that the protocols should support and what requirements exist for interworking with other protocols. This background information will clarify the reasons behind the protocols in the first place.

RFC 2805, "Media Gateway Control Protocol Architecture and Requirements," is an informational *Request for Comments* (RFC), describing the functions to be supported by a distributed gateway system, as depicted in Figure 6-3. Although the focus in this chapter is interworking between IP-based networks and circuit-switched networks, the document addresses generic requirements to be met for interworking with any type of network, not just circuit-switched networks. At a very high level, the requirements for the control protocol are that it should

- Enable the creation, modification, and deletion of media streams across an MG, including the capability to negotiate the media formats to be used.
- Allow the specification of the transformations to be applied to media streams as they pass through an MG.
- Allow the MGC to request and the MG to report the occurrence of specified events within the media stream (such as *Dual Tone Multifrequency* [DTMF] tones or call-associated signaling), and allow the MGC to specify the actions that should be automatically taken by the MG when such events are detected.
- Allow the MGC to request that the MG apply tones or announcements either by direct instruction or upon the occurrence of certain events, such as applying a dial tone to an analog line when an off-hook event is detected.

Motorola - Exhibit 1022 - Page 273

- Enable the establishment of media streams according to certain *quality of service* (QoS) requirements and to change QoS during a call.
- Enable the reporting of QoS statistics from an MG to an MGC. Since the MGC is not involved in the media transfer, it has no visibility of quality considerations such as delay, jitter, and packet loss and will need to be informed for network management purposes.
- Enable the reporting of billing/accounting information such as call start and stop times, the volume of content carried by media flow (such as the number of packets/bytes), and other statistics.
- Support the management of associations between an MG and an MGC, such as the capability for a given gateway to be controlled by a different MGC in the case of failure of a primary MGC.
- Support a flexible and scalable architecture, whereby different MGs with different capacities and different interfaces may be controlled by an MGC.
- Facilitate the independent upgrade of MGs and MGCs, including the use of a version indicator in message exchanges.

Protocols for Media Gateway Control

As mentioned earlier, several protocols have been written for media gateway control. Of these, two have been widely embraced. The first of these is MGCP (RFC 2705). This is an interesting document in that it is purely an informational RFC (rather than a standards-track protocol), even though it includes a protocol specification. The reason why it is categorized as informational is because it is being succeeded by MEGACO/H.248. However, work on MEGACO started much later than MGCP. Even while MGCP was still an Internet draft, many systems developers proceeded to include it within their product development rather than wait for MEGACO. Therefore, the decision was taken to release the MGCP specification as an informational RFC.

RFC 2885 was the first release of MEGACO/H.248 as a standards-track protocol. It was subsequently discovered that the protocol had some errors, and a separate RFC (RFC 2886) was prepared to document the errata and propose appropriate corrections. Subsequently, the original RFC was updated, and a new RFC (RFC 3015) was published and included the necessary corrections to RFC 2885. Thus, RFC 3015 is now the official version

of MEGACO version 1. As of this writing, version 2 of the MEGACO specification is being prepared at the IETF and ITU. The second version adds some new capabilities and modifies some aspects of the version 1 specification.

Although MEGACO is clearly the way forward for media gateway control, many systems developers continue to implement MGCP within their systems, and MGCP specification enhancement work continues at the IETF. In this chapter the description of MGCP is based on the latest draft update to RFC 2705. Given that MGCP and MEGACO are both popular, we describe both in this chapter.

MGCP

MGCP is a master-slave protocol in which controllers, known as call agents, control the operation of MGs. The call control intelligence and related call signaling is taken care of by the call agent, while the MG takes instruction from a call agent and basically does what the call agent commands. The commands from the call agent generally relate to the establishment of connections and the teardown of connections from one side of the gateway to another. In most cases, the call agent tells the MG to make a connection from a line or trunk on the circuit-switched side of the gateway to a *Real-Time Transport Protocol* (RTP) port on the IP side of the gateway, as depicted in Figure 6-4.

MGCP only addresses the communication between a call agent and an MG and does not address communication from one call agent to another.

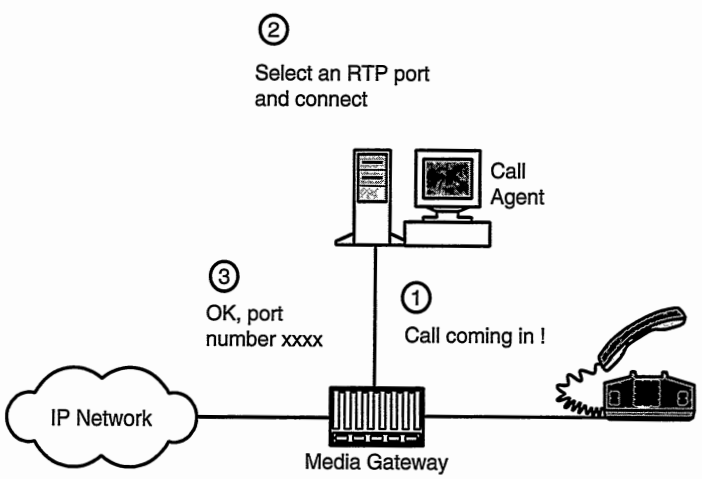


Figure 6-4
The basic approach to MGCP connection establishment

Since instances will occur where media is to be exchanged between gateways controlled by different call agents, signaling communication will be needed between call agents. MGCP assumes that such a signaling protocol exists, but does not specify what it is. In reality, the protocol between call agents is likely to be SIP.

The MGCP Model

MGCP describes a model containing endpoints and connections. Endpoints are sources or sinks of media and include such elements as trunk interfaces or *plain old telephone service* (POTS) line interfaces. Endpoints exist within MGs, and depending on the type of endpoint, it might or might not have one or more external channel or line interfaces. For example, an analog line endpoint would be connected to a physical analog line, which would in turn connect to an analog telephone. On the other hand, endpoints also exist that have no external connections. One example is an announcement endpoint that would reside internally in the gateway. Other entities in the IP network would connect to it for the purpose of retrieving an announcement.

Connections are the allocation of IP resources to an endpoint such that the endpoint may be accessed from the IP network. For example, when a call is to be established across a gateway from a circuit-switched line to an IP interface, an ad hoc relationship is established between the circuit-switched line and an RTP port on the IP side. This relationship is called a connection, and a single endpoint can have several connections. A general representation of endpoints and connections is provided in Figure 6-5.

MGCP Endpoints

MGCP describes a number of different endpoints as follows:

- **DS0 channel** A digital channel operating at 64 Kbps. Such a channel will typically be multiplexed within a larger transmission facility such as a DS1 (1.544 Mbps) or E1 (2.048 Mbps). In most cases,

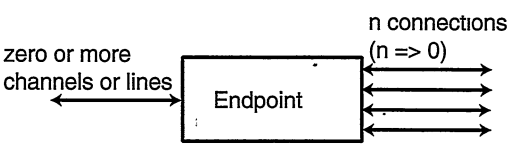


Figure 6-5 A general representation of an endpoint in MGCP

the DS0 will carry voice coded according to either G.711 μ-law or G.711 A-law. In some cases, the DS0 will carry signaling instead of voice, such as an *Integrated Services Digital Network* (ISDN) D-channel, in which case it is necessary that the signaling information received be passed from the MG to the call agent for processing.

- **Analog line** Typically, an endpoint that interfaces to a standard telephone line and provides service to a traditional telephone. In most cases, the media to be received on the analog line will be an analog voice stream, though it could also be audio-encoded data from a modem, in which case the gateway shall be required to extract the data and forward it as IP packets.
- **Announcement server access point** An endpoint providing access to a single announcement. Normally, a connection to this type of endpoint will be one-way, since the requirement is for media to be sent from the announcement rather than to it. We assume that the announcement server is within the IP network and IP-based connections are to be made to it. Thus, the endpoint does not have any external circuit-switched channels or lines associated with it.
- ***Interactive Voice Response* (IVR) access point** An endpoint providing access to an IVR system, where prompts or announcements may be played, and responses from the listener may be acted upon.
- **Conference bridge access point** An endpoint where media streams from multiple callers may be mixed and provided back to some or all of the callers.
- **Packet relay** A specific form of conference bridge supporting only two connections. This type of endpoint will use packet transport on both connections. A typical example would be a firewall between an open and a protected network, where the media stream must pass through such a relay point rather than directly from end to end.
- **Wiretap access point** A point from which a connection is made to another endpoint for the purposes of listening to the media transmitted or received at that endpoint. Connections to a wiretap access point will be one-way only.
- **ATM trunk-side interface** An endpoint corresponding to the termination of an *Asynchronous Transfer Mode* (ATM) trunk, where such a trunk might be an ATM virtual circuit. Such an endpoint might be used in a gateway that provides an interface between a VoIP network on one side and a Voice over ATM network on the other.

Each endpoint is identified with an endpoint identifier. This identifier will be comprised of the domain name of the gateway to which it belongs and a local name within the gateway. The local name within the gateway will obviously depend on the gateway in question, but it will generally have a hierarchical form such as X/Y or X/Y/Z, where Y is an entity within X, and Z is an entity within Y. An example would be a gateway that supports multiple channelized T3 interfaces. If one wanted to identify a single DS0 channel on the gateway, then one would have to identify the particular T3 (X), the particular DS1 within the T3 (Y), and the particular DS0 (Z) within that DS1. If we wanted to identify DS0 number 7 within DS1 number 12 on DS3 number 4, then the identifier could be something like trunk4/12/7@gateway.somenetwork.net.

We can issue a wildcard to certain components of the identifier using either $ (any) or * (all). Thus, if we wanted to refer to any DS0 within the fifth DS1 within the first T3 in our example, then we could do so through the term trunk1/5/$@gateway.somenetwork.ne. This type of wildcarding is useful when a call agent wants to create a connection on an endpoint in a gateway and does not really care which endpoint is used. Through the use of the $ wildcard, the choice of a particular endpoint can be left to the gateway itself. Through the use of the * wildcard, the call agent can instruct the gateway to perform some action related to a number of endpoints, such as a situation where the call agent requests statistical information related to all endpoints on a gateway.

MGCP Calls and Connections

A connection is the relationship established between a given endpoint and an RTP/IP session. Consider, for example, a DS0 endpoint. If the DS0 is idle, then no connection is associated with the endpoint and no resources are allocated to the endpoint on the IP side of the gateway. When the DS0 is to be used to carry voice traffic, however, then IP resources are allocated to the endpoint, and a link is created from the DS0 on the line side of the gateway to an IP session on the IP side of the gateway. In MGCP terminology, this link is called a connection. If two endpoints are involved in a call, then two connections are used.

A call is a grouping of connections such that the endpoints associated with those connections may send media to each other and/or receive media from each other. Figure 6-6 shows a scenario where a call is established between two analog phones, such that they communicate across an IP

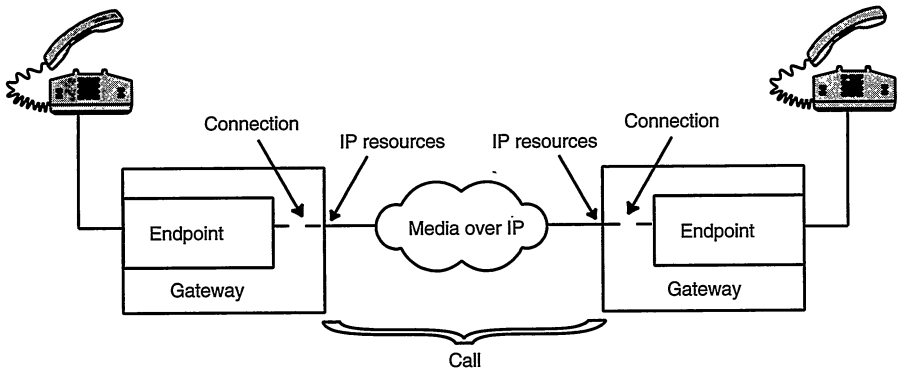


Figure 6-6
The MGCP relationship between calls and connections

network using two gateways. On each gateway, a connection is created between an endpoint on the line side of the gateway and IP resources (such as the IP address and RTP port number) on the IP side of the gateway. Once a connection is created on each of the two endpoints, then the call can be considered the linking of these two connections such that media may be passed between the endpoints.

When a gateway creates a connection for an endpoint and allocates IP resources, those resources are described in terms of a *Session Description Protocol* (SDP) session description. The establishment of a call involves the endpoints sharing their respective session descriptions with each other so that they may exchange media streams. The primary function of MGCP is to enable those connections to be created and to enable the session descriptions to be exchanged. MGCP does so through the use of commands and responses between call agents and MGs.

Overview of MGCP Commands

MGCP defines nine commands, some of which are sent from call agent to gateway and some from gateway to call agent. MGCP commands are composed of a command line, a number of parameter lines, and, optionally, a session description. The command line and parameters are text, using the US-ASCII character set. The session description uses SDP, also a text-based protocol.

The command line and the various parameters are separated from each other by a *carriage return and line feed character* (CRLF). The session description is separated from the command line and the various parameter lines by a single empty line, similar to the way things are done in SIP.

The command line is composed of the requested verb (the name of the command), a transaction identifier, the endpoint for which the command applies (or from which the command has been issued), and the protocol version (MGCP 1.0). These four items are separated from each other by a *space* (SP) character. Thus, the command appears as

```
CommandVerb SP TransactionID SP EndpointID SP MGCP 1.0
```

MGCP defines the nine commands in the following list. For each command, the requested verb is coded as a four-letter uppercase ASCII string:

- **EndpointConfiguration (EPCF)** Issued by a call agent to a gateway to inform the gateway about bearer/coding characteristics on the line side of one or more endpoints. The only characteristic currently specified is whether the encoding on the line side is A-law or μ-law.
- **CreateConnection (CRCX)** Issued by a call agent to a gateway and used for the creation of a connection at an endpoint.
- **ModifyConnection (MDCX)** Issued by a call agent to a gateway and used to change some of the characteristics of an existing connection. One of the primary functions is to provide information to the endpoint regarding the far end of the call.
- **DeleteConnection (DLCX)** Issued by a call agent to a gateway to instruct the gateway to terminate a connection (or multiple connections). The command can also be issued by a gateway to a call agent (for example, when the gateway has detected that line-side connectivity has been lost).
- **NotificationRequest (RQNT)** Issued by a call agent to a gateway to request that the gateway notify the call agent when certain events occur. The gateway could be instructed, for example, to detect an off-hook signal from an analog line.
- **Notify (NTFY)** Issued by a gateway to a call agent to report the occurrence of certain events.
- **AuditEndpoint (AUEP)** Issued by a call agent to a gateway to query the status of a given endpoint and/or other data regarding the endpoint. The gateway will respond with the requested information.
- **AuditConnection (AUCX)** Issued by a call agent to a gateway to query information regarding a specific connection.
- **RestartInProgress (RSIP)** Issued by a gateway to a call agent to inform the call agent when one or more endpoints are taken in or out of service.

Command Encapsulation MGCP supports the concept of encapsulation, where one command can be included within another. For example, when instructing a gateway to create a connection (a CRCX command), a call agent can simultaneously instruct the gateway to notify the call agent of certain events. Thus, we can find a NotificationRequest encapsulated within a CreateConnection command. This feature is particularly useful in cases when certain actions need to be taken or events need to be detected only in conjunction with other activities. For example, a call agent might require an MG to detect and report DTMF tones, but only when a call is in progress. In such a case, a NotificationRequest command instructing the gateway to detect and report DTMF tones would be encapsulated within a CreateConnection command, so that the tone detection would only occur in the context of the connection created.

MGCP enables only one level of encapsulation. In other words, one command cannot be encapsulated with a second command, if that second command is already encapsulated within a third command. However, MGCP does enable multiple commands to be sent at the same time within the same *User Datagram Protocol* (UDP) packet.

MGCP Parameters Each parameter is identified by a code that comprises one or two uppercase characters. A parameter code is followed by a single colon, one space character, and the parameter value. In some cases, the parameter value might be a single numeric value. In other cases, the parameter value might be a single hexadecimal string. In still other cases, it might be a comma-separated list of values or might comprise a list of subfields. The same set of parameters is available for use in commands and responses.

MGCP defines the following parameters, where the parameter code is indicated within parentheses:

- **BearerInformation (B)** Specifies the line side encoding as A- or μ-law. For example, B:e:mu means μ-law encoding.
- **CallId (C)** The unique identifier for a single call, comprised of hexadecimal digits.
- **Capabilities (A)** Coded in the same way as LocalConnectionOptions, this is used in response to an audit to indicate the capabilities supported by an endpoint.
- **ConnectionId (I)** Unique identifier for a connection on an endpoint, comprised of hexadecimal digits.

Motorola - Exhibit 1022 - Page 281

- **ConnectionMode (M)** Includes options such as send-only, receive-only, and send-receive. These choices enable a given endpoint to be configured for one-way or two-way media transfers.
- **ConnectionParameters (P)** Connection-related statistical information such as average latency, average jitter, the number of packets sent/received, and the number of lost packets. These statistics are reported to a call agent at the end of a call, within the response to the DeleteConnection command. They may also be returned to the call agent as a response to the AuditConnection command.
- **DetectEvents (T)**—A list of events that an endpoint should detect. This list might include the detection of such items as off-hook, on-hook, hook-flash, DTMF digits, and so on.
- **DigitMap (D)**—A digit map is a representation of a dialing plan and specifies rules for which combinations of digits can be dialed. The purpose of the digit map is to enable the gateway to understand what types of digit strings might be received from a caller. When equipped with a digit map, the gateway is able to collect a set of digits and send them en bloc to the call agent, rather than sending each digit individually. The digit map is sent in a *Notification Request* (RQNT) command from the call agent to the gateway.
- **EventStates (ES)** Used in response to an audit command, this parameter is a comma-separated list of events associated with the current state of the endpoint and can indicate which events caused the endpoint to be in that particular state.
- **LocalConnectionDescriptor (LC)** An SDP session description for the current connection on an endpoint, indicating at which IP address and port number the endpoint is expecting to receive media and in what format. In a two-party call, the LocalConnectionDescriptor for the connection at one end corresponds to the RemoteConnectionDescriptor for the connection at the other end.
- **LocalConnectionOptions (L)** A list of connection options such as bandwidth, packetization period, silence suppression, gain control, echo cancellation, and encryption. These are connection-related characteristics that a call agent suggests to a gateway when a connection is to be established. In the syntax, the parameter is identified by L:. The individual components are separated by commas, and each is identified by one or two characters, followed by a colon and the specific value. For example, to turn echo cancellation off and to turn silence suppression on, the parameter would be L: e:off, s:on.

Motorola - Exhibit 1022 - Page 282

- **MaxMGCPDatagram (MD)** Indicates the maximum size MGCP datagram (MGCP packet) supported by an MG. The parameter can be included in the response to an AUEP command.
- **NotifiedEntity (N)** An address for a call agent to which notify commands should be sent from a gateway.
- **ObservedEvents (O)** A list of events detected by an endpoint. This parameter is used to report the events to the call agent. Generally, the events reported will be a subset of those requested by the RequestedEvents in a command previously received from the call agent. The RequestIdentifier (described later) is used to correlate events requested by the call agent with events reported by the gateway.
- **Package List (PL)** A list of packages supported by an endpoint. Different types of endpoints have different characteristics. A given endpoint can support only certain types of events, tones, or actions. For example, a dial tone is an event that would be associated only with an analog line endpoint. To ease protocol handling, events and signals are grouped into packages. For example, the line (L) package includes events and signals (such as the detection of off-hook or on-hook) that are associated only with analog line endpoints. The PL parameter can be included in the response to multiple commands.
- **QuarantineHandling (Q)** Indicates how an endpoint should handle events that occur during the period while the gateway is waiting for a response to a Notify command sent to a call agent (the quarantine period). The choices are that the endpoint should go ahead and process the events or simply discard them.
- **ReasonCode (E)** Used by a gateway when deleting a connection to indicate the cause of the deletion or after a restart of an endpoint to indicate the reason for the restart.
- **RemoteConnectionDescriptor (RC)** An SDP session description related to the far end of a call. This parameter indicates an address and port to which media should be sent and the format of that media.
- **RequestedEvents (R)** A list of events that an endpoint is to watch for. Associated with each event, the endpoint may be instructed to perform one or more actions, such as notify the call agent immediately, collect digits, or apply a signal to the line. For example, an endpoint might be instructed to detect an off-hook evet on an analog line and, upon detection of the event, to apply a dial tone to the line.
- **RequestedInfo (F)** Information to be provided by a gateway to a call agent in response to either of the two audit requests. The requested

Motorola - Exhibit 1022 - Page 283

information can include such items as the current value of RequestedEvents, the current value of DigitMap, or the current value of NotifiedEntity. This parameter should not be confused with the RequestedEvents parameter.

- **RequestIdentifier (X)** Used in RQNT and NTFY commands to identify a particular request for information. The purpose of the parameter is to enable a call agent to correlate a given notification from a gateway with a particular notification request previously sent from the call agent to the gateway.
- **ResponseAck (K)** A list of one or more transaction ID ranges.
- **RestartDelay (RD)** A number of seconds indicating when an endpoint will be brought back into service.
- **RestartMethod (RM)** Used to indicate whether a restart at an endpoint is graceful (existing connections are not terminated) or forced (existing connections are terminated). This parameter is also used to indicate that an endpoint is about to be brought back into service after a specified delay.
- **SecondConnectionId (I2)** Connection identifier associated with the connection on a second endpoint. When a command, such as CreateConnection, specifies a second endpoint, then actually two connections are created. Each has its own identifier and each may be manipulated through commands such as ModifyConnection.
- **SecondEndpointID (Z2)** Used to indicate a second endpoint on the same gateway. This would apply when a call agent wants to create a connection between two endpoints on the same gateway. It can be used in the command to specify the second endpoint to be included in the connection. Its use in a command may include the use of the any ($) wildcard. In such a case, the parameter will also be used in the response to the command to indicate the specific endpoint chosen as the second endpoint.
- **SignalRequests (S)** Signals to be applied by an endpoint to the media stream, such as a ring tone or dial tone.
- **SpecificEndpointID (Z)** Used to indicate a single endpoint, particularly in a response to a command where a wildcard was used by the call agent. For example, when creating a connection using CRCX, the call agent might not care exactly which endpoint is used for the connection and might let the gateway make the choice. In such a case, the gateway would respond to the CRCX command and would use this parameter in that response to indicate the identity of the exact endpoint chosen.

Not all parameters will appear in all commands. Rather, for each command, certain parameters are mandatory, certain parameters are optional, and certain parameters are forbidden. Table 6-1 shows the mapping of command parameters to commands, where M = mandatory, O = optional, and F = Forbidden.

Overview of MGCP Responses

For each MGCP command, a response is returned. Responses are comprised of a response line, optionally followed by a number of response parameters. The response line is made up of a return code, followed by the transaction identifier, optionally followed by a commentary or reason phrase. Each of these is separated by a *single space* (SP) character and the response line is terminated by a *carriage return line feed* (CRLF) and may be represented as

```
Return code SP TransactionID SP Commentary CRLF
```

A return code is an integer value. Return codes are divided into the following categories:

- **0XX (000 to 099)** This indicates an acknowledgement to a response.
- **1XX (100 to 199)** Provisional responses. A final response will follow later.
- **2XX (200 to 299)** The command successfully completed.
- **4XX (400 to 499)** Failure due to a transient error
- **5XX (500 to 599)** Failure due to a permanent error.
- **8XX (800 to 899)** Package-specific responses.

Table 6-2 gives a complete listing of the return codes defined in MGCP and their meanings.

A response must be correlated with the command that triggered the response. Therefore, the TransactionId appears in both commands and responses. The response to a command will use the same TransactionId as the command that generated the response.

The same parameters that are available for use in MGCP commands are also available for use in MGCP responses. The allowed usage in responses differs from the allowed usage in commands, however. For example, the LocalConnectionOptions parameter is an optional parameter in a CRCX command, but is forbidden in a response to a CRCX command. Table 6-3 provides a mapping of response parameters to the commands that generate

Table 6-1

Command parameters and commands (as specified in proposed update to RFC 2705)

Parameter name	EPCF	CRCX	MDCX	DLCX	RQNT	NTFY	AUEP	AUCX	RS IP
BearerInformation	O[1]	O	O	O	O	F	F	F	F
CallId	F	M	M	O	F	F	F	F	F
Capabilities	F	F	F	F	F	F	F	F	F
ConnectionId	F	F	M	O	F	F	F	M	F
ConnectionMode	F	M	O	F	F	F	F	F	F
ConnectionParameters	F	F	F	O[2]	F	F	F	F	F
DetectEvents	F	O	O	O	O	F	F	F	F
DigitMap	F	O	O	O	O	F	F	F	F
EventStates	F	F	F	F	F	F	F	F	F
LocalConnectionOptions	F	O	O	F	F	F	F	F	F
MaxMGCPDatagram	F	F	F	F	F	F	F	F	F
NotifiedEntity	F	O	O	O	O	O	F	F	F
ObservedEvents	F	F	F	F	F	M	F	F	F
PackageList	F	F	F	F	F	F	F	F	F
QuarantineHandling	F	O	O	O	O	F	F	F	F
ReasonCode	F	F	F	O	F	F	F	F	O
RequestedEvents	F	O	O	O	O[3]	F	F	F	F
RequestIdentifier	F	O[4]	O[4]	O[4]	M	M	F	F	F
RequestedInfo	F	F	F	F	F	F	O	M	F
ResponseAck	O	O	O	O	O	O	O	O	O
RestartDelay	F	F	F	F	F	F	F	F	O
RestartMethod	F	F	F	F	F	F	F	F	M
SecondConnectionId	F	F	F	F	F	F	F	F	F

Motorola - Exhibit 1022 - Page 286

Table 6-1 cont.

Command parameters and commands (as specified in proposed update to RFC 2705)

Parameter name	EPCF	CRCX	MDCX	DLCX	RQNT	NTFY	AUEP	AUCX	RSIP
SecondEndpointId	F	O	F	F	F	F	F	F	F
SignalRequests	F	O	O	O	O*	F	F	F	F
SpecificEndpointId	F	F	F	F	F	F	F	F	F
RemoteConnectionDescriptor	F	O	O	F	F	F	F	F	F
LocalConnectionDescriptor	F	F	F	F	F	F	F	F	F

[1]The BearerInformation parameter is only conditionally optional and must be included under certain circumstances.

[2]The ConnectionParameters parameter is only valid in a DeleteConnection request sent by the gateway.

[3]The RequestedEvents and SignalRequests parameters are optional in the NotificationRequest. If these parameters are omitted, the corresponding lists will be considered empty.

[4]The RequestIdentifier parameter is optional in connection creation, modification, and deletion commands. The parameter is required, however, if the command contains an encapsulated notification request.

Table 6-2

MGCP return codes

Return Code	Description
000	Response acknowledgement.
100	The transaction is currently being executed. An actual completion message will follow later.
101	The transaction is being queued for later execution. A completion response will follow later.
200	The requested transaction was completed normally.
250	The connection was deleted.
400	The transaction could not be executed due to a transient error.
401	The phone is already off-hook.
402	The phone is already on-hook.
403	Transaction not executed because the endpoint does not have sufficient resources at this time.
404	Insufficient bandwidth at this time.
405	The transaction could not be executed because the endpoint is restarting.
406	Transaction timeout. The transaction did not complete in a reasonable time and has been aborted.
407	Transaction aborted. The transaction has been aborted due to some external (from management, for example) action.
409	The transaction has not been executed because of internal overload.
410	No endpoint available. An any ($) wildcard was used in the request, but no suitable endpoint is available to satisfy the request.
500	The transaction was not executed because the endpoint is unknown.
501	The transaction was not executed because the endpoint is not ready.
502	The transaction was not executed because the endpoint does not have sufficient resources.
503	All of (*) wildcard too complicated.
504	Unknown or unsupported command.
505	Unsupported RemoteConnectionDescriptor.
506	Unable to satisfy both LocalConnectionOptions and RemoteConnectionDescriptor.
508	Unknown or unsupported quarantine handling.
509	Error in RemoteConnectionDescriptor.
510	The transaction was not executed because a protocol error was encountered.
511	The transaction was not executed because the command contained an unrecognized extension.
512	The transaction was not executed because the gateway is not equipped to detect one of the requested events.

Motorola - Exhibit 1022 - Page 288

Table 6-2 cont.
MGCP return codes

Return Code	Description
513	The transaction was not executed because the gateway is not equipped to generate one of the requested signals.
514	The transaction was not executed because the gateway cannot send the requested announcement.
515	The transaction refers to an incorrect ConnectionID (may already have been deleted).
516	The transaction refers to an unknown CallID.
517	Unsupported or invalid mode.
518	Unsupported or invalid package.
519	Endpoint does not have a digit map.
520	Transacting could not be executed because the endpoint is restarting.
521	Endpoint redirected to another call agent.
522	No such event or signal.
523	Unknown action or illegal combination of actions.
524	Internal inconsistency in LocalConnectionOptions.
525	Unknown extension in LocalConnectionOptions.
526	Insufficient bandwidth.
527	Missing RemoteConnectionDescriptor.
528	Incompatible protocol version.
529	Internal hardware failure.
530	CAS signaling protocol error.
531	Failure of a grouping of trunks (such as facility failure).
532	Unsupported value(s) in LocalConnectionOptions.
533	Response too large.
534	Codec negotiation failure.
535	Packetization period not supported.
536	Unknown or unsupported RestartMethod.
537	Unknown or unsupported digit map extension.
538	Event/signal parameter error (missing, erroneous, unsupported, unknown, and so on) package or vendor extension parameter.
539	Invalid or unsupported command parameter. This code should only be used when the parameter is neither a package or vendor extension parameter.
540	Per-endpoint connection limit exceeded.
541	Invalid or unsupported LocalConnectionOptions. This code should only be used when the LocalConnectionOptions is neither a package nor a vendor extension LocalConnectionOptions.

the response, where M is mandatory, O is optional, and F is forbidden. We can see from the table that responses generally contain fewer parameters than commands.

Table 6-3 Response parameters

Parameter name	EPCF	CRCX	MDCX	DLCX	RQNT	NTFY	AUEP	AUCX	RSIP
BearerInformation	F	F	F	F	F	F	O	F	F
CallId	F	F	F	F	F	F	F	O	F
Capabilities	F	F	F	F	F	F	O[1]	F	F
ConnectionId	F	O[2]	F	F	F	F	O[1]	F	F
ConnectionMode	F	F	F	F	F	F	F	O	F
ConnectionParameters	F	F	F	O[3]	F	F	F	O	F
DetectEvents	F	F	F	F	F	F	O	F	F
DigitMap	F	F	F	F	F	F	O	F	F
EventStates	F	F	F	F	F	F	O	F	F
LocalConnectionOptions	F	F	F	F	F	F	F	O	F
MaxMGCPDatagram	F	F	F	F	F	F	O	F	F
NotifiedEntity	F	F	F	F	F	F	O	O	O
ObservedEvents	F	F	F	F	F	F	O	F	F
QuarantineHandling	F	F	F	F	F	F	O	F	F
PackageList	O[4]	O[4]	O[4]	O[4]	O[4]	O[4]	O	O[4]	O[4]
ReasonCode	F	F	F	F	F	F	O	F	F
RequestIdentifier	F	F	F	F	F	F	O	F	F
ResponseAck	O[5]	O[5]	O[5]	O[5]	O[5]	O[5]	O[5]	O[5]	O[5]
RestartDelay	F	F	F	F	F	F	O	F	F
RestartMethod	F	F	F	F	F	F	O	F	F
RequestedEvents	F	F	F	F	F	F	O	F	F
RequestedInfo	F	F	F	F	F	F	F	F	F
SecondConnectionId	F	O	F	F	F	F	F	F	F
SecondEndpointId	F	O	F	F	F	F	F	F	F

Motorola - Exhibit 1022 - Page 290

Table 6-3 cont.

Response parameters

Parameter name	EPCF	CRCX	MDCX	DLCX	RQNT	NTFY	AUEP	AUCX	RSIP
SignalRequests	F	F	F	F	F	F	O	F	F
SpecificEndpointId	F	O	F	F	F	F	O[1]	F	F
LocalConnection Descriptor	F	O[6]	O	F	F	F	F	O[7]	F
RemoteConnection Descriptor	F	F	F	F	F	F	F	O[7]	F

[1]Multiple ConnectionId, SpecificEndpointId, and Capabilities parameters may be present in the response to an AuditEndpoint command.

[2]In the case of a CreateConnection message, the response line is followed by a ConnectionId parameter and a LocalConnectionDescriptor. It may also be followed a SpecificEndpointId parameter if the creation request was sent to a wildcarded EndpointId. The ConnectionId 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 response to a non-wildcarded DeleteConnection command sent by the call agent.

[4]The PackageList parameter is only allowed with return code 518 (unsupported package), except for AuditEndpoint, where it can also be returned if audited.

[5]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 should trigger a Response Acknowledgement; any ResponseAck values provided will be ignored.

[6]A LocalConnectionDescriptor *must* be transmitted with a positive response (code 200) to a CreateConnection. The parameter must also be transmitted in response to a ModifyConnection command, if the modification resulted in a modification of the session parameters. The LocalConnectionDescriptor is encoded as an SDP session description.

[7]When several session descriptors are encoded in the same response, they are encoded one after each other, separated by an empty line.

Command and Response Details

From the information in Tables 6-1 and 6-3, we can identify the structure of each command and the corresponding response. All that remains is to indicate where a given command can be encapsulated within another command. This information is included in the following command and response details, where optional parameters and optional encapsulated commands are shown within square brackets.

Note that the representation of the command and response formats here is slightly different than that in certain parts of the MGCP specification. That is because the MGCP specification defines both a protocol and an

Motorola - Exhibit 1022 - Page 291

application programming interface (API). The format here is based on the protocol and shows the complete command line with the EndpointId and MGCP version included. The format here also shows the complete response line with the ReturnCode, TransactionId, and Commentary all included. The information is presented in this way in order to enable a better visualization of the commands and responses as actually transmitted by a call agent or gateway.

EndpointConfiguration (EPCF) Command

```
EPCF  TransactionId EndpointId MGCP 1.0
          [BearerInformation]
```

EndPointConfiguration (EPCF) Response

```
ReturnCode TransactionId Commentary
             [PackageList]
```

NotificationRequest (RQNT) Command

```
RQNT TransactionId EndpointId MGCP 1.0
           [NotifiedEntity]
           [RequestedEvents]
           RequestIdentifier
           [DigitMap]
           [SignalRequests]
           [QuarantineHandling]
           [DetectEvents]
           [encapsulated EndpointConfiguration]
```

NotificationRequest (RQNT) Response

```
ReturnCode TransactionId Commentary
           [PackageList]
```

Notify (NTFY) Command

```
NTFY TransactionId EndpointId MGCP 1.0
          [NotifiedEntity]
          RequestIdentifier
          ObservedEvents
```

Notify (NTFY) Response

```
ReturnCode TransactionId Commentary
                  [PackageList]
```

CreateConnection (CRCX) Command

```
CRCX TransactionID EndpointID MGCP 1.0
          CallId
          [NotifiedEntity]
          [LocalConnectionOptions]
          Mode
          [RemoteConnectionDescriptor | SecondEndpointId]
          [Encapsulated NotificationRequest]
          [Encapsulated EndpointConfiguration]
```

CreateConnection (CRCX) Response

```
ReturnCode TransactionId Commentary
                ConnectionId
                [SpecificEndpointId]
                [LocalConnectionDescriptor]
                [SecondEndpointId]
                [SecondConnectionId]
                [PackageList]
```

ModifyConnection (MDCX) Command

```
MDCX TransactionId EndpointId MGCP 1.0
          CallId
          ConnectionId
          [NotifiedEntity]
          [LocalConnectionOptions}
          [Mode]
          [RemoteConnectionDescriptor]
          [encapsulated NotificationRequest]
          [encapsulated EndpointConfiguration]
```

ModifyConnection (MDCX) Response

```
ReturnCode TransactionId Commentary
                    [LocalConnectionDescriptor]
                    [PackageList]
```

DeleteConnection (DLCX) Command (from Call agent to Gateway)

```
DLCX TransactionId EndpointId MGCP 1.0
          CallId
          [ConnectionId]*
          [encapsulated NotificationRequest]
          [encapsulated EndpointConfiguration]
```

*A call agent might want to delete all connections for a given endpoint and a given call, rather than just a single connection. In such a case, the DLCX command from the call agent would not include the ConnectionId parameter.

DeleteConnection (DLCX) Response (from Gateway to Call agent)

```
ReturnCode TransactionId Commentary
                        ConnectionParameters
                        [PackageList]
```

DeleteConnection (DLCX) Command (from Gateway to Call Agent)

```
DLCX TransactionId EndpointId MGCP 1.0
          CallId
          ConnectionId
          ReasonCode
          Connectionparameters
```

DeleteConnection (DLCX) Response (from Call Agent to Gateway)

```
ReturnCode TransactionId Commentary
                       [PackageList]
```

AuditEndpoint (AUEP) Command

```
AUEP TransactionId EndpointId MGCP 1.0
            [RequestedInfo]
```

AuditEndpoint (AUEP) Response

```
ReturnCode TransactionId Commentary
                        EndpointIdList | {
                        [RequestedEvents]
                        [QuarantineHandling]
                        [DigitMap]
                        [SignalRequests]
                        [RequestIdentifier]
                        [NotifiedEntity]
                        [ConnectionIdentifiers]
                        [DetectEvents]
                        [ObservedEvents]
                        [EventStates]
                        [BearerInformation]
                        [RestartMethod]
                        [RestartDelay]
                        [ReasonCode]
                        [MaxMGCPDatagram]
                        [Capabilities]
                        [PackageList]
```

AuditConnection (AUCX) Command

```
AUCX TransactionId EndpointId MGCP 1.0
           ConnectionId
           RequestedInfo
```

AuditConnection (AUCX) Response

```
ReturnCode TransactionId Commentary
                         [CallId]
                         [NotifiedEntity]
                         [LocalConnectionOptions]
                         [Mode]
                         [RemoteConnectionDescriptor]
                         [LocalConnectionDescriptor]
                         [ConnectionParameters]
                         [PackageList]
```

RestartInProgress (RSIP) Command

```
RSIP TransactionId EndpointId MGCP 1.0
          RestartMethod
          [RestartDelay]
          [ReasonCode]
```

RestartInProgress (RSIP) Response

```
ReturnCode TransactionId Commentary
                        [NotifiedEntity]
                        [PackageList]
```

Call Setup Using MGCP

Utilizing the foregoing descriptions of commands and responses, it is possible to describe MGCP-based call setup in some detail. The following description of the process assumes that two endpoints are to be used on two different gateways, but that both gateways are connected to the same call agent.

Call setup using MGCP involves the creation of at least two connections, as shown in Figure 6-7. First, a call agent uses the CRCX command to instruct a gateway to create a connection on an endpoint. The call agent may specifically identify the endpoint to be used or may use a wildcard notation to enable the gateway to choose the endpoint. The gateway creates the connection and responds to the call agent with a session description (LocalConnectionDescriptor) applicable to the connection. The LocalConnectionDescriptor is an SDP description of the session as it applies to the connection just created. Therefore, it contains information such as an IP address, port number, and media coding.

Upon receipt of the response from the first gateway, the call agent instructs the second gateway to create a connection on an endpoint. The call agent has more information available for inclusion in the CRCX command

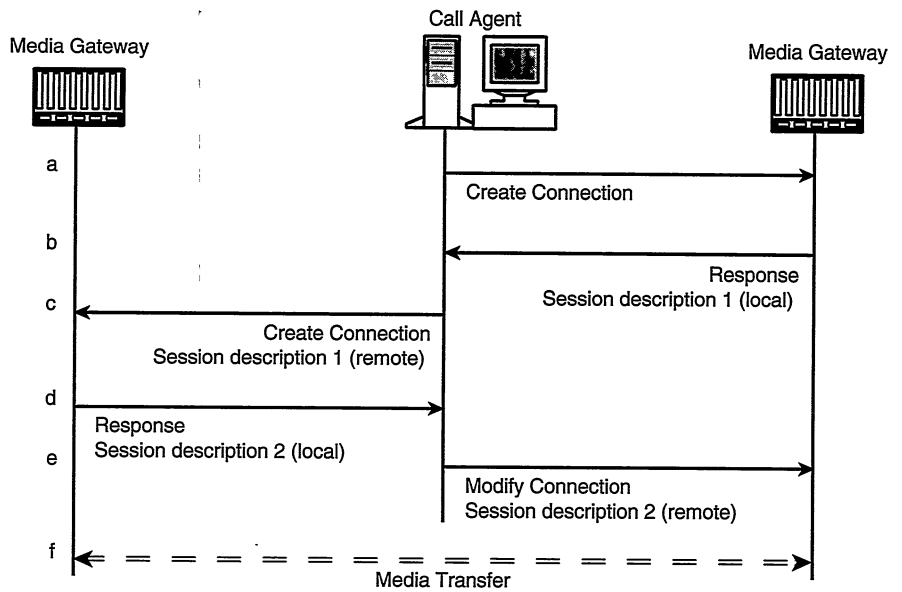


Figure 6-7 MGCP call establishment overview

to the second gateway than it had for the CRCX command for the first gateway. Specifically, the call agent is armed with knowledge of the session description applicable to the connection on the first endpoint, and it can pass this information to the second endpoint. Therefore, the call agent issues a CRCX command to the second gateway and includes the optional RemoteConnectionDescriptor parameter, which contains the session description obtained for the connection made on the first endpoint. Thus, the second endpoint in the call now has a session description related to the first endpoint and knows where to send media. For the moment, the first endpoint in the call is still in the dark.

The second endpoint sends a response to the CRCX command and includes the LocalConnectionDescriptor parameter, which is a session description related to the connection created on the second endpoint.

Third, the call agent issues an MDCX command to the first gateway. The command contains a RemoteConnectionDescriptor parameter, corresponding to the session description received from the second gateway for the second endpoint. At this stage, both endpoints now hold a session description for the other and know exactly where to send the media stream and in what format. Media can now be exchanged.

Figure 6-8 shows a detailed example of call establishment between two gateways, with each supporting DS0 endpoints and managed by the same call agent, including the session descriptions involved.

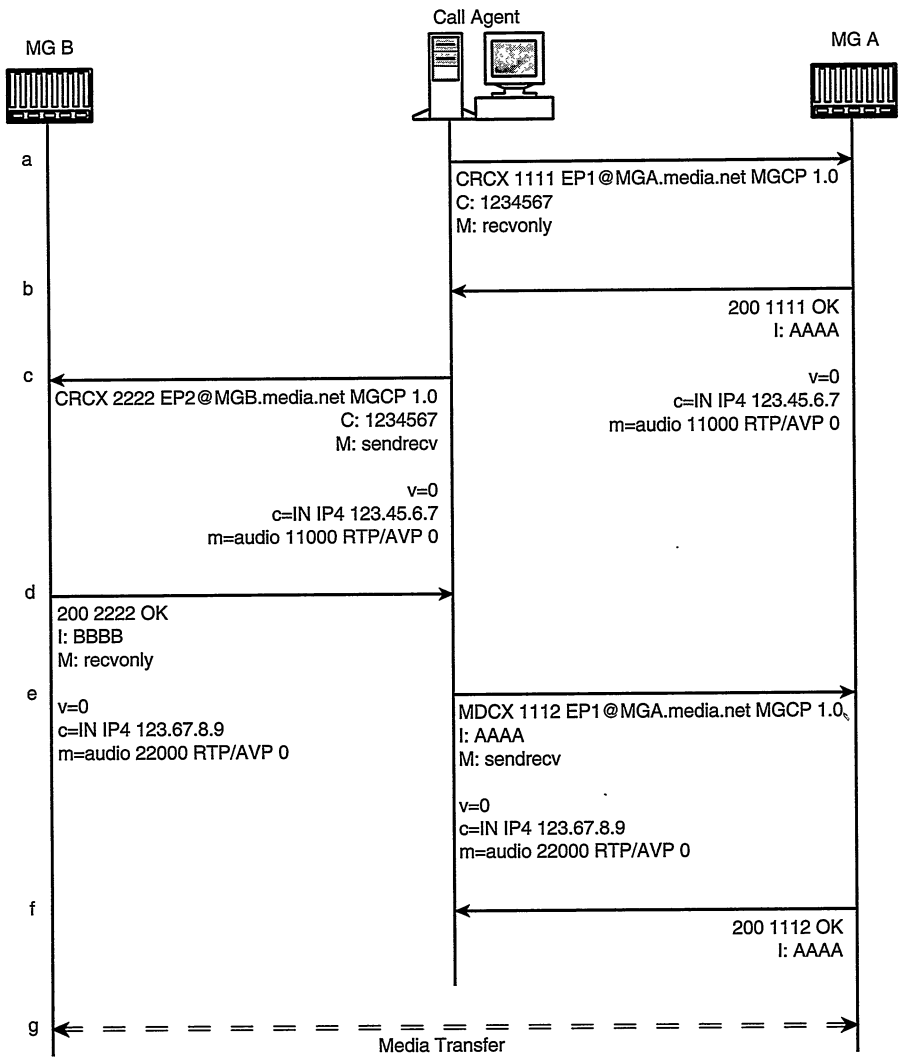


Figure 6-8
MGCP call setup between two gateways

The following text describes the messages of Figure 6-8 in more detail.

a. The call agent sends a CRCX command to MGA, specifying that a connection is to be created on *endpoint 1* (EP1). The command also specifies the TransactionId (111) and the CallId (1234567). No BearerInformation is sent in this case, because the endpoint is specifically identified, and we assume that the bearer-related information for the line side of the endpoint is already known. If that

information were not known or if the selection of endpoint were wildcarded, then the bearer information would specify either A-law or μ-law. The CRCX command specifies the mode to be receive-only for now. This means that the endpoint can receive media from the IP network, but cannot send media to the IP network. The reason for this choice is the fact that the endpoint does not yet have a session description for the remote end and hence cannot know where to send a media stream. The remote end is about to be presented with a session description for the local end, however, and could start to send media before the local endpoint receives a session description for the remote end. Therefore, the local endpoint should be prepared to receive media from the remote endpoint.

b. MG A responds to the CRCX command with the same TransactionId, a ConnectionId (AAAA), and a session description indicating voice coded according to G.711 μ-law (RTP payload format 0). Note that this is the voice coding chosen on the IP side of the endpoint as opposed to the line side of the endpoint. It could equally well be some other coding choice such as G.728 or G.729.

c. The call agent sends a CRCX command to the second gateway (MG B). Again a specific endpoint (EP2) is indicated in the command. The TransactionId is 2222 and the same CallId (1234567) is used. The call agent includes a RemoteConnectionDescriptor, which is simply the session description received earlier from MG A. This time the CRCX command specifies the mode as send-receive, since a session description for the remote end is available, which means that the endpoint has an address and port to which it can send media.

d. MG B sends a positive response to the CRCX, including a ConnectionId (BBBB) and a session description corresponding to that connection. The session description in this case also happens to be G.711 μ-law coded voice.

e. The call agent sends an MDCX command to the command to the first gateway. The command specifically specifies the ConnectionId and includes a RemoteConnectionDescriptor, which is the session description received from MG B. Given that a session description is available for the remote end, MG A now has an address and port to which it may send media. Therefore, the MDCX command specifies the mode to be send-receive.

f. MG A sends a positive response to the MDCX command.

g. Media can now be exchanged between the two endpoints.

MGCP Events, Signals, and Packages

The example of Figure 6-8 was a simple scenario between two trunking gateways. The example also assumes that the call agent has some prior knowledge of the requirements for the call. Specifically, in Figure 6-8, the call agents know *a priori* that a call is to be established between the two gateways. Such a situation implies that the call agent is exchanging signaling with an external network, such as the SS7 network.

The situation can be more complex if the call is triggered in another way. If, for example, MGA were to be a residential gateway that supports analog lines, then the call would be instigated and controlled by events and signals on the line itself. Such events would include a transition from on-hook to off-hook, the application of dial tone to the line, the receipt of dialed digits, and so on. In other words, the MG would detect an off-hook event and would notify the call agent. The MG would also apply a dial tone and wait for digits. Upon receipt of those digits, the MG would pass them to the call agent, which would make the necessary call-routing decisions. The MG would pass the dialed digits to the call agent only if those digits match with a digit map previously downloaded from the call agent. If the dialed digits were invalid (did not match a digit map stored at the MG), then the MG would be required to apply an appropriate tone to the line.

Alternatively, one could imagine a situation where one of the gateways uses *multifrequency* (MF) signaling. In such a case, the gateway would need to detect and send MF signals, including digits for a calling party number and a called party number. We can see that the events that a gateway must detect and the signals that a gateway must apply depend on the function of the gateway and its role in the network.

Because different types of endpoints will need to detect different events and apply different signals, MGCP groups events and signals into packages, and a given endpoint can implement one or more packages. These packages have package names and a given event or signal is identified by a package name and an event or signal name, with the format "package name/event name." For example, a gateway that terminates analog lines will implement the line package (L). The off-hook event has the abbreviation hd. Therefore, when referring to an off-hook event detected by an analog line endpoint, the formal identification is L/hd. Incidentally, the package name is actually optional when it comes to identifying an event or signal, since each event or signal has a unique identifier.

The use of package names is useful with wildcarding, however. If, for example, a gateway were required to detect every event in L, then the

events to be detected can be represented by L/*. This format is a lot easier than listing every event that might occur. Wildcarding can also be applied to the package name. If for example, the event "surprise" happened to be included in more than one package, then the syntax */surprise would refer to the surprise event in every package.

When the SignalRequests and the RequestedEvents parameters are both specified in an RQNT command, then the signals that are requested are synchronized with the events that are to be detected. For example, if a ringing signal is to be applied to a line and an off-hook event is to be detected, then the ringing signal will be stopped as soon as the off-hook event occurs.

A number of signals and events are defined in the MGCP specification and are grouped into various packages. Whenever new events or packages need to be defined, then these should be documented and the names of the events or packages should be registered with the *Internet Assigned Numbers Authority* (IANA). Prior to registration with the IANA, developers may want to test new packages. Such experimental packages should have names beginning with the two characters "x-." These characters indicate an experimental extension to the protocol, and packages starting with these characters do not need to be registered.

Interworking Between MGCP and SIP

The example of Figure 6-8 assumes that two gateways are used, but that a single call agent controls both gateways. It is equally likely that separate call agents control the gateways. In such a case, a need exists for signaling between the two call agents. In the softswitch architecture, we commonly use SIP.

Figure 6-9 is an adaptation of Figure 6-8, with the difference that separate call agents control the gateways. The figure shows how MGCP and SIP fit very well together, particularly since they both utilize SDP for describing sessions. Once the connection is created on the first endpoint (line b of Figure 6-10), the returned session description becomes the message body of a SIP INVITE sent to the other call agent (line c). Note that most of the SIP headers and other fields have been omitted, and to avoid clutter, the names of the endpoints and the TransactionIds have been shortened compared to Figure 6-8.

Once the second connection is created, on the second gateway (line e), the corresponding session description becomes the message body of the SIP 200

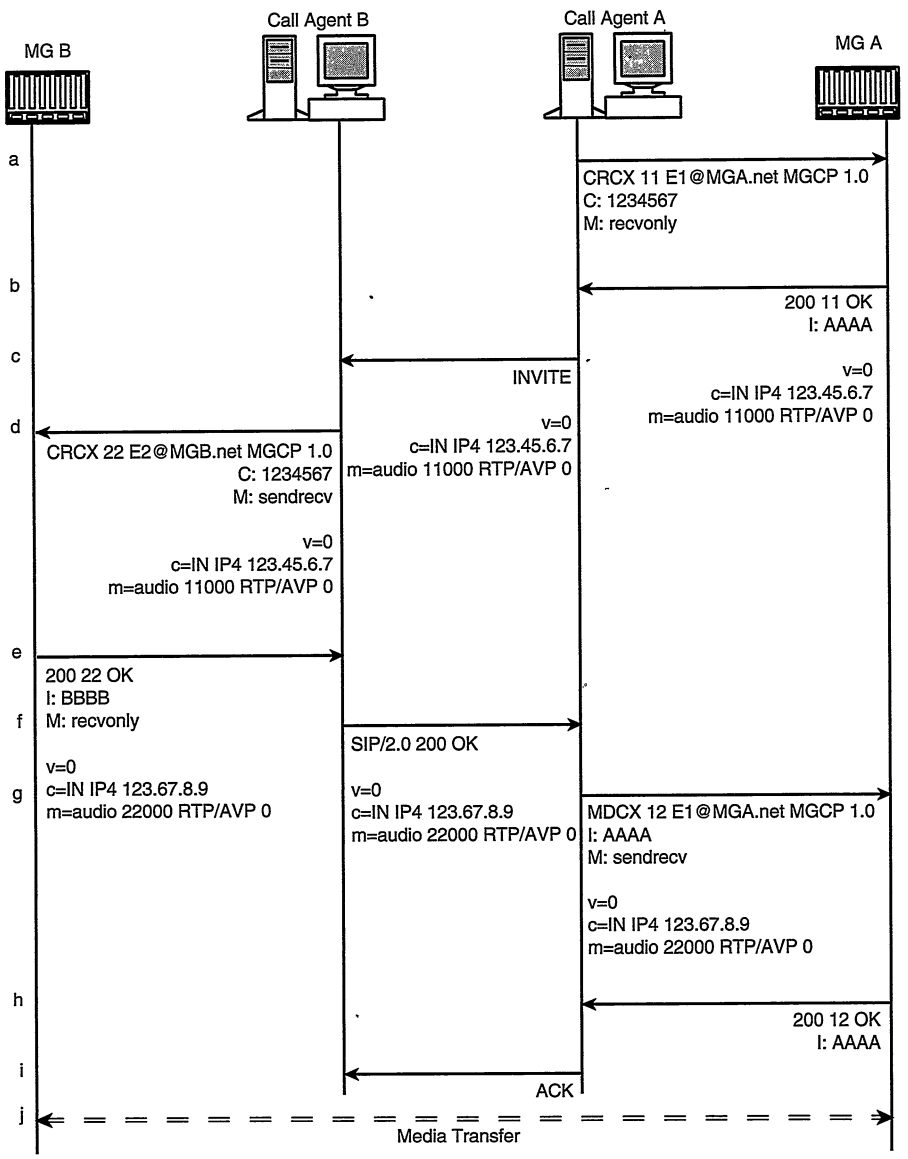


Figure 6-9
Call setup using MGCP and SIP

(OK) response (line f). In turn, the session description is mapped to the MDCX command sent to the first endpoint (line g). Finally, the positive acknowledgment to the MDCX command (line h) is mapped to the SIP ACK (line i), thereby completing the MGCP and SIP signaling sequences.

MEGACO/H.248

Known as MEGACO within the IETF and H.248 within the ITU, this protocol has widespread backing within both the Internet and telecommunications community. In fact, the cooperation between the two industry organizations reflects the convergence that has occurred between traditional telecommunications and the Internet.

Given the fact that the term MEGACO/H.248 is rather a mouthful, for the purpose of brevity, the remainder of the book uses only the term MEGACO.

MEGACO Architecture

MEGACO has a number of architectural similarities to MGCP. Similar to MGCP, it defines MGs, which perform the conversion of media from the format required in one network to the format required in another. MEGACO defines MGCs, which control call establishment and teardown within MGs and are akin to call agents as defined by MGCP. The similarities are hardly that surprising, given that fact that they have both been designed to address a similar set of requirements.

The MEGACO protocol involves a series of transactions between MGCs and MGs. Each transaction involves the sending of a TransactionRequest by the initiator of the transaction and the sending of a TransactionReply by the responder. A TransactionRequest comprises a number of commands and the TransactionReply comprises a corresponding number of responses. For the most part, transactions are requested by an MGC and the corresponding actions are executed within an MG. As in MGCP, however, a number of cases occur when an MG initiates the transaction request.

The MEGACO specification is written such that it provides both a text encoding and a binary encoding of the protocol. The text encoding is written according to *Augmented Backus-Naur Form* (ABNF), described in RFC 2234. The binary encoding is written according to *Abstract Syntax Notation One* (ASN.1), described in ITU-T Rec. X.680, 1997. This reflects one difference between the IETF, which normally uses ABNF, and the ITU where ASN.1 is the format of choice. MGCs should support both types of encoding, while MGs may support both types of encoding but are not required to do so. Within this book, the ABNF form is used when offering examples of MEGACO usage as it is more human-readable than ASN.1.

Terminations MEGACO defines terminations, which are logical entities on an MG that act as sources or sinks of media streams. Certain terminations are physical. They have a semipermanent existence and may be associated with external physical facilities or resources. These would include a termination connected to an analog line or a termination connected to a DS0 channel. Physical terminations exist as long as they are provisioned within the MG and are akin to endpoints as defined in MGCP.

Other terminations have a more transient existence and only exist for the duration of a call or media flow. These are known as ephemeral terminations and represent media flows such as an RTP session. These terminations are created as a result of a MEGACO Add command, which is somewhat similar in function to the CreateConnection command in MGCP. Ephemeral terminations are destroyed by means of the Subtract command, which is similar to the MGCP DeleteConnection command.

Terminations have specific properties and the properties of a given termination will vary according to the type of termination. Clearly, for example, a termination connected to an analog line will have different characteristics to a termination connected to a TDM channel (such as a DS0). The properties associated with a termination are grouped into a set of descriptors. These descriptors can be included in MEGACO commands, thereby enabling termination properties to be changed according to instruction from MGC to MG.

A TerminationID references a termination, and the MG chooses the TerminationID schema to be used. Similar to MGCP, MEGACO enables the use of the wildcards all (*) and choose ($). A special TerminationID called Root can also be used. This TerminationID is used to refer to the gateway as a whole rather than to any specific terminations within the gateway.

Contexts A context is an association between a number of terminations for the purposes of sharing media between those terminations. Terminations may be added to contexts, removed from contexts, or moved from one context to another. A termination can exist in only one context at any time, and terminations in a given gateway can exchange media only if they are in the same context.

A termination is added to a context through the use of the Add command. If the Add command does not specify a context to which the termination should be added, then a new context is created as a result of the execution of the Add command. This is the only mechanism for creating a new context. A termination is moved from one context to another through the use of the Move command and is removed from a context through the use of the Subtract command. If the execution of a Subtract command results in the

removal of the last termination from a given context, then that context is deleted.

The relationship between terminations and contexts is illustrated in Figure 6-10, where a gateway is depicted with four active contexts. In context C1, we see a simple two-way call across the MG. In Context C2, we see a three-way call across the MG. In contexts C3 and C4, we see a possible implementation of call waiting. In context C3, terminations T6 and T7 are involved in a call. Another call arrives from termination T8 for termination T7. If the user wanted to accept this waiting call and place the existing call on hold, then this could be achieved by moving termination T7 from context C3 to context C4.

The existence of several terminations within the same context means that they have the potential to exchange media. The existence of terminations in the same context does not necessarily mean that they can all send data to each other and receive data from each other at any given time, however. The context itself has certain attributes, including the topology, which

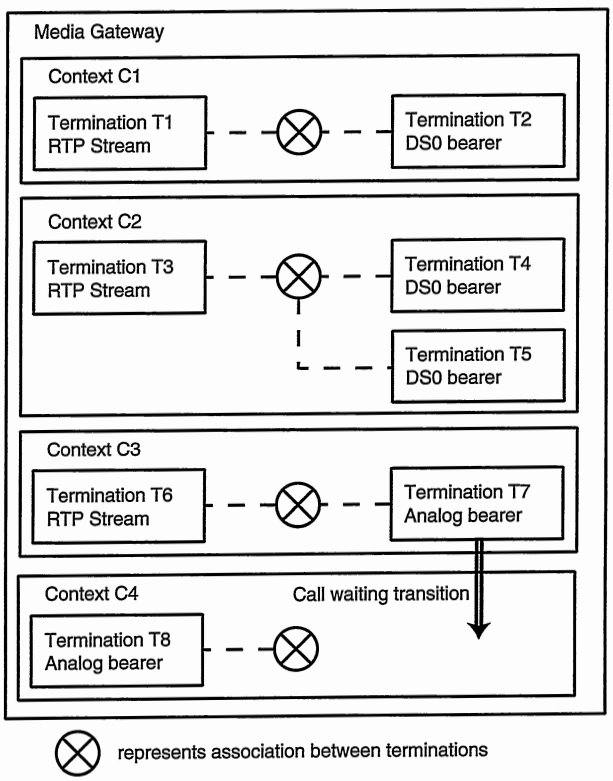


Figure 6-10 MEGACO contexts and terminations, showing possible Move command

indicates the flow of media between terminations (which terminations may send media to others or receive media from others). Also, the priority attribute indicates the precedence applied to a context when an MGC must handle many contexts simultaneously, and an emergency attribute is used to give preferential handling to emergency calls.

A context is identified by a ContextID, which is assigned by the MG and is unique within a single MG. As is the case for terminations, MECAGO enables wildcarding when referring to contexts, such that the all (*) and choose ($) wildcards may be used. The all wildcard may be used by an MGC to refer to every context on a gateway. The choose wildcard ($) is used when an MGC requires the MG to create a new context.

A special context is also available, known as the null context. This context contains all terminations that do not exist in any other context. Idle terminations normally exist in the null context. The ContextID for the null context is simply "-".

Transactions MEGACO transactions involve the passing of commands and the responses to those commands. Commands are directed towards terminations within contexts. In other words, every command specifies a ContextID and one or more TerminationIDs to which the command applies. This is the case even for a command that requires some action by an idle termination that does not exist in any specific context. In such a case, the null context is applicable.

Commands between an MGC and an MG are grouped together in a transaction structure where a set of commands related to one context may be followed by a set of commands related to another context. The grouped commands are sent together in a single TransactionRequest. This can be represented as

```
TransactionRequest (TransactionID {
          ContextID1 {Command, Command, ... Command},
          ContextID2 {Command, Command, ... Command},
          ContextID3 {Command, Command, ...Command}  } )
```

A TransactionRequest does not need to contain commands for more than one context or even contain more than one command. A TransactionRequest containing just a single command for a single context is perfectly valid.

Upon receipt of a TransactionRequest, the recipient executes the enclosed commands. The commands are executed sequentially in the order specified in the TransactionRequest. Upon completed execution of the commands, a TransactionReply is issued. This has a similar structure to the

TransactionRequest in that it contains a number of responses for a number of contexts. A TransactionReply may be represented as

```
TransactionReply (TransactionID {
          ContextID1 {Response, Response, ... Response},
          ContextID2 {Response, Response, ... Response},
          ContextID3 {Response, Response, ...Response}  } )
```

As mentioned, commands within a transaction are executed in sequence. If a given command in the transaction happens to fail, then the subsequent commands in the transaction are not processed. One exception is when a transaction contains one or more optional commands. If an optional command fails, then the processing of the remaining commands will continue. Optional commands in a transaction are indicated by inserting the string O- immediately prior to the command name.

If the recipient of a TransactionRequest might take some time to execute the request, the recipient can return an interim reply so that the sender of the request does not assume that it has been lost. This interim reply is simply an indication that the TransactionRequest has been received and is being handled. The term applied to this response is TransactionPending. It simply returns the TransactionID received and no parameters. The TransactionID is a 32-bit integer.

```
TransactionPending (TransactionID { })
```

Messages Not only is it possible to combine multiple commands within a single transaction, but it is also possible to concatenate multiple transactions within a message. In ABNF syntax, a message begins with the word MEGACO followed by a slash, the protocol version number (1), a message ID (mId), and finally the message body. The mId can be the domain name or IP address (and optionally the port) of the entity transmitting the message. As an alternative, it can be the MTP point code and network indicator of the entity issuing the message.

The following example combines the concept of a message, transactions, and commands and it shows the text format of a message:

```
MEGACO/1 [111.111.222.222]:34567
Transaction = 12345 {
          Context = 1111 {
                    Add = A5555,
                    Add = A6666
          }
            Context = $ {
                    Add = A7777
            }
}
```

In this example, the message is issued by an MGC at address 111.111.222.222 and port 34567. One transaction is identified by TransactionID 12345 and two commands are related to context 1111. Both commands are Add commands, causing terminations A5555 and A6666 to be added to context 1111. The transaction includes one command related to any context that the MG may choose. This command is an Add command, whereby termination A7777 is added to the context. The processing of this command would result in the MG creating a new context, for which the ContextID would be returned in the response.

Transactions within a message are treated independently. The order of the transactions within the message does not imply an order in which the recipient of the message should handle the transactions. Note that this is quite different from the handling of commands within a transaction, where the ordering of the commands is of significance.

Overview of MEGACO Commands

MEGACO defines eight commands that provide the ability to control and manipulate contexts and terminations. Most of the commands are sent from an MGC to an MG. The exceptions are the exception of the Notify command, which is always sent from MG to MGC and the ServiceChange command, which may be sent from either an MG or an MGC. The eight commands are as follows:

- **Add** The Add command adds a termination to a context. If the command does not specify a particular context to which to add the termination, then a new context is created. If the command does not indicate a specific TerminationID but instead uses the choose ($) wildcard, the MG will create a new ephemeral termination and add it to the context.
- **Modify** The Modify command changes the property values of a termination, instructs a termination to issue one or more signals, or instructs a termination to detect and report specific events.
- **Subtract** The Subtract command removes a termination from a context. The response to the command can provide statistics related to the termination's participation in the context. These reported statistics depend upon the type of termination in question. For an RTP termination, the statistics can include items such as packets sent,

packets received, jitter, and so on. In this respect, the command is similar to the MGCP DLCX command. If the result of a Subtract command is the removal of the last termination from a context, then the context itself is deleted.

- **Move** The Move command moves a termination from one context to another. The command is not used to move a termination from or to the null context as these operations must be performed with the Add and Subtract commands respectively. The capability to move a termination from one context to another provides a useful tool for accomplishing the call-waiting service.
- **AuditValue** The AuditValue command is used by the MGC to retrieve current values for properties, events, and signals associated with one or more terminations.
- **AuditCapabilities** The AuditCapabilities command is used by an MGC to retrieve the possible values of properties, signals, and events associated with one or more terminations. At first glance, this command may appear very similar to the AuditValue command. The difference between them is that the AuditValue command is used to determine the current status of a termination, whereas the AuditCapabilities command is used to determine the possible statuses that a termination might assume. For example, AuditValue would indicate any signals that are currently being applied by a termination, where AuditCapabilities could indicate all the possible signals that the termination could apply if required.
- **Notify** The Notify command is issued by an MG to inform the MGC of events that have occurred within the MG. The events to be reported will typically have previously been requested as part of a command from the MGC to the MG, such as a Modify command. The events reported will be accompanied by a RequestID parameter to enable the MGC to correlate reported events with previous requests.
- **ServiceChange** The ServiceChange command enables an MG to inform an MGC that a group of terminations is about to be taken out of service or is being returned to service. The command is also used in a situation where an MGC is handing over control of an MG to another MGC. In that case, the command is first issued from the controlling MGC to the MG to instigate the transfer of control. Subsequently, the MG issues the ServiceChange command to the new MGC as a means of establishing the new relationship.

Descriptors

MECAGO defines a number of descriptors available for use with commands and responses. These descriptors form the parameters of the command and/or response and provide additional information to qualify a given command or response. Depending on the command or response in question, a given descriptor will be mandatory, forbidden, or optional. In most cases, when a descriptor is not mandatory, it is optional. Relatively few cases occur when a descriptor is forbidden.

The general format of a descriptor is

```
Descriptorname=<someID>{parm=value, parm=value, . . . }
```

The parameters of a descriptor may be fully specified, underspecified, or overspecified. Fully specified means that the parameter is assigned a unique and unambiguous value, which the command recipient must use in processing the command. Underspecified involves the use of the choose wildcard ($), allowing the command recipient to select any value it can support. Overspecified means that the sender of the command provides a list of acceptable parameter values in order of preference. The command recipient selects one of the offered choices for use. The response to the command will indicate which of the choices was selected.

Modem Descriptor This descriptor specifies the modem type and associated parameters to be used in modem connections for audio, video, or data communications or for text conversion. The following types of modems are included: V.18 (text telephony), V.22 (1,200 bps), V.22bis (2,400 bps), V.32 (9,600 bps), V.32bis (14,400 bps), V.34 (33,600 bps), V.90 (56 Kbps), V.91 (64 Kbps), and synchronous ISDN.

By default, a termination does not have a modem descriptor. In other words, at start up no modem-related properties will be specified for a termination. They will be applied later as a result of a command (Add or Modify) from an MGC to an MG.

The modem descriptor was included in the first version of the MEGACO specification (RFC 3015). The descriptor should not be used in subsequent versions of the protocol and should be ignored if received. Later in this chapter we describe the syntax of the MEGACO protocol. The modem descriptor is listed as a parameter in various commands and responses, but it is included in the MEGACO version 2 syntax only for backwards-compatibility reasons.

Motorola - Exhibit 1022 - Page 309

Multiplex Descriptor In multimedia communications, different media streams can be carried on different bearer channels. The multiplex descriptor specifies the association between media streams and bearers. The following multiplex types are supported: H.221. H.223, H.226, V.76, and Nx64K.

Media Descriptor The media descriptor describes the various media streams. This is a hierarchical descriptor in that it contains an overall descriptor, known as the *termination state descriptor*, which is applicable to the termination in general and it contains a number of *stream descriptors* applicable to each media stream. Furthermore, each stream descriptor also contains up to three subordinate descriptors, known respectively as the *local control descriptor*, the *local descriptor*, and the *remote descriptor*. This type of hierarchy may be represented by the following list:

Media descriptor
  Termination state descriptor
    Stream descriptor
      Local control descriptor
      Local descriptor
      Remote descriptor

Termination State Descriptor This descriptor contains the two properties ServiceStates and EventBufferControl, plus other properties of a termination that are not specific to any media stream.

The ServiceStates property indicates whether the termination is available for use. Three values are applicable: test, out of service, and in service. In service does not mean that the termination is currently involved in a call. Rather it means that the termination is either actively involved in a call or available for use in a call. In service is the default state of a termination.

The EventBufferControl property specifies whether events detected by the termination are to be buffered following detection or processed immediately. To begin, a termination will report events that the MGC has asked it to report, which the MGC will do through the issuance of a command that contains an EventsDescriptor. Whether the termination will actually report the events indicated in the EventsDescriptor depends on whether the EventBufferControl is set to off or lockstep. If set to off, then the termination will report detected events immediately. If the value is set to lockstep, then events will be buffered in a *first-in, first-out* (FIFO) buffer, along with

the time that the event occurred. The contents of the buffer will be examined upon receipt of a new EventsDescriptor specifying the events that the termination is to detect. Inclusion of the EventBufferControl within the TerminationStateDescriptor enables the MGC to turn on or off the immediate notification of events, without having to send a separate EventsDescriptor each time.

The termination state descriptor is optional within a media descriptor.

Stream Descriptor A stream is defined by its subordinate descriptors LocalControlDescriptor, LocalDescriptor, and RemoteDescriptor, and it is identified by a StreamID.

StreamID values are used between an MG and MGC to indicate which media streams are interconnected. Within a given context, streams with the same StreamID are connected. A stream is created by specifying a new StreamID on a particular termination in a context. A stream is deleted by setting empty local and remote descriptors and setting the values of ReserveGroup and ReserveValue in the subordinate LocalControlDescriptor to false.

LocalControlDescriptor This descriptor contains properties specific to the media stream, in particular the Mode property, the ReserveGroup property, and the ReserveValue property.

Mode can take one of the following values: sendonly, receiveonly, send/receive, inactive, or loopback. Send and receive directions are with respect to the exterior of the context. If mode is set to sendonly, then a termination can only send media to entities outside the context. The termination cannot pass media to other entities within the same context. If mode is set to receiveonly, then a termination can receive media from outside the context and pass it to other terminations in the context, but it cannot receive media from other terminations and pass it to destinations outside of the context.

When an MGC wants to add a termination to a context, it can specify a set of choices for the session (using local and remote descriptors) that the MGC would prefer the MG to use, and the MGC specifies these in order of preference. The MG is not obligated to select the first choice offered by the MGC, but it might be obligated to reserve resources for the sessions indicated by the MGC. The properties ReserveValue and ReserveGroup indicate which resources should be reserved for the alternatives specified by the MGC in the LocalDescriptor and RemoteDescriptor.

The LocalDescriptor and RemoteDescriptor may specify several properties and/or property groups. For example, a given SDP description could

indicate two groups of properties: one for G.711 A-law and one for G.729. If the value of ReserveGroup is true, then the MG should reserve resources for one of these property groups. The use of ReserveValue is similar, but it is applied to reserve resources for a particular property value as opposed to a group of properties.

LocalDescriptor and RemoteDescriptor When used with a text encoding of the protocol, the LocalDescriptor and RemoteDescriptor contain zero or more SDP session descriptions describing the session at the local end of a connection and the remote end of a connection respectively.

The MEGACO usage of SDP includes some variations from the strict syntax of SDP as specified in RFC 2327. In particular, the s=, t=, and o= lines are optional, the choose wildcard ($) is permitted, and it is permissible to indicate several alternatives in places where a single parameter value should normally be used.

Each of the LocalDescriptors and RemoteDescriptors may contain several session descriptions, with the beginning of the session description indicated with the SDP v= line.

Events Descriptor This descriptor contains a RequestIdentifier and a list of events that the MG should detect and report, such as off-hook transition, fax tone, and so on. The purpose of the RequestIdentifier is to correlate requests and the consequent notifications. Normally, detected events will be reported immediately to the MGC. However, events may be buffered depending on the value of the EventControlBuffer property (which is a property specified in the TerminationStateDescriptor).

If events are buffered and subsequently need to be reported to the MGC, then the information related to those events is stored in the EventBufferDescriptor.

SignalsDescriptor This descriptor contains a list of signals that a termination is to apply. Signals can be applied to just a single stream or to all streams in a termination. Typical signals will include, for example, tones applied to a subscriber line such as a ring tone or a dial tone. Three types of signals exist:

- **On/off** The signal remains on until it is explicitly turned off.
- **Timeout** The signal persists until a defined period of time has elapsed or until it is turned off, whichever occurs first.
- **Brief** The signal is to be applied only for a very limited time (such as would be the case for an MF signal on R1 MF trunks).

A SignalsDescriptor may include a sequential sequence of signals that are to be played one after another.

Audit Descriptor The audit descriptor specifies a list of information to be returned from an MG to an MGC. The audit descriptor is simply a list of other descriptors that should be returned in the response. These can include the following descriptors: multiplex, events, signals, ObservedEvents, DigitMap, statistics, packages, and EventBuffer.

ServiceChange Descriptor This descriptor is used only in association with the ServiceChange command and includes information such as the type of service change that has occurred or is about to occur, the reason for the service change, and a new address to be used after the service change.

The type of service change is defined by the parameter ServiceChangeMethod, which can take one of the defined values: Graceful, Forced, Restart, Disconnected, Handoff, or Failover. Graceful indicates the removal of terminations from service after a specified delay and without interrupting existing connections. Forced indicates an abrupt removal from service including the loss of existing connections. Restart indicates that restoration of service will begin after a specified delay. Disconnected applies to the whole MG and indicates that connection with the MGC has been restored after a period of lost contact. The MGC can issue an Audit command to verify that termination characteristics have not changed during the period of lost contact. Handoff is used from an MGC to an MG to indicate that the MGC is about to go out of service and that a new MGC is taking over. The handoff value is also used from an MG to a new MGC when trying to establish contact after receiving a handoff from the previous MGC. A failover is sent from an MG to an MGC if an MG has detected a fault and a backup MG is taking over.

The ServiceChangeDelay specifies a number of seconds indicating the delay applicable. It would be used, for example, in conjunction with a Graceful service change. If the parameter is absent or set to zero, then no delay is applicable. In the case of a graceful service change, then a value of zero indicates that the MGC should wait for the natural removal of terminations from their contexts; that is, the MGC should wait for calls on the specified terminations to come to an end on their own.

As suggested by its name, the ServiceChangeReason parameter indicates the reason for the service change. Table 6-4 shows the reasons currently defined in MEGACO.

Motorola - Exhibit 1022 - Page 313

Table 6-4 Service change reasons

Reason Code	Interpretation
900	Service restored
901	Cold boot
902	Warm boot
903	MGC-directed change
904	Termination malfunctioning
905	Termination taken out of service
906	Loss of lower-layer connectivity
907	Transmission failure
908	MG impending failure
909	MGC impending failure
910	Media capability failure
911	Modem capability failure
912	Mux capability failure
913	Signal capability failure
914	Event capability failure
915	State loss
916	Packages change
917	Capability change

DigitMap Descriptor The DigitMap is a description of a dialing plan. In other words, a DigitMap specifies a set of valid dialed digit combinations, and it is stored in the MG so that the MG can send received digits en bloc to the MGC rather than one by one. A digit map can be loaded into an MG through operation and maintenance action or it can be loaded from the MGC by MECAGO commands. If it is loaded from the MGC, then the DigitMap descriptor is used to convey the information.

From a syntax perspective, a digit map is a string or a list of strings, with each string comprising a number of symbols representing the digits 0

through 9 and the letters A through K. The letter X can be used as a wildcard, designating any digit from 0 through 9, and the dot (.) symbol is used to indicate zero or more repetitions of the immediately preceding digit or digit string. This symbol can be used to indicate an indeterminate number length in the dial plan. For example, in the United States, domestic calls use 10-digit numbers. International calls, however, use numbers of variable lengths, such that a call to Germany may require more digits than a call to Iceland. To indicate calls to international numbers, the string "011x." could be used. The 011 indicates an international call and the "x." indicates some number of digits, each in the range 0 to 9.

In addition, the string can contain three symbols indicating a start timer (T), a short timer (S), and a long timer (L). To understand the usage of these timers, consider what happens when someone picks up a standard residential telephone to make a call. First, the off-hook is detected and the system (for example, the telephone exchange) applies a dial tone and gets ready to collect digits. The system will only wait for a certain amount of time for the person to press a button on the phone. If the person does not press a button in time, the system will timeout and take some other action, such as prompting the user.

This initial timeout value corresponds to the start timer. Once the person starts to dial digits, an interdigit timer will be applied. This timer will either be a short timer or a long timer depending on the dial plan. If the person has dialed a certain number of digits and the system still needs more digits to be able to route the call, a short timer will be applied while waiting for the next digit. On the other hand, if the person has dialed sufficient digits to route the call, but further digits could still be received and would cause a different routing, then a long timer will be applied. The purpose of the long timer is for the system to be confident that the person has finished dialing.

An example in the Unites States would be where dialing 0 causes routing to an operator from the local exchange carrier and dialing 00 causes routing to a long-distance operator. After the system has received the first 0, it must wait before routing the call to make sure that the person does not intend to dial a second 0 in the expectation of being connected to a long distance operator. Table 6-5 gives an example of a dial plan.

This dialing plan could be represented in a digit map as

(0|00|[1-7]xxx|8xxxxxxx|Fxxxxxxx|Exx|91xxxxxxxxx|9011x.)

Note that the timers have been omitted from this digit map. This omission does not mean that timing does not occur at the MG; rather, it just means that the timers are applied by default according to the usage rules

Table 6-5
Dial plan

Dialed digits	Interpretation
0	Local operator
00	Long-distance operator
Xxxx	Local extension number (start with 1 through 7)
8xxxxxxx	Local number
#xxxxxxx	Off-site extension
*xx	Star services
91xxxxxxxxxx	Long-distance number
9011 + up to 15 digits	International number

just described. For example, each one of the patterns in this digit map could be preceded by the start timer symbol (T). The fact that the symbol is not listed in the digit map means that the MG acts as if it were placed at the start of each pattern in any case.

StatisticsDescriptor This descriptor provides statistical information regarding the usage of a termination within a given context. The specific statistics to be returned depend upon the type of termination in question.

Strictly speaking, this descriptor is always optional and can be returned as a result of several different commands. The descriptor should be returned when a termination is removed from a context through the Subtract command and should also be returned in response to the AuditValue command.

ObservedEvents Descriptor This descriptor is a mandatory parameter in the Notify command, where it is used to inform the MGC of events that have been detected. This descriptor is optional in most other command responses, except for ServiceChange. When used in a response to the AuditValue command, it is used to report events stored in the event buffer and that have not yet been reported to the MGC. The descriptor contains a RequestIdentifier with a value matching that received in the events descriptor that listed the events to be detected in the first place. This enables a correlation between the events requested and the events reported.

The descriptor enables the optional inclusion of a timestamp with each observed event indicating the time at which the event was detected. If the

timestamp is to be used, it is structured in the form yyyymmddThhmmssss, which includes hundredths of a second. The T separates the year, month, and day from the hour, minute, and seconds. When used, the timestamp itself is separated from the specific event description by a colon (:).

Error Descriptor The error descriptor is returned in a transaction reply when the requested command could not be executed. The descriptor can also be included in a Notify command from MG to MGC. The error descriptor consists of an error code, optionally accompanied by a textual description of the error. The error codes are registered at the IANA. Table 6-6 lists the various MEGACO error codes and related descriptions. New error codes can be specified, in which case they must be registered with the IANA.

Table 6-6 MEGACO error codes

Error code	Error description
400	Bad request
401	Protocol error
402	Unauthorized
403	Syntax error in transaction
404	Syntax error in TransactionReply
405	Syntax error in TransactionPending
406	Version not supported
410	Incorrect identifier
411	The transaction refers to an unknown ContextId
412	No ContextIds available
421	Unknown action or illegal combination of actions
422	Syntax error in action
430	Unknown TerminationID
431	No TerminationID matched a wildcard
432	Out of TerminationIDs or no TerminationID available
433	TerminationID is already in a context
440	Unsupported or unknown package
441	Missing remote descriptor
442	Syntax error in command
443	Unsupported or unknown command
444	Unsupported or unknown descriptor
445	Unsupported or unknown property
446	Unsupported or unknown parameter
447	Descriptor not legal in this command

Motorola - Exhibit 1022 - Page 317

Table 6-6 cont.

MEGACO error codes

Error code	Error description
448	Descriptor appears twice in a command
450	No such property in this package
451	No such event in this package
452	No such signal in this package
453	No such statistic in this package
454	No such parameter value in this package
455	Parameter illegal in this descriptor
456	Parameter or property appears twice in this descriptor
461	TransactionIds in reply do not match request
462	Commands in transaction reply do not match commands in request
463	TerminationID of transaction reply does not match request
464	Missing reply in transaction reply
465	TransactionId in transaction pending does not match any open request
466	Illegal duplicate transaction request
467	Illegal duplicate transaction reply
471	Implied add for multiplex failure
500	Internal gateway error
501	Not implemented
502	Not ready
503	Service unavailable
504	Command received from unauthorized entity
505	Command received before restart response
510	Insufficient resources
512	MG unequipped to detect requested event
513	MG unequipped to generate requested signals
514	MG cannot send the specified announcement
515	Unsupported media type
517	Unsupported or invalid mode
518	Event buffer full
519	Out of space to store digit map
520	MG does not have a digit map
521	Termination is "ServiceChanging"
526	Insufficient bandwidth
529	Internal hardware failure
530	Temporary network failure
531	Permanent network failure
581	Does not exist

Motorola - Exhibit 1022 - Page 318

Topology Descriptor The topology descriptor is somewhat different from other descriptors in that it is relevant to a context only, rather than to a specific termination within a context. The purpose of the descriptor is to indicate how media streams should flow within a context (who can hear or see whom). By default, all terminations in a context can send and receive media to and from each other. If a different situation is desired, then the topology descriptor is used.

The descriptor comprises a sequence of triples in the form Termination1, Termination 2, association. Such a triple indicates whether there is to be no media flow, one-way media flow, or two-way media flow between the two terminations. These three alternatives are indicated by *isolate*, *oneway*, and *bothway* respectively. The order in which the terminations appear in the triple is important. For example, the triple (T1, T2, oneway) means the T1 can send media to T2, but T2 must not send media to T1.

If more than two terminations are used in a context, then more than one triple is required. For example, if three terminations (T1, T2 and T3) are used, then three triples are needed to describe the association between T1 and T2, the association between T1 and T3, and the association between T2 and T3. Figure 6-11 provides an example of several topologies between three terminations. It is assumed that the topologies are applied in sequence.

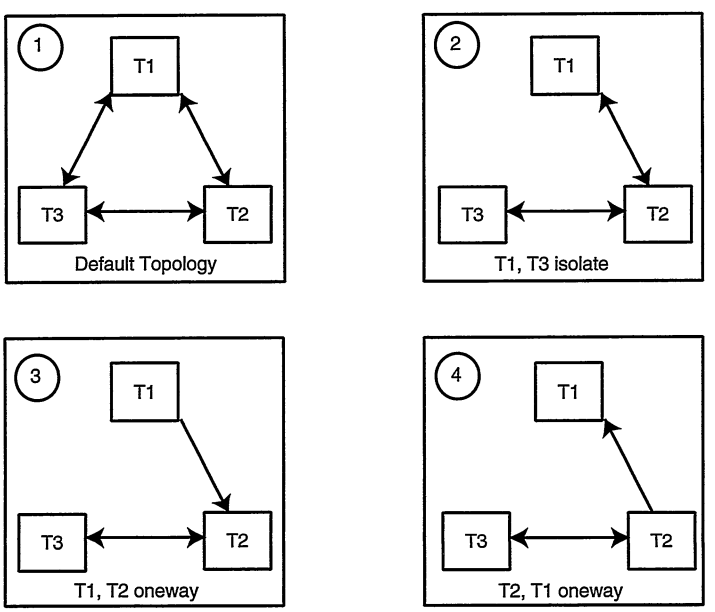


Figure 6-11
Context topologies

The topology descriptor can be a useful tool for implementing services such as call waiting, where the controlling party needs to swap between two other parties. In a conference call situation, this descriptor can facilitate features such as the ability for one or more parties to the call to have a private discussion before returning to the whole group.

The topology descriptor will not be found as a parameter to any of the MEGACO commands. This is because the commands are directed at specific terminations within a context and the topology is a characteristic of a context as a whole, rather than any specific termination. When the descriptor is to appear in a transaction, it appears before any commands for terminations in a particular context. The topology descriptor can only appear in a transaction for a context that already exists.

Packages

In the previous section on the various descriptors, instances occur where a reference is understandably made to specifics that depend on the type of termination in question. Clearly, different types of terminations exist, and the properties and capabilities of those terminations will vary according to type. For example, it is obvious that a termination connected to an analog telephone line will be quite different from a termination connected to a digital trunk. Consequently, it is not practical to define within the protocol itself all the possible termination properties that could exist, all the possible signals that could possibly be sent, all the possible statistics that might apply, and all the possible events that might be detected.

To address the variations between different types of terminations, MEGACO incorporates the concept of packages. Packages are groups of properties, signals, events, and statistics. Within a package, these items are defined and are given identifiers, and the parameters associated with them are specified. A given termination implements a set of packages, and the MGC can audit a termination to determine which packages it supports. Several basic packages are described in the MEGACO specification. Among these are the Tone Detection Package, the DTMF Generator Package, and the Analog Line Supervision Package, to name just a few. The MEGACO spec also provides guidance for the specification of new packages, which should be registered with the IANA.

Packages Descriptor In the previous discussion of descriptors, one descriptor is purposely not mentioned. This is the packages descriptor. It was left out until the concept of packages could be introduced. The

packages descriptor is used only in responses, and it provides the MGC with a list of the packages implemented in a termination.

MEGACO Command and Response Details

Having described the purpose of the various commands and descriptors, it is worth examining the details of how they are put together into transactions and messages. This is done here prior to specifying which descriptors are applicable to each command so that the reader may have a good understanding of how commands and responses are syntactically structured. This knowledge can help to better visualize the various command/descriptor combinations.

Syntax for Commands and Responses

The following sections specify the syntax for each command and response.

Command Syntax As described earlier, commands are applied to terminations within a specific context (or a choose [$] context in some cases). Commands applicable to the terminations within the same context are grouped. Several such groups form a transaction and several transactions form a message. In the text syntax of the protocol, groupings of commands are enclosed by the brackets { and }. Similarly, the contents of the different descriptors are enclosed in brackets and individual descriptors are separated by commas. This structure leads to a syntax that appears as in the following example. The line numbers A to K are not part of the syntax, but they are used just for reference in describing the various components:

```
A.  MEGACO/1 [111.111.222.222]:3456
B.  Transaction = 12345 {
C.       Context = - {
D.          Modify = * {
E.              Events = 7777 {al/of},
F.              DigitMap = Map1 {
G.                  (0|00|[1-7]xxx|8xxxxxxx|Fxxxxxxx|Exx|91xxxxxxxxxx|9011x.) }
H.          }
I.       }
J.  }
```

This example shows a message, using MEGACO version 1, from an entity at IP address 111.111.222.222. The message indicates that the MGC is using port number 3456 for MEGACO messages (line A). This transaction contains only one message, and it has a TransactionID of 12345 (line B).

The transaction relates to only one context: the null context identified by the symbol "-" (line C). Only one command is specified, a Modify command, and it applies to all terminations in this context as indicated by the all (*) wildcard (line D).

The command has two descriptors. The first is an events descriptor with a RequestID value of 7777 (line E). Only one event is to be detected: the off-hook (of) event, which belongs to the Analog Line Supervision package (al) (line E). Note that line E, by having both a right and left bracket, totally describes this descriptor. Note also the comma, which indicates that another descriptor is to be expected.

The next descriptor is the DigitMap descriptor, which specifies a DigitMap name of Map1 (line F). The details of the digit map are given on line G. This line also contains a right bracket, which terminates the DigitMap descriptor. The fact that no subsequent comma is used indicates that no more descriptors are included for this command.

Line H just contains a right bracket, terminating the Modify command. Line I just contains a right bracket, terminating the set of commands applied to this context. Line J contains a right bracket terminating the transaction.

Within the syntax, we know that this is a transaction request, as opposed to a transaction reply or a transaction pending. This is due to the use of the string Transaction on line B. A transaction reply would utilize the string Reply and a transaction pending would utilize the string Pending.

Response Syntax The previous example command leads to a fairly simple response as follows:

```
A2.  MEGACO/1 [111.111.333.333]:5678
B2.  Reply = 12345 {
C2.       Context = - { Modify = *}
D2.  }
```

The same basic structure is applied to the response. In this case, the message is issued from the MG, and consequently the IP address and port numbers are different (line A2). The TransactionID is the same as in the transaction request; however the Reply token is used to indicate that this is a reply (line B2). Line C2 specifies the context, the null context, as in the transaction request. Line C2 also lists the commands the response is issued to (Modify) and the terminations the commands apply to (all of them in this case). No descriptors are used in the response. Line D2 uses a right bracket to indicate the end of the reply.

Responses in the Case of Failure The previous example showed a reply where the command was carried out successfully. Cases will occur, however, when one or more commands cannot be carried out. Situations might also arise when a complete transaction might fail. To accommodate such situations, MEGACO uses the error descriptor described earlier to indicate failure of a command and the reason for the failure.

Let's assume that the MG was completely unable to execute the foregoing command due to an internal error. In that case, the MG would respond as follows:

```
MEGACO/1 [111.111.333.333]:5678
Reply = 12345 {
    Context = - { Modify = * {
                        Error = 500 { Internal Gateway Error}
                }
    }
}
```

By examining this response, we can see that the error being reported is applicable to the Modify command. Equally, the error descriptor might be applied higher up, say, at the level of a transaction if a complete transaction could not be executed. This could arise, for example, in the case of some internal error in the MG, a syntax error in the transaction request, or for any number of reasons.

Add, Modify, and Move Commands The Add, Modify, and Move commands are practically identical in terms of the syntax used. The only syntactical difference between them is in the names of the commands themselves. All three commands include the descriptors media, modem, mux, events, EventBuffer, signals, DigitMap, and audit—all of which are optional. Version 2 of the specification does not actually use the modem descriptor, but it is retained in the command syntax for backwards compatibility with MEGACO version 1. To indicate that items are optional in the ABNF syntax, we enclose them in square brackets ([,]). Therefore, the syntax of the commands can be represented as follows, where the command is Add, Move, or Modify.

```
Command ( TerminationID
          [, MediaDescriptor]
          [, ModemDescriptor]
          [, MuxDescriptor]
          [, EventsDescriptor]
          [, EventBufferDescriptor]
          [, SignalsDescriptor]
```

Motorola - Exhibit 1022 - Page 323

```
        [, DigitMapDescriptor]
        [, AuditDescriptor]
    )
```

Note that this code is a representation of the command and is not exactly how the command will appear as it crosses the interface between the MGC and MG. The exact protocol structure as it crosses the interface is described in the "Command Syntax" section earlier.

Although the three commands are identical in terms of the descriptors that may be used, they obviously have different functions. This fact has an impact on whether wildcards can be used and where. In the Add command, the ContextID can be choose ($), in which case a new context is created. Equally, the TerminationID can take choose ($), in which case a new ephemeral termination is created as a result of the execution of the Add command. The context cannot be the null context, however, since one does not add a termination to the null context. Rather, a termination exists in the null context when it has been subtracted from whatever other context it was in.

For the Modify command, the choose ($) option is not permitted if used either for the TerminationID or for the ContextID. It is not possible to make modifications to terminations that do not exist, and it is not possible to create a new context except with the Add command. It is possible, however, to refer to the null context when issuing a modify command, since it is perfectly valid to modify a termination that exists in the null context.

For the Move command, the null context is not permitted, because the Move command is used only to move terminations from one active context to another. The Move command cannot be used to move a termination to or from the null context. We can use the all (*) wildcard as a TerminationID in the Move command, but we cannot use the choose ($) wildcard.

Add, Modify, and Move Commands' Responses A successful response to a command can also contain a number of descriptors. As is the case for the command itself, all the allowed descriptors in a response to an Add, Modify, or Move command are optional. The response can be represented as follows. Again, the command can be either Add, Modify, or Move. As is the case for the command itself, the modem descriptor is included in version 2 of the protocol only for backwards-compatibility reasons.

```
Command ( TerminationID
          [, MediaDescriptor]
          [, ModemDescriptor]
          [, MuxDescriptor]
          [, EventsDescriptor]
```

```
                [, SignalsDescriptor]
                [, DigitMapDescriptor]
                [, ObservedEventsDescriptor]
                [, EventBufferDescriptor]
                [, StatisticsDescriptor]
                [, PackagesDescriptor]
        )
```

At first glance, with the exception of some changes in the list of allowed descriptors, the response appears very similar to the command, which it is. Within the syntax, however, the two are easily distinguished by the use of the token Transaction when sending a command and the use of the token Reply when responding.

Subtract Command The Subtract command is used to remove a termination from a context. If the termination removed by a Subtract command happens to be the last termination in a given context, then the context itself is deleted. The Subtract command includes only one optional descriptor, the Audit descriptor. The command is represented as follows:

```
Subtract ( TerminationID
                [, AuditDescriptor]
         )
```

Subtract Command Response Although the syntax of the Subtract command differs significantly from that of the Add, Modify, and Move commands, the syntax of the response to a Subtract command is the same as the syntax of a response to an Add, Move, or Modify command:

```
Subtract ( TerminationID
                [, MediaDescriptor]
                [, ModemDescriptor]
                [, MuxDescriptor]
                [, EventsDescriptor]
                [, SignalsDescriptor]
                [, DigitMapDescriptor]
                [, ObservedEventsDescriptor]
                [, EventBufferDescriptor]
                [, StatisticsDescriptor]
                [, PackagesDescriptor]
         )
```

Although all the descriptors in the response to a Subtract command are optional, the statistics descriptor will be returned by default. If the termination happens to represent an RTP session, this provides the MGC with information such as the number of packets sent, the number of packets received, the number of lost packets, the average latency, and so on.

Motorola - Exhibit 1022 - Page 325

AuditValue Command The AuditValue command includes just a single mandatory descriptor, the audit descriptor. The command is used to collect the current settings of a termination's attributes, such as properties and signals:

```
AuditValue ( TerminationID
                    AuditDescriptor
           )
```

AuditValue Command Response

```
AuditValue ( TerminationID
                    [, MediaDescriptor]
                    [, ModemDescriptor]
                    [, MuxDescriptor]
                    [, EventsDescriptor]
                    [, SignalsDescriptor]
                    [, DigitMapDescriptor]
                    [, ObservedEventsDescriptor]
                    [, EventBufferDescriptor]
                    [, StatisticsDescriptor]
                    [, PackagesDescriptor]
           )
```

As is to be expected, the response to the AuditValue command contains a significant number of optional descriptors, each of which can provide information regarding the current state of play at the termination. As is the case for all commands and responses, the modem descriptor should be supported only for backwards-compatibility reasons.

AuditCapabilities Command With the exception of the command name, the syntax of the AuditCapabilities command is identical to that of the AuditValue command. Whereas the AuditValue command returns information regarding the current settings for properties, events, and signals, the AuditCapabilities command returns information on what properties, events, and signals can be supported by a termination:

```
AuditCapabilities ( TerminationID
                           AuditDescriptor
                  )
```

AuditCapabilities Command Response

```
AuditCapabilities ( TerminationID
                           [, MediaDescriptor]
                           [, ModemDescriptor]
                           [, MuxDescriptor]
```

```
                                    [, EventsDescriptor]
                                    [, SignalsDescriptor]
                                    [, ObservedEventsDescriptor]
                                    [, EventBufferDescriptor]
                                    [, StatisticsDescriptor]
                        )
```

Notify Command The Notify command is issued only by an MG to an MGC, and it provides a list of events detected on a termination. As such, the command contains just a single mandatory descriptor: the ObservedEvents descriptor. The command can optionally contain the error descriptor. Within the ObservedEvents descriptor is a RequestID. This matches the RequestID of an Events descriptor previously sent from the MGC to the MG. This parameter enables the MGC to correlate reported events with a particular request to detect and report such events.

```
Notify ( TerminationID
            ObservedEventsDescriptor
              [, ErrorDescriptor]
       )
```

Notify Command Response The only response that will be issued to a Notify command will occur if an erroneous command is made, in which case the error descriptor is used in the response. This approach is the case for all failed commands.

ServiceChange Command This command, which is used at restart, failover of an MG, or handoff from one MGC to another, carries only one mandatory descriptor — the ServiceChange descriptor. The TerminationID may be specific, may be wildcarded using the all wildcard (*), or may have the value Root to indicate the MG as a whole.

```
ServiceChange ( TerminationID
                                ServiceChangeDescriptor
              )
```

ServiceChange Command Response The response to the ServiceChange also includes the ServiceChange descriptor, but the descriptor is optional in the response.

```
ServiceChange ( TerminationID
                                [ServiceChangeDescriptor]
              )
```

Call Setup Using MEGACO

In Figure 6-8, the setup of a call between two gateways using MGCP is examined. Based on the foregoing descriptions of MEGACO commands and responses, we are in a position to describe how the same call establishment would be handled if MEGACO were the control protocol instead of MGCP. The corresponding call flow is shown in Figure 6-12. The overall objective is to establish communication between termination T1 on MG A and termination T4 on MG B, and we assume that the MGC knows from external signaling (such as SIP or SS7) that the connection is required. MG A has an IP address of 311.311.1.1, while MG B has an IP address of 322.322.1.1. The MGC has an IP address of 333.333.1.1 and uses port number 3333 for MEGACO messages.

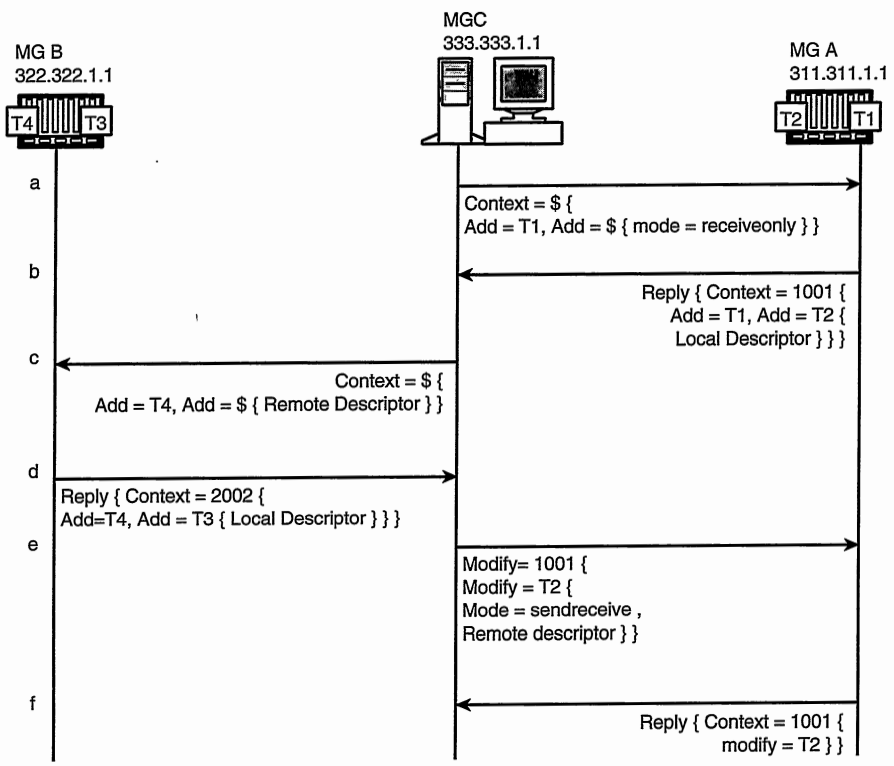


Figure 6-12
Call setup using MEGACO

The figure includes only a representation of the MEGACO messages and commands used, and it does not include a complete syntax of the messages. The complete syntax is given in the following code, along with descriptive text. The syntax assumes the use of MEGACO version 2:

```
Line (a)
MEGACO/2 [333.333.1.1] : 3333
Transaction = 1 {
     Context = $ {
          Add = T1,
          Add = $ {
               Media {
                    Stream = 1 {
                         LocalControl {
                              Mode = receiveonly
                              }
                         Local {
                              v=0
                              c=IN IP4 $
                              m=audio $ RTP/AVP 0
                              v=0
                              c=IN IP4 $
                              m=audio $ RTP/AVP 15
                                }
                          }
                    }
               }
          }
     }
```

The MGC issues a transaction request to MG A using a TransactionId value of 1. The transaction is to be executed on a new context created by the MG, as indicated by the choose ($) wildcard as the ContextID. Two Add commands are used. The first is to add termination T1 to the new context. The second is to add a new termination to the same context. This is indicated by the use of the $ as a TerminationID.

The media descriptor for the media associated with the new termination has a StreamID value of 1, assigned by the MGC. The LocalControl is set to receiveonly, which means that the termination may accept media from the remote end, but not send any.

The MGC is also making two suggestions as to the media properties to be applied by the termination. This can be seen in the use of two session descriptions, with the v= line indicating the boundary between each description. The first is the use of RTP payload type 0 (G.711 μ-law). The second is the use of RTP payload type 15 (G.728). Note the use of the choose wildcard within the SDP descriptions. The wildcard is used for both the IP address and the port number to be used in each of the descriptions. Note also that the o=, s=, and t= lines are not included in the SDP descriptions.

Motorola - Exhibit 1022 - Page 329

Although the SDP specification defines those fields to be mandatory, the MEGACO specification enables the fields to be omitted when SDP is used by MEGACO. The use of a wildcard ($) in the c= line is also a MEGACO-specific modification to SDP and applies only when the MGC is using SDP to suggest media characteristics.

```
Line (b)
MEGACO/2 [311.311.1.1] : 1111
Reply = 1 {
     Context = 1001 {
          Add = T1,
          Add = T2 {
               Media {
                    Stream = 1 {
                         Local {
                              v=0
                              c=IN IP4 311.311.1.1
                              m=audio 1199 RTP/AVP 0
                                }
                               }
                          }
                        }
                         }
          }
```

MG A replies to the transaction using the same TransactionID. It provides the ContextID value of 1001 to indicate the newly created context. It indicates that the addition of termination T1 has been successful, and it also indicates the TerminationID of the newly created termination, T2. Associated with this termination, the MG provides a media description, using the same StreamID as specified by the MGC. The MG also provides a session description as the local descriptor. In this case, the session description indicates voice coded according to G.711 μ-law (RTP payload type 0). Furthermore, the IP address and port for the media stream have been filled in by the MG.

```
Line (c)
MEGACO/2 [333.333.1.1] : 3333
Transaction = 2 {
     Context = $ {
          Add = T4,
          Add = $ {
               Media {
                    Stream = 2 {
                         LocalControl {
                              Mode = sendreceive
                              }
                         Local {
                              c=IN IP4 $
                              m=audio $ RTP/AVP 0
```

```
                    v=0
                },
                Remote {
                    c=IN IP4 311.311.1.1
                    m=audio 1199 RTP/AVP 0
                    v=0
                        }
                }
            }
        }
    }
}
```

The MGC issues a message to MG B, with a TransactionId value of 2. The transaction applies to a new context to be created on MG B, and two terminations are to be added to the new context. The first is termination T4 and the second is a termination that the MG needs to create, as indicated by the use of $ as the TerminationID.

For the new termination, the mode is set to sendreceive, so that the endpoint can both send and receive media. The MGC makes a suggestion as to the local session description that should be applied, indicating RTP payload type 0, but leaves the choice of IP address and port number to MG B. The MGC specifies the remote descriptor in detail. This descriptor happens to be the session description received from MG A for termination T2.

```
Line (d)
MEGACO/2 [322.322.1.1] : 2222
Reply = 2 {
     Context = 2002 {
          Add = T4,
          Add = T3 {
               Media {
                    Stream = 2 {
                         Local {
                              v=0
                              c=IN IP4 322.322.1.1
                              m=audio 2299 RTP/AVP 0
                              }
                         }
                    }
               }
          }
     }
```

MG B replies to the transaction request, using the same value of TransactionID. The MG indicates the ContextID of the newly created context (2002) and also indicates that the two terminations, T4 and T3, have been successfully added to the context. T3 is a newly created termination. The reply also specifies media-related information for T3. Specifically, the reply includes a local descriptor in the form of an SDP description, indicating the

IP address of 322.322.1.1, the port number of 2299, and a media format according to RTP payload type 0.

```
Line (e)
MEGACO/2 [333.333.1.1] : 3333
Transaction = 3 {
    Context = 1001 {
          Modify = T2 {
               Media {
                    Stream = 1 {
                         LocalControl {
                              Mode = sendreceive
                              }
                         Remote {
                              v=0
                              c=IN IP4 322.322.1.1
                              m=audio 2299 RTP/AVP 0
                                   }
                         }
                    }
               }
          }
    }
```

The MGC issues a new message to MG A. The message has one transaction request with a TransactionID value of 3, and it includes a Modify command for termination T2 in context 1001. The Modify command first changes the local control to send-receive so that the termination can send media to the far end. Of course, the termination needs a session description for the far end so that it has an IP address and port number to which to send the media. This is provided in the remote descriptor, which contains the session description received by the MGC from MG B.

```
Line (f)
MEGACO/2 [311.311.1.1] : 1111
Reply = 3 {
    Context = 1001 {
          Modify = T2
                   }
          }
```

MG A sends a positive response to the MG to confirm that the modification has been performed. At this stage, termination T1 is in the same context as termination T2 and the two can share media. Terminations T2 and T3 have exchanged session descriptions and can share media. Terminations T3 and T4 are in the same context and can share media. The net effect is that a two-way path is formed between terminations T1 and T4, which means that the original objective has been achieved.

Motorola - Exhibit 1022 - Page 332

Interworking Between MEGACO and SIP

Interworking between MEGACO and SIP works in very much the same manner as interworking between MGCP and SIP. Once a local descriptor is available at the gateway where the call is being originated, then this can be passed in a SIP INVITE as a SIP message body. Upon acceptance of the call at the far end, the corresponding session description is carried back as a SIP message body within the SIP 200 (OK) response and is passed to the originating side gateway. Once the gateway on the originating side has acknowledged receipt of the remote session description, the MGC can send a SIP ACK to complete the SIP call setup.

No fundamental difference exists between MGCP and MEGACO as far as the manner in which SIP interworking is achieved. This is understandable, since MEGACO and MGCP deal only with MGC-MG control and both handle call setup in quite similar ways.

Motorola - Exhibit 1022 - Page 333

CHAPTER 7

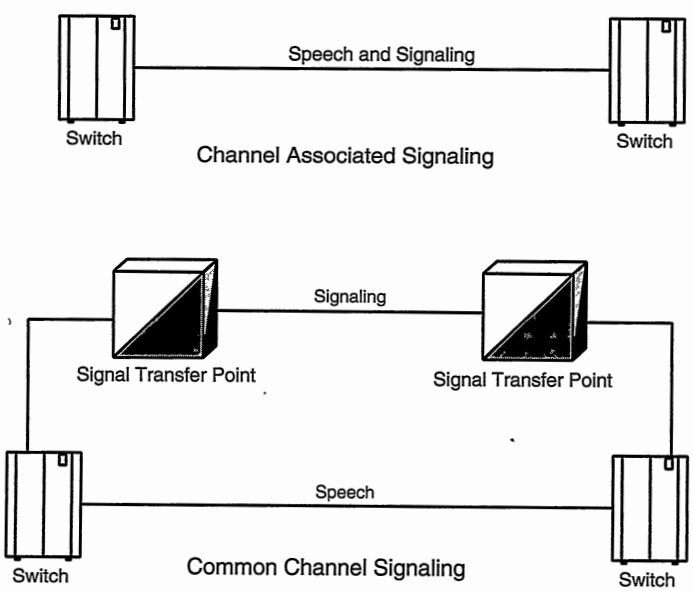
VoIP and SS7

For a long time, signaling in circuit-switched networks was such that the signaling related to a particular call followed the same path as the speech for that call. This approach is known as *Channel Associated Signaling* (CAS), and the technique is still widely deployed today. Examples include the R1 *Multifrequency* (MF) signaling used in North America and the R2 *Multifrequency Compelled* (MFC) signaling used in many other countries. Although it is still widely used in circuit-switched networks, CAS is considered old technology. The prevailing technology in newer circuit-switched networks is *Common Channel Signaling* (CCS).

CCS involves the use of a separate transmission path for call signaling compared to the bearer path for the call itself, as shown in Figure 7-1. This separation enables the signaling to be handled in a different manner to the call itself. Specifically, other nodes in the network may analyze the signals and take action based on the content of the signals, without needing to be involved in the bearer path. Furthermore, CCS enables signaling messages to be exchanged in cases where no call is to be established at all. Imagine, for example, dialing a star-code to activate some feature. In such a scenario, the local switch might communicate with a service node to activate the feature in the network without actually establishing a call to the service node.

The standard for CCS today is *Signaling System 7* (SS7). Various versions of it are deployed all over the world. In fact, SS7 could be considered the ultimate signaling standard in telecommunications. Anyone who wants to implement a circuit-switched network and who does not implement SS7

Figure 7-1
CAS versus CCS



is not really playing in the big leagues and cannot really call the network carrier-grade.

Why is SS7 so important? SS7 enables a wide range of services to be provided to the end-user. These services include Caller-ID, toll-free calling, call screening, number portability, and others. In fact, SS7 is the foundation for *Intelligent Network* (IN) services.

Given that SS7 is so prevalent and so important to telecommunications, it follows that any *Voice over IP* (VoIP) carrier that wants to interwork with the *Public Switched Telephone Network* (PSTN) must support SS7 simply in order to communicate with circuit-switched networks. If a VoIP network is to be considered carrier-grade, then it must support SS7. The capability to speak the same language as the PSTN is not the only reason, however. Customers who use VoIP networks will expect the same types of features that are available in existing networks and will expect them to work in the same way. In order to compete effectively, a new carrier has no alternative but to match (if not exceed) the features and functionality already available.

To do so, a carrier can utilize the capabilities of SS7 or try to create a totally new way of providing such services. Utilizing SS7 is the obvious choice. This does not mean that the carrier is limited to what SS7 can provide—far from it. In fact, the *Internet Protocol* (IP) provides the opportunity to offer a wide range of new services. Customers will still expect to see existing services such as toll-free calling, however, and a new carrier has little alternative but to use SS7 to provide many of those services.

This chapter provides a brief introduction to SS7, followed by a description of the technical solutions that can be deployed to support interworking between VoIP networks and SS7. The interworking that needs to be provided has two aspects. First, SS7 needs to be supported by applications in an IP environment even though SS7 was not originally designed for such an environment. Second, different network configuration issues can arise, depending on the architecture of the VoIP network itself. For example, the VoIP network may follow the H.323 model. Alternatively, the VoIP network may use a softswitch architecture based on the *Session Initiation Protocol* (SIP) and the *Media Gateway Control Protocol* (MCGP) or MEGACO. To some degree, how the interworking is done is influenced by the network architectures in question.

The SS7 Protocol Suite

SS7 is not a single protocol per se; rather, it is a suite of protocols operating in concert. In the same way that the *Transmission Control Protocol* (TCP)

or the *User Datagram Protocol* (UDP) use the services of the IP in a stack arrangement, SS7 also defines a stack that has certain similarities with the *Open Systems Interconnection* (OSI) seven-layer model. Figure 7-2 shows the SS7 stack.

The Message Transfer Part (MTP)

The three lower levels of the SS7 stack comprise the *Message Transfer Part* (MTP). This is the part of the protocol responsible for getting a particular message from the source to the destination. Level 1 maps closely to layer 1 of the *Open System Interconnection* (OSI) layers. Level 1 handles the issues related to the signals on the physical links between one signaling node and another. These links are known as signaling links and typically operate at 56 Kbps or 64 Kbps.[1] At a node that supports SS7, the physical termination of a signaling link is known as a *signaling terminal*.

MTP level 2 (MTP2) deals with the transfer of messages on a given link from one node to another. Signaling messages from higher-layer protocols are passed down the stack to MTP2, which packages them for transmission on a given signaling link and takes care of getting the message across that link. As well as providing functions for passing messages on behalf of MTP users, MTP2 also has some messages of its own, such as the *Link Status Signal Unit* (LSSU) and the *Fill-In Signal Unit* (FISU). The LSSU is used between two ends of a link to ensure alignment and correct link functioning, and FISUs are sent when nothing else can be sent. The FISU can also carry acknowledgments of received messages.

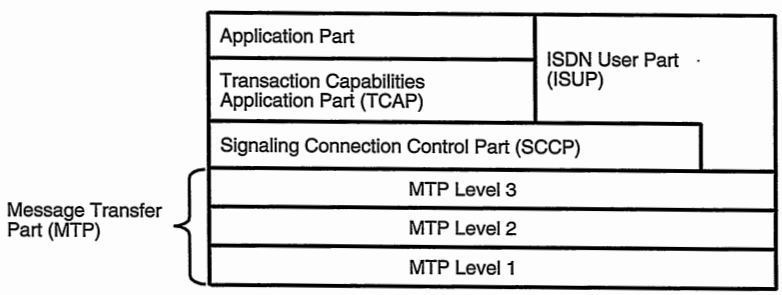


Figure 7-2
The SS7 protocol stack

[1]This statement applies only to narrowband SS7 links. High-speed links also operate over 1.5 Mbps or 2 Mbps interfaces. High-speed links operate over *Asynchronous Transfer Mode* (ATM).

Motorola - Exhibit 1022 - Page 337

MTP level 3 (MTP3) deals with the management of messaging on the signaling network as a whole. For a message destined for a particular signaling destination, many different paths could be taken from the source to the destination. These paths could include a direct signaling link if one exists or a path via a number of different intermediate nodes, known as *signal transfer points* (STPs). MTP3 takes care of determining which outgoing link should be used for a particular message. Therefore, MTP3 includes functions for the mapping of signaling destinations to various signaling link sets. MTP3 manages load-sharing across different signaling links and also handles the rerouting of signaling in the case of link failure, congestion, or the failure of another node in the signaling network. In order to help manage the overall signaling network, MTP3 includes a complete signaling network management protocol for ensuring the proper operation of the SS7 network.

Furthermore, MTP3 provides a number of services to the protocol layer above it. These services involve the transfer of messages to and from the upper layer, indicating (for example) the availability or unavailability of a particular destination and signaling network status (such as congestion). These services are provided through the primitives MTP-Transfer request, MTP-Transfer indication, MTP-Pause indication, MTP-Resume indication, and MTP-Status indication.

ISDN User Part (ISUP) and Signaling Connection Control Part (SCCP)

Above MTP, two main alternatives exist: the *ISDN User Part* (ISUP) and the *Signaling Connection Control Part* (SCCP).[2] Let's deal with ISUP first.

Strictly speaking, ISUP is a signaling protocol used to provide services to *Integrated Services Digital Network* (ISDN) applications. In reality, ISUP is most often used as the protocol for setting up and tearing down phone calls between switches and is the most commonly used of the SS7 applications. Examples of ISUP messages include call setup messages such as the *Initial Address message* (IAM) that is used to initiate a call between two switches, the *Answer message* (ANM) that is used indicate that a call has been accepted by the called user, and the *Release message* (REL) that is used to

[2]Other alternatives exist, most notably the *Telephony User Part* (TUP). This can be considered a simpler version of ISUP. Numerous national variants of TUP are available.

initiate call disconnection. ISUP is a connection-oriented protocol, which means that it relates to the establishment of connections between users. ISUP messages between the source and destination are related to a bearer path between the source and destination, although the path of the messages and the path of the bearer might be very different. In terms of mapping to the OSI model, ISUP maps to layers 4 through 7.

SCCP provides both connection-oriented and connectionless signaling, although it is most often used to provide connectionless signaling. Connectionless signaling refers to signaling that is passed from one switch or network element to another without the need to establish a signaling connection. In other words, connectionless signaling means that information is simply sent from the source to the destination, without the prior establishment and subsequent release of a dedicated signaling relationship. For example, consider a mobile phone user who roams away from his home network. When the user arrives at the visited network, it is necessary for the visited network and home network to communicate so that the visited network can learn a little about the subscriber and so that the home network can know where the subscriber is. Such signaling is connectionless and it uses the services of SCCP.

SCCP also provides an enhanced addressing mechanism to enable signaling between entities, even when those entities do not know each other's signaling addresses (known as point codes). This addressing is known as *global title addressing*. Basically, it is a means whereby some other address, such as a telephone number, can be mapped to a point code either at the node that initiated the message or some other node between the originator and destination of the message.

The *Transaction Capabilities Application Part* (TCAP) enables the management of transactions and procedures between remote applications, such that operations can be invoked and the results of those operations can be communicated between the applications. TCAP is defined for connectionless signaling only. TCAP provides services to any number of application parts. Common application parts include the *Intelligent Network Application Part* (INAP) and the *Mobile Application Part* (MAP).



SS7 Network Architecture

SS7 is purely a signaling protocol that enables services and features in the telephony network. The SS7 network, however, should be considered as a separate network in its own right for a number of reasons. First, SS7

involves the separation of signaling from the bearer (and in some cases, there might be no bearer at all). Second, SS7 involves a number of network elements that deal only with signaling itself and that are tied together in a network of their own. Consequently, when we think about SS7 in the context of interworking between circuit-switched networks and VoIP networks, we should think in terms of three interworking networks, as shown in Figure 7-3. The VoIP system and the circuit-switched network pass signaling via the SS7 network and pass media directly to and from one another.

Figure 7-4 depicts a typical SS7 network arrangement. The two switches shown do not communicate signaling to each other via direct paths. Instead, the signaling is carried via one or more STPs, typically arranged in a quad. This configuration serves several purposes. To start, it avoids the need for direct signaling links between all endpoints (such as switches), meaning that a fully meshed signaling network is not required. Instead, a separate, more compact signaling network can be created. Second, the quad arrangement ensures great robustness. Each switch will be connected to two STPs, and multiple paths exist between the different STPs. Thus, the failure of a single link will never result in a complete loss of signaling capability between different endpoints.

In fact, SS7 networks in general are extremely robust. The design of SS7 is such that very reliable signaling networks can be built, and the complete failure of a node or of a number of signaling links does not mean catastrophe for the network. As far as the applications are concerned, once information is sent to the SS7 stack, it is as good as delivered to the ultimate destination. SS7 really is that reliable—and that fast.

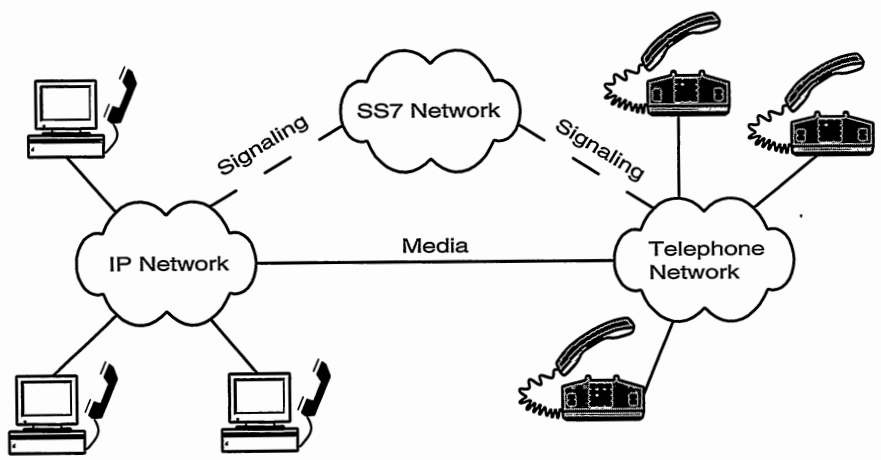


Figure 7-3
Interworking between IP, telephony, and SS7 networks

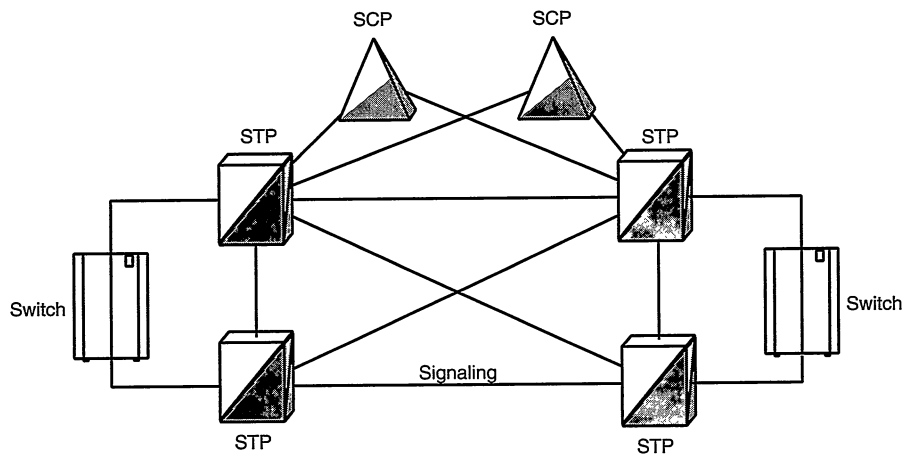


Figure 7-4
SS7 network

Signaling Points (SPs)

Each node in an SS7 network is a *signaling point* (SP) and has one or more signaling addresses, each of which is known as a *signaling point code* (SPC). SPs that are directly connected are said to be adjacent and the links between such adjacent SPs are known as signaling links. Each of these is typically a 56 Kbps or 64 Kbps link. Several signaling links might exist between two adjacent signaling points for capacity or security reasons, and the set of links is not surprisingly called a linkset. More formally, a linkset is that group of signaling links directly connecting two SPCs.

Signal Transfer Point (STP)

The function of an STP is to transfer messages from one SPC to another. Basically, it works as depicted in Figure 7-5. Let us assume that Switch A, with SPC value 1, wants to send a message known as a *message signal unit* (MSU), to Switch B, which has an SPC value of 2. Let's also assume that no signaling link exists between them, but that they are both connected to an STP with SPC value 3. In that case, the routing tables at Switch A could be set up to send the MSU to the STP. The MSU contains the destination signaling address of Switch B, SPC value 2. When the message is received at the STP, the STP first checks to see if the message is destined for itself by checking the destination address. Given that the message is not, the STP

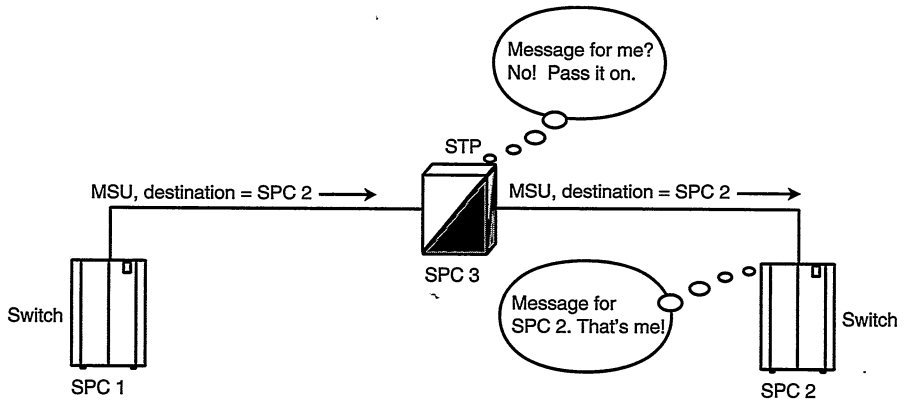


Figure 7-5
STP function

then checks its routing tables to see where messages for SPC 2 should be sent. The STP determines that such MSUs should be sent on the link to Switch B. When the MSU arrives at Switch B, the same process is repeated, except that Switch B recognizes that the MSU is destined for it and sends the content of the message to the appropriate signaling application.

Service Control Point (SCP)

A *service control point* (SCP) is a network entity that contains additional logic and that can be used to offer advanced services. To use the service logic of the SCP, a switch needs to contain functionality that will enable it to act upon instructions from the SCP. In such a case, the switch is known as a *service switching point* (SSP). If a particular service needs to be invoked, the SSP sends a message to the SCP asking for instructions. The SCP, based upon data and service logic that is available, will tell the SSP which actions need to be taken.

A good example of such a service is a call to a toll-free 800 number. When a subscriber dials such a number, the SSP knows that it needs to query the SCP for instructions (which it does). The SCP contains translation information between the dialed 800 number and the real number to which the call should be sent. The SCP responds to the SSP with a routable number, which the SSP uses to route the call to the correct destination. Note that the signaling between the SSP and the SCP is connectionless signaling. Hence, this signaling uses the services of SCCP. In fact, this type of application uses the services of TCAP, which in turn uses the services of SCCP.

Message Signal Units (MSUs)

As mentioned, the messages that are sent in the SS7 signaling network are known as MSUs. Each MSU has the format shown in Figure 7-6. The *Service Information Octet* (SIO) contains the service indicator, which indicates the upper-level protocol to which the message applies (such as SCCP or ISUP). The SIO also contains the Subservice field, which indicates the signaling numbering plan in use. We will discuss this topic in more detail shortly. The *Signaling Information Field* (SIF) contains the actual user information being sent.

If we look at the details of the SIF, then an MSU has one of the formats shown in Figure 7-7, depending on whether the *American National Standards Institute* (ANSI) version of SS7 or the *International Telecommunication Union—Telecommunications Standardization Sector* (ITU-T) version of SS7 is being used. Of particular importance is the routing label, which contains two very important fields: the *destination point code* (DPC) and the *origination point code* (OPC). These are the signaling addresses of the

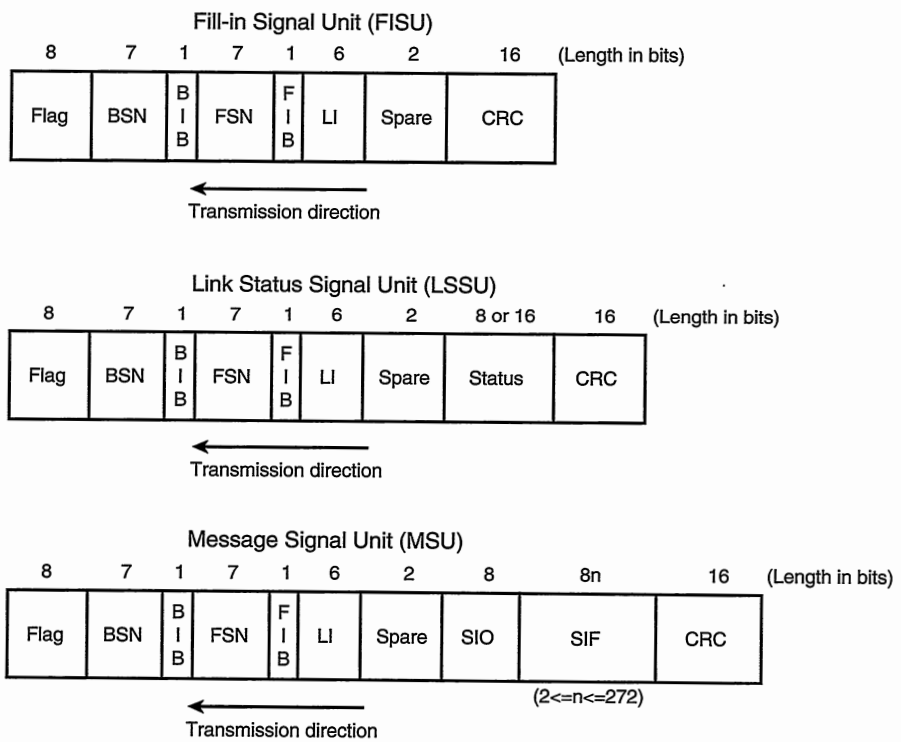


Figure 7-6
SS7 messages

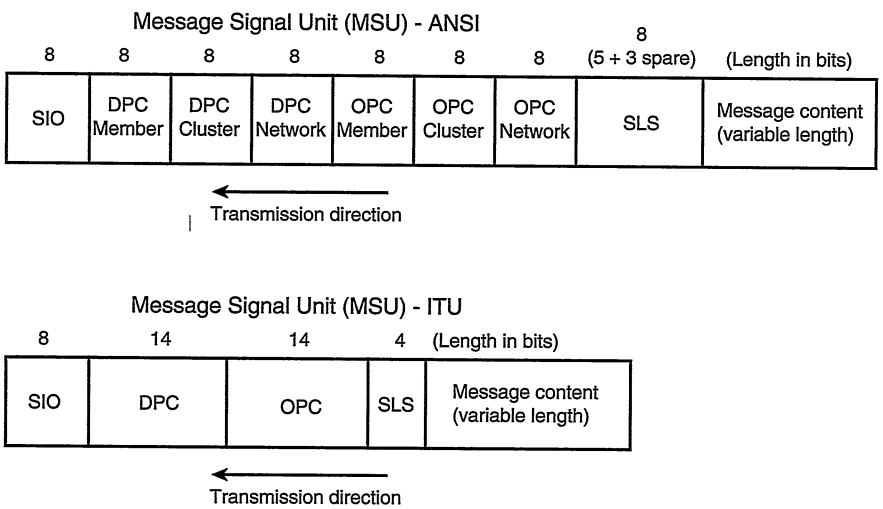


Figure 7-7
SS7 message format

SIO = Service Information Octet
SLS = Signaling link selection
The ANSI SLS is 8 bits, but historically was 5 bits with 3 bits to spare.

destination of the message and the originator of the message respectively. The *Signaling Link Selection* (SLS) field also exists, which indicates the particular signaling link to be used for carrying this message between the two nodes in question.

SS7 Addressing

As mentioned, signaling entities in an SS7 network are addressed by SPCs. A quick look at Figure 7-7 shows that unfortunately different versions of SS7 exist, with different point code structures. For example, in North America, point codes are 24 bits long, whereas in most other countries, point codes are 14 bits long. A notable exception is China, where point codes are 24 bits long.

In North America, point codes have a hierarchical structure, with the format network-cluster-member, each of eight bits. The operator of a very large network would be allocated a network code, enabling that operator to have about 65,000 different SPCs.

Given that different SS7 formats and addressing schemes exist, how do SS7 entities in different countries communicate? The answer is through

international *signaling gateways* (SGs). These gateways are signaling points that support the national SS7 variant at one side and the ITU-T international variant at the other. This fact of both national and international variants brings us back to the Subservice field mentioned earlier. Recall that this field is used to indicate the signaling numbering plan in use. Four variants exist: National, National Spare, International, and International Spare. A network element that acts as an international gateway will have at least two point codes, one a national point code and one an international point code. Messages to and from other nodes in the national network will be addressed with the national point code, and messages to and from other international nodes will use the international point code. Thus, a hierarchy exists in the SS7 network whereby messages between countries must go from the national signaling plane to the international signaling plane and back down to the national signaling plane in the destination country, as shown in Figure 7-8.

ISUP

In regular telephony, ISUP is the most often used SS7 application. ISUP is the protocol utilized for the establishment and release of telephone calls. Figure 7-9 shows a typical call establishment and release.

The call begins with the IAM, which contains information about the called number, the calling number, the transmission requirement (typically 64 Kbps), the type of caller (ordinary, operator, payphone, and so on), and other information, such as whether a satellite link has been included in the call so far.

Upon receipt of the IAM at the destination switch, an *Address Complete Message* (ACM) is returned. This message indicates that the call is through-connected to the destination. The ACM causes a one-way audio path to be opened from the destination switch to the originating switch so that the

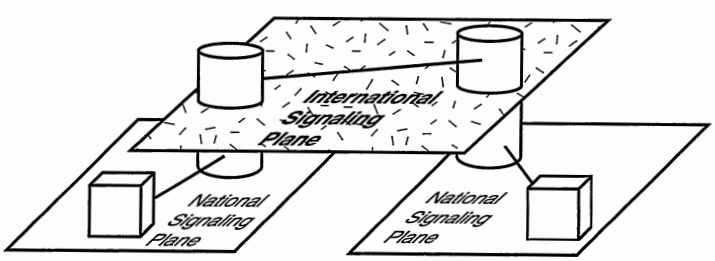


Figure 7-8 International signaling

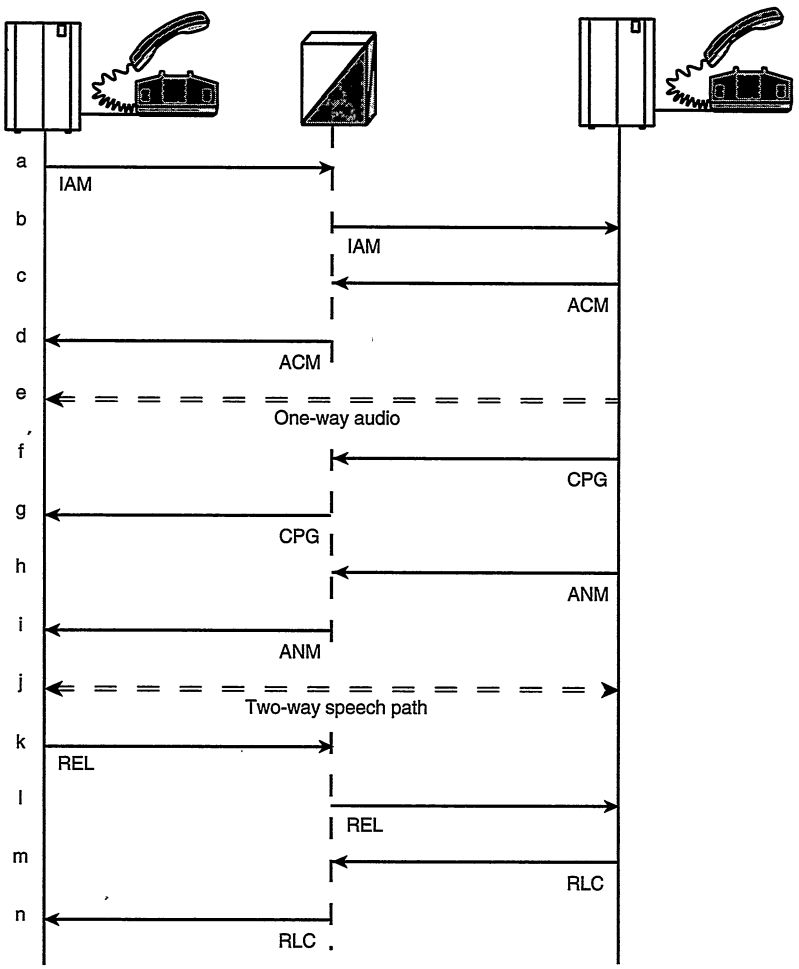


Figure 7-9
ISUP call establishment and release

caller can hear a ring-back tone. Note that the tone is generated at the destination end.

Strictly speaking, the ACM is an optional message. Although it is returned in most calls, that is not always the case. Situations can exist where an ACM is not returned. If it is not returned, the caller does not hear a ring-back tone at all, and it appears as though the call is answered immediately. This happens quite frequently on calls to toll-free numbers, particularly if the call is answered automatically.

After the ACM, there might be one or more *Call Progress* (CPG) messages. The CPG is an optional message and is used to provide additional information to the calling switch regarding the handling of the call.

Once the called party answers, an *Answer Message* (ANM) is returned. This message typically has two purposes. The first is to open the transmission path in both directions so that the parties can converse. The second is to instigate charging for the call, since charging for most calls begins when the call is answered.

After conversation, one of the parties hangs up. This causes a *Release* (REL) message to be sent to the other end. The end that receives the REL responds with a *Release Complete* (RLC) message. At this point, the call is complete.

Note that the signaling passes through one or more STPs, whereas the speech takes a more direct transmission path from A to B. That situation leads to one important question. Given that there might be many simultaneous calls between switch A and switch B, how does the ISUP signaling differentiate between calls? In particular, how is a given ISUP message correlated with a given speech path between the switches? The answer is the *Circuit Identification Code* (CIC), which is contained within each ISUP message. For an ISUP message between two switches, the CIC indicates the specific trunk between the switches to which the message applies. In fact, a given circuit between two switches is completely identified by the combination of OPC, DPC, and CIC. The CIC can be seen in the ISUP message formats shown in Figure 7-10. The figure also serves to further highlight the differences between different SS7 variants.

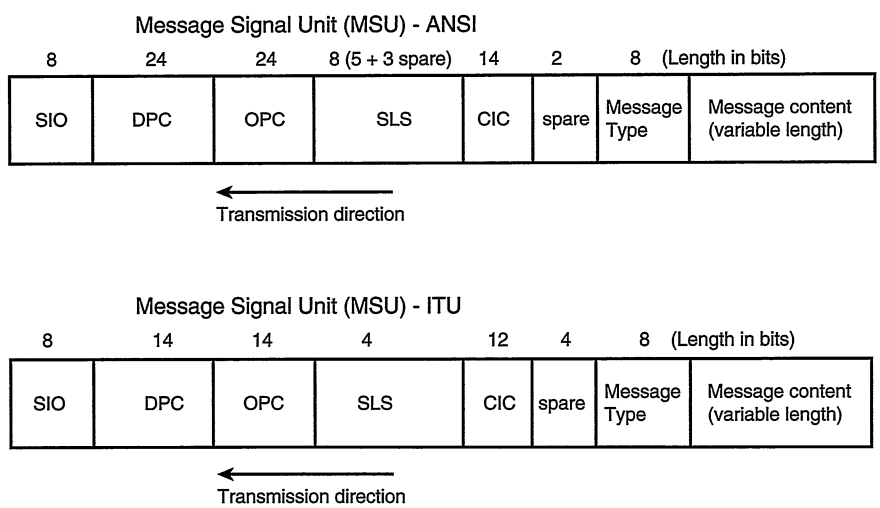


Figure 7-10
The ISUP message format

SIO = Service information octet
SLS = Signaling link selection

Performance Requirements for SS7

Given that VoIP networks need to interwork with the PSTN, it is clear that VoIP networks need to speak SS7 (at least to the outside world). Not only should VoIP networks support the messages of SS7, however, but they must also support the performance requirements specified for SS7. For example, for MTP, Bellcore[3] specification GR-246-Core states that a given route set should not be out of service for more than 10 minutes per year. Other requirements are that no more than 1×10^{-7} messages should be lost, and no more than 1×10^{-10} messages should be delivered out of sequence. In ISUP, numerous timing requirements must be met. For example, a 2-second timer is initiated when a continuity check request is sent, requiring that a continuity check tone be returned within that time.

Imagine a VoIP network that operates as a long-distance network, as depicted in Figure 7-11. This network receives an IAM from a network at one side of the country, processes the information, and determines that the call should be sent to a network at the other side of the country. The VoIP network sends an IAM to the destination network, which responds with an ACM. The VoIP network must process the ACM and send it to the originating network. This is exactly the same process as is performed by a traditional long-distance network.

If we think about the amount of internal signaling that goes on in an H.323 network or even in a softswitch network, however, we realize that a lot of work is done between the passing of external SS7 messages. Furthermore, if we think about the postdial delay of today's telephone networks (a

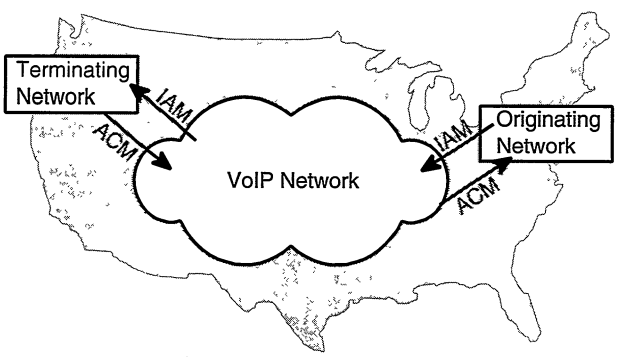


Figure 7-11
Long-distance VoIP network

[3]Bellcore is now known as Telcordia. It is an organization that has written many technical standards, most of which are adopted by North American network operators.

second or two), we realize just how fast this work needs to be done. The point is that a VoIP network using SS7 must meet the stringent requirements that are already met by today's circuit-switched networks. The issue, then, is how to make sure that VoIP networks can emulate the signaling performance of SS7. Fortunately, many groups have been working on this issue. In particular, the *Signaling Transport* (Sigtran) group of the *Internet Engineering Task Force* (IETF) has made great progress in this area. The following sections of this chapter are devoted to describing the architectures and solutions developed within Sigtran.

Sigtran

Sigtran is a working group within the IETF dedicated to addressing the issues regarding the transport of signaling within IP networks. In particular, the group addresses the issues related to signaling performance within IP networks and the interworking with other networks, such as the PSTN.

In order to better understand the issues to be addressed, let us again consider the scenario depicted in Figure 7-11. Let's assume that the VoIP network has a softswitch architecture. Therefore, the VoIP network comprises a number of *media gateways* (MGs), SGs, and *media gateway controllers* (MGCs)/call agents. The purpose of the SGs is to connect directly with signaling points or STPs of the external SS7 network. The function of the MGs and MGCs is as described in Chapter 6, "Media Gateway Control and the Softswitch Architecture." Let's assume for now that the call agents communicate with each other using SIP and that they communicate with MGs using MEGACO.

Given that the MGC is the call control entity within the network, it follows that ISUP signaling received at an SG must be passed to the MGC so that the MGC can control call establishment and call release appropriately. If we assume a straightforward call from a trunking gateway at one side to a trunking gateway at the other side, then the scenario would be similar to Figure 7-12.

The carriage of ISUP signaling information from an SG to an MGC or from an MGC to an SG is indicated in the Figure 7-12 with the letters "IP." For example, an IAM carried in the IP network is denoted "IP IAM." Note that the content of the message does not change since it is assumed that the MGC contains a standard ISUP application. The only difference is that the ISUP messages are carried to and from the MGC over IP rather than over standard MTP. Thus, all that the SG does is to take the ISUP message from

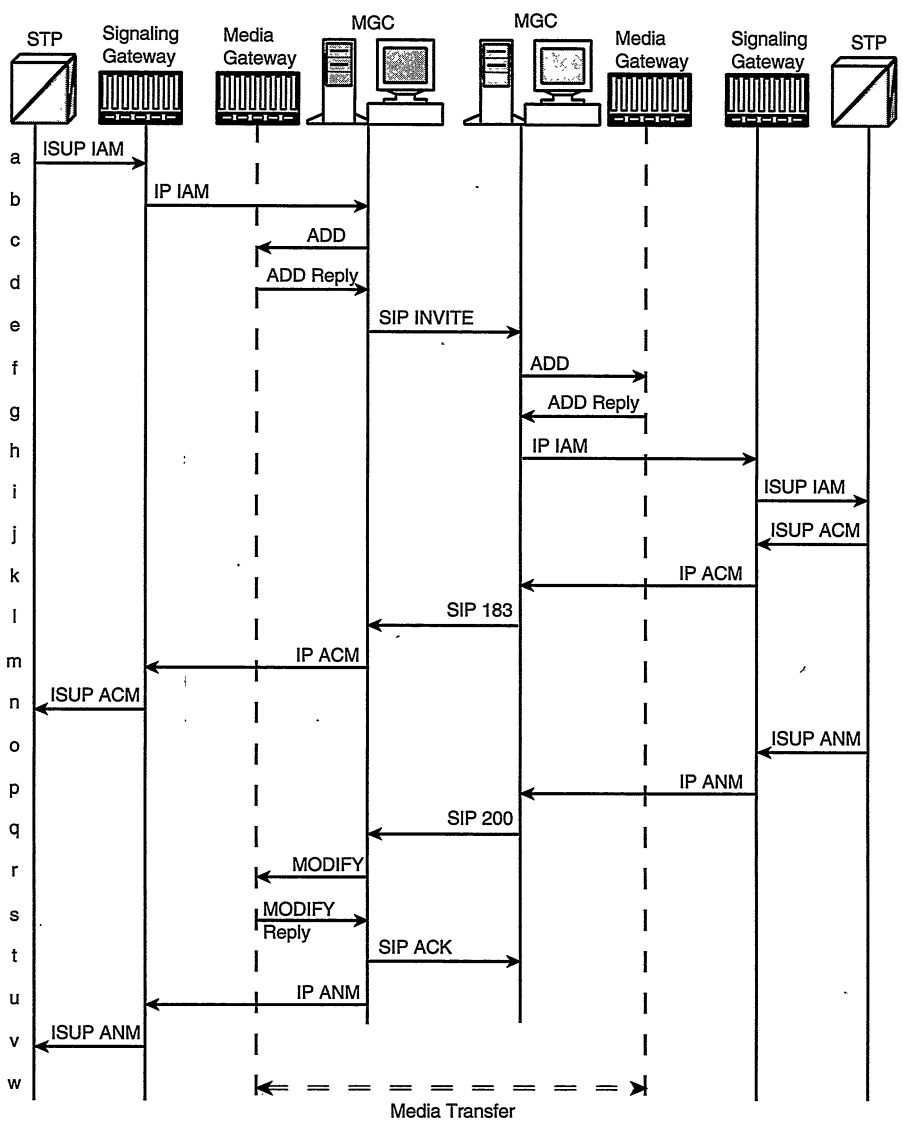


Figure 7-12
SIP/MEGACO/ISUP interworking

the SS7 network and get it to the appropriate entity in the IP network. This task might sound simple—just a simple translation from point code to IP address, right? Unfortunately, things are rarely that simple.

Certainly, point code to IP address translation is one of the issues to be addressed, but we also face the issue of which transport (layer 4) protocol to use for carrying the messages. Furthermore, using a standard SS7

application such as ISUP in an IP environment is also an issue. How can we deploy such an application that expects certain services from lower layers such as MTP when those lower layers do not exist in the IP network?

For the transport protocol, the ISUP messages must be carried in the IP network with the same speed and reliability as in the SS7 network. In some implementations, proprietary mechanisms are used for getting information from an SG to a call agent. This approach requires, however, that the MGC and SG be sourced from the same vendor, thereby reducing the network operator's vendor selection choices. In some other implementations, the ISUP message is simply packaged as a UDP or TCP packet and sent from the SG to the call agent.

This approach can have problems, however. First, the strict performance requirements should be met with regard to message loss and message sequencing. These requirements lead us away from UDP, because it is inherently unreliable. We now have TCP as an alternative, but TCP is not suitable either because it cannot always meet the strict timing requirements of SS7. Although TCP can ensure the accurate in-sequence delivery of messages, it is not particularly fast. TCP also suffers from head-of-line blocking. Therefore, something else is needed.

In order to address these issues, the Sigtran working group has prepared a number of documents describing the issues and solutions. One such document is RFC 2719, "Framework Architecture for Signaling Transport," which describes an overall approach and methodology for signaling transport within IP networks.

Sigtran Architecture

The Sigtran architecture is defined in RFC 2719. The architecture uses the signaling components shown in Figure 7-13. Signaling over standard IP uses a common transport protocol that ensures reliable signaling delivery, and it uses an adaptation layer that supports specific primitives as required by a particular signaling application. In other words, the common signaling transport makes sure that messages are delivered error free and in sequence regardless of the vagaries of the underlying IP network. The adaptation layer provides an interface to the upper-layer protocols and applications such that those applications do not realize that the underlying transport is IP rather than traditional transport, such as MTP.

Although most of the adaptation layers fulfill SS7-related functions, the *Stream Control Transmission Protocol* (SCTP) (which corresponds to the common signaling transport of Figure 7-13) is a generic signaling transport

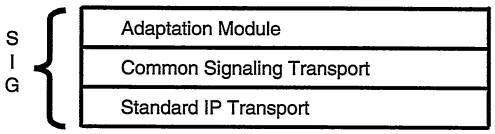


Figure 7-13
Sigtran signaling transport components

protocol, which means that SCTP can also be used to reliably carry other types of signaling traffic. Therefore, Sigtran also includes adaptation layers that enable non-SS7 signaling to be carried on SCTP.

If we apply the Sigtran architecture to a situation where an SG is connected to the SS7 network on one side and to an MGC on the other, then the transport of ISUP signaling would appear as in Figure 7-14. Within the SG, the *Nodal Interworking Function* (NIF) is the function responsible for interworking between the SS7 network on one side and the IP network on the other.

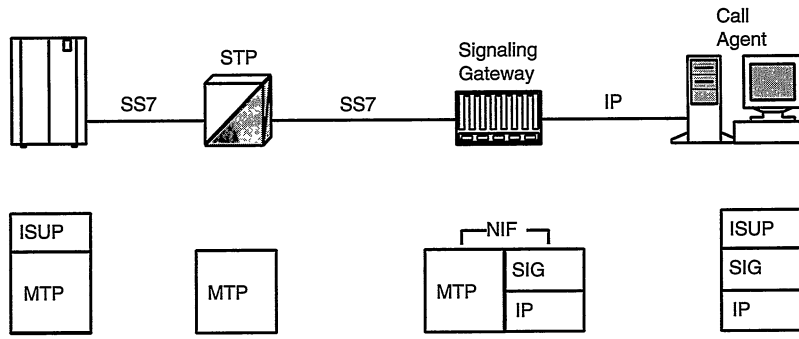
Figure 7-15 shows the Sigtran architecture in more detail. SCTP corresponds to the common signaling transport shown in Figure 7-13. SCTP ensures the error-free, in-sequence delivery of user messages, enables fast delivery of messages, and avoids head-of-line blocking. Therefore, SCTP is a more suitable protocol than TCP. Moreover, SCTP supports network-level fault tolerance, something critical for carrier-grade network performance. SCTP is described in more detail a little later in this chapter.

Before delving into the details of SCTP, however, let's first take a quick look at the other components of the Sigtran architecture. These components are a number of adaptation layers that provide the interface between SCTP and upper-layer protocols. Several such components exist, most of which are SS7 specific, but two of which are related to other types of signaling. In the following sections, more focus is applied to those components that support ISUP interworking, because ISUP is the most commonly used SS7 application and is the protocol with which VoIP networks must most often interwork.

As mentioned, a number of adaptation modules operate on top of SCTP. These include the following:

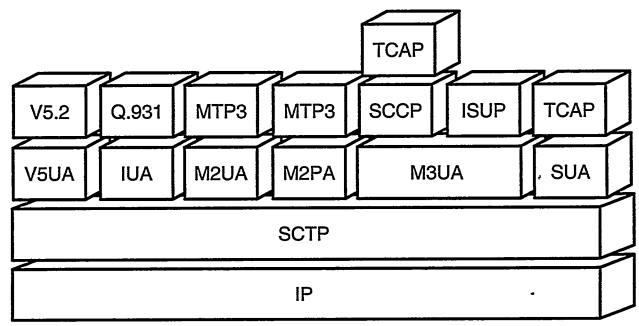
- The *SS7 MTP2-User Adaptation Layer* (M2UA) provides adaptation between MTP3 and SCTP. M2UA provides an interface between MTP3 and SCTP such that standard MTP3 may be used in the IP network without the MTP3 application software realizing that messages are being transported over SCTP and IP instead of MTP2. For example, a standard MTP3 application implemented at the MGC could exchange MTP3 signaling network management messages with the external SS7

Figure 7-14
ISUP transport to MGC



NIF = Nodal Interworking Function

Figure 7-15
Sigtran protocol stack



network. In the same manner that MTP2 provides services to MTP3 in the SS7 network, M2UA provides services to MTP3 in the IP network. M2UA has a registered port number of 2904.

The usage of M2UA is depicted in Figure 7-16. In this scenario, two SGs provide an interface to the outside SS7 network. Both are connected to an MGC. On the MGC side of the SGs, we have M2UA over SCTP over IP, whereas on the SS7 side, we have the standard SS7 MTP. At the MGC, we have a standard MTP3 operating over M2UA and IP. In a regular SS7 network, MTP3 utilizes the services of MTP2. In the scenario depicted in Figure 7-16, however, the MTP3 at the MGC utilizes the services of the MTP2 located at the SG without realizing that it is not local. The function of M2UA is to provide transparent access from the standard MTP3 at the call agent to the standard MTP2 at the SG. In the example of Figure 7-16, the MTP3 application at the MGC can receive MTP3 signaling network management messages such as *Transfer Allowed* (TFA) and *Transfer Prohibited* (TFP). MTP3 can

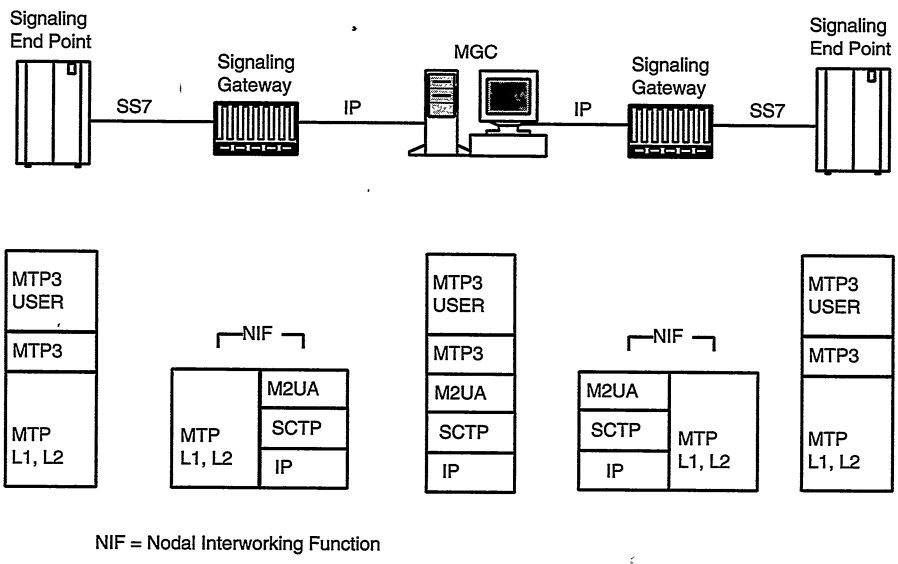


Figure 7-16
M2UA usage in SG to MGC applications

NIF = Nodal Interworking Function

use that information in determining how to route messages from a higher-level MTP3 user, such as ISUP.

- The *SS7 MTP2 Peer-to-Peer Adaptation Layer* (M2PA) provides adaptation between MTP3 and SCTP. As is the case for M2UA, the MTP3 layer at an IP node (such as an MGC) communicates with M2PA as if it were regular MTP2. Consider Figure 7-17, which shows a potential usage of M2PA.

  Even though M2UA and M2PA have some similarities, they are different in a number of respects. Although M2UA enables an IP-based node to utilize a standard MTP2 at a remote SG, M2PA is more akin to an IP-based version of MTP2. In other words, the SG-MGC connection shown in Figure 7-17 is effectively an SS7 link. That is not the case for the corresponding connection on Figure 7-16. One result of this difference is that an SG that utilizes M2UA is effectively a remote signaling terminal for the MGC shown in Figure 7-16. An SG that utilizes M2PA is a signaling node in its own right; it has its own signaling point code, and it is effectively an IP-based STP. These characteristics mean that the SG can also process higher-layer signaling functions, such as SCCP. Therefore, an SG that uses M2PA could perform *Global Title Translation* (GTT), for example. An M2UA-based SG cannot perform such functions.

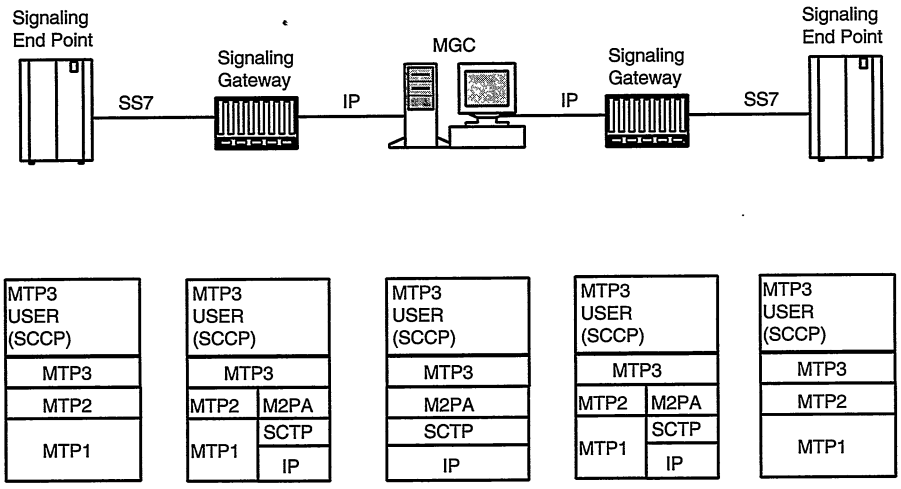


Figure 7-17
M2PA usage in an SG to MGC application

Both M2UA and M2PA can have valid roles to play when transporting SS7 signaling over an IP-based transport. The choice of which protocol to use will depend on how the network architect wants to design the network. Specifically, the choice will depend on which functions are to be performed at which nodes in the network. If, for example, an SG needs to perform functions such as GTT, then M2PA will be the appropriate choice. On the other hand, if the SG is simply meant as an SS7 signaling terminal for an IP-based node (such as an MGC), then M2UA will be sufficient. M2PA has a registered port number of 3565.

- The *SS7 MTP3-User Adaptation Layer* (M3UA) provides an interface between SCTP and those applications that typically use the services of MTP3, such as ISUP and SCCP. M3UA and SCTP enable seamless peer-to-peer communication between MTP3 user applications in the IP network and identical applications in the SS7 network. The application in the IP network does not realize that SCTP over IP transport is used instead of the typical SS7. In the same manner that MTP3 provides services to applications such as ISUP in the SS7 network, M3UA offers equivalent services to applications in the IP network. M3UA has a registered port number of 2905.

  We must recognize that M3UA is simply an adaptation layer between the upper protocols and SCTP. Therefore, although it provides the same primitives to the upper layer as MTP3 offers in standard SS7, M3UA is

not an IP-flavored MTP3. For example, M3UA does not implement standard MTP3 signaling network management messages such as TFA, TFP, and so on. This point is important and will be emphasized shortly.

Consider an MGC that needs to run an application such as ISUP. The MGC can do so in several ways. For example, the MGC can run ISUP over MTP3 over M2UA (or M2PA) over SCTP, as shown in Figure 7-16 and Figure 7-17. Alternatively, the MGC can implement ISUP over M3UA over SCTP, as shown in Figure 7-18. The difference between the two approaches is a matter of where the MTP3 function really resides. In the scenario depicted in Figure 7-18, the real MTP3 exists at the SGs. M3UA simply enables the ISUP application at the MGC to remotely access the MTP3 function at the SG, without the ISUP application realizing that the MTP3 function is not local.

The MGC of Figure 7-18 might have its own point code, separate from that of the SG. In that case, the SG function likes an STP and appears as an STP to the outside SS7 network. The outside SS7 network views the MGC as a typical SS7 signaling endpoint to which access is achieved via one or more SG STPs.

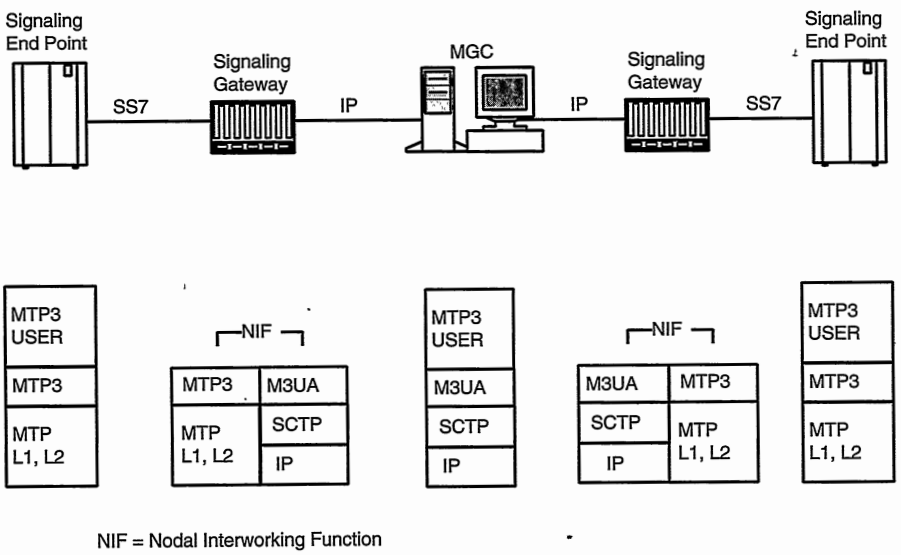


Figure 7-18 M3UA usage in an SG to MGC application

Given that the M3UA provides services to ISUP and given the importance of ISUP interworking, the functioning of M3UA is described in more detail later in this chapter.

- The *SS7 SCCP-User Adaptation Layer* (SUA) provides an interface between SCCP user applications and SCTP. Applications such as TCAP use the services of SUA in the same way that they use the services of SCCP in the SS7 network. In fact, those applications do not know that the underlying transport is different in any way. Similar to other adaptation layers, SUA merely provides a transparent conduit whereby an application such as TCAP at an IP node can use the services of SCCP at an SG. Hence, there can be transparent peer-to-peer communication between applications in the SS7 network and applications in the IP network. SUA has a registered port number of 14001.
- The *ISDN Q.921-User Adaptation Layer* (IUA) also operates on top of SCTP. In ISDN, user signaling, such as Q.931, is carried by the Q.921 Data-link layer. The equivalent Sigtran specification in an IP network is IUA. Thus, Q.931 messages may be passed from the ISDN to the IP network, with identical Q.931 implementations in each network, neither of them recognizing any difference in the underlying transport. The registered port number for IUA is 9000.
- The *V5.2-User Adaptation Layer* (V5UA) has a number of similarities with the IUA protocol and can be considered an extension of IUA. V5.2 defines an *European Telecommunications Standards Institute* (ETSI) interface for a connection between an access network and a local exchange. V5.2 supports network access for analog telephone lines, ISDN basic rate access, ISDN primary rate access, and other types of access that do not use out-of-band signaling. The main purpose of V5.2 is to enable an access concentrator system to be placed between the user device (such as an analog line) and the local exchange, as depicted in Figure 7-19. For example, a concentrator might be placed in a neighborhood, and thousands of subscriber lines might be attached to the concentrator. Up to 16 E1 links would connect the concentrator back to the local exchange. The links between the concentrator and local exchange operate according to the V5.2 protocol, which enables the multiplexing and concentration of user traffic across those links. This situation is depicted in Figure 7-19.

  If the local exchange of Figure 7-19 were to be replaced by a softswitch configuration (MGC and MG), then an SG would be required between the access system and the MGC, as shown in Figure 7-20. The V5UA

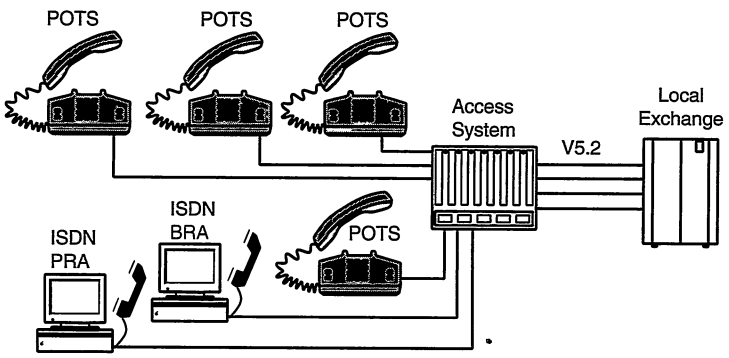


Figure 7-19
Application of V5.2

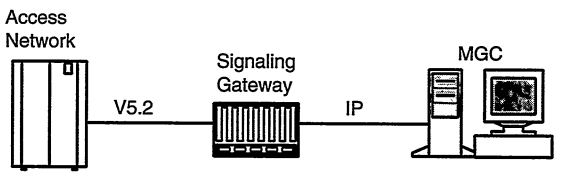
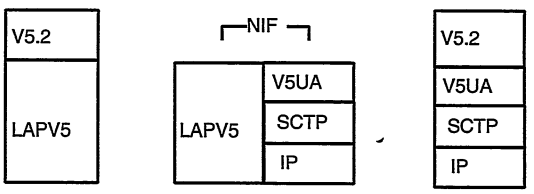


Figure 7-20
V5UA usage between MGC and an access network



NIF = Nodal Interworking Function

protocol enables the V5.2 applications at the MGC to utilize the native V5.2 functions on the access network side of the SG. V5UA has a registered port number of 5675.

SCTP

SCTP is specified in RFC 2960. The primary motivation behind the development of SCTP was that neither UDP nor TCP offered both the speed and reliability required of a transport protocol used to carry signaling. SCTP

offers both speed and reliability, and it offers upper-layer applications the opportunity to take advantage of those characteristics. In the SCTP specification, such an application is known as an *Upper-layer Protocol* (ULP). A ULP can be any of the protocols directly above the SCTP layer, as illustrated in Figure 7-15, such as M2UA or M3UA.

Some SCTP Concepts Before attempting to explain how SCTP provides this reliable transport to the various SCTP users, it is first worth becoming familiar with some SCTP terminology. Note that the following terms comprise just a selection of the most important terms necessary for an understanding of SCTP basics.

SCTP Endpoint An SCTP endpoint is a logical sender or receiver of SCTP packets. In protocol terms, an SCTP endpoint is a combination of one or more IP addresses and a port number. SCTP enables an endpoint to have multiple IP addresses, meaning that the endpoint can be "multihomed" (spread over several physical platforms). This concept is important, because it enables for fault tolerance in the network, something critical to carrier-grade performance.

Even though a given SCTP endpoint can have multiple IP addresses, it can use only one port number. Thus, if an endpoint has several IP addresses, the same SCTP port number is applicable to each. The combination of an IP address and a port number is known as a *transport address*. Note that a given transport address may apply to only one SCTP endpoint, though that endpoint may have several transport addresses.

Association SCTP works by establishing a relationship between SCTP endpoints. Such a relationship is known as an *association* and is defined by the SCTP endpoints involved and the current protocol state. Before applications at two endpoints can communicate, an association must be established. Once communication is complete, the association can be terminated. The association can also be terminated in error situations.

The upper-layer protocols such as ISUP, SCCP, or TCAP are not aware of such associations. After all, they are blind to the fact that the signaling is being carried by something other than standard MTP. Therefore, the task of instigating an SCTP association falls to the applicable adaptation layer.

Packets and Chunks SCTP sits on top of IP. When SCTP wants to send a piece of information to the remote end, it sends what is known as an SCTP packet to IP, which routes the packet to the destination. The SCTP packet comprises a common header and a number of chunks, as depicted in

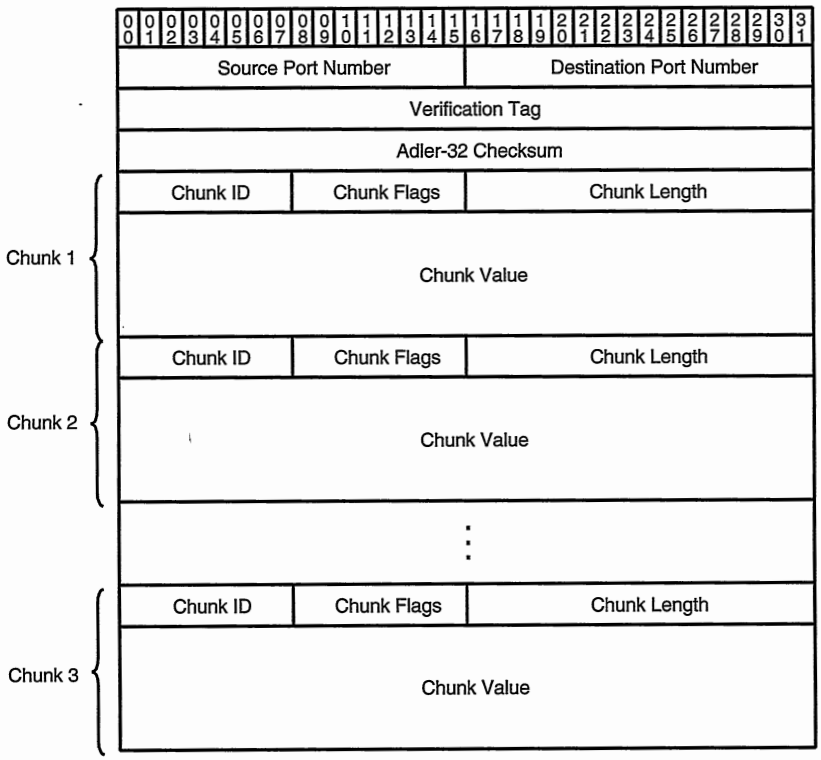


Figure 7-21
The SCTP packet format

Figure 7-21. The common header includes the source and destination port numbers, which, when combined with the source and destination IP addresses, uniquely identify the endpoints. The header also includes a verification tag, which is used to validate the sender of the packet. The verification tag is described in further detail later in this chapter.

The common header also includes an Adler-32 checksum, which is a particular calculation based on the values of the octets in the packet. This checksum is used to ensure that the packet has been received without corruption and provides another level of protection over and above the IP header checksum.

A number of chunks follow the common header. Each chunk is comprised of a chunk header, plus some chunk-specific content. This content can be either SCTP control information or SCTP user information. In the case of SCTP user information from a ULP, the value of the Chunk ID is 0, indicating user payload data. Otherwise, the Chunk ID will have a value indicating a particular type of SCTP control information. The possible values for the chunk flags and chunk length depend upon the value of the Chunk ID.

Streams A stream is a one-way logical channel between SCTP endpoints. One can think of a stream as a sequence of messages from one SCTP user to another. When an association is established between endpoints, part of the establishment of the association involves specifying how many streams are to be supported by the association. If we think of a given association as a one-way highway between endpoints, then the individual streams are analogous to the individual traffic lanes on that highway.

To understand the stream concept in a network scenario, imagine a call agent that uses ISUP and that communicates with the outside SS7 network via an SG. The physical signaling links from the SS7 network terminate at the SG and the actual speech circuits from the PSTN terminate at an MG controlled by the call agent, as shown in Figure 7-22.

If the call agent uses ISUP over MTP3 over M2UA, then the MTP3 function operating at the call agent will attempt to direct messages from ISUP to a particular signaling link since that is one of the functions of MTP3. No physical signaling link is used at the call agent, however; only an SCTP association is made between the M2UA layer at the call agent and the M2UA layer at the SG. In order to handle this situation, the SCTP association should include N streams, where N is the number of signaling links terminated at the SG. In this manner, the M2UA function at the call agent can map a particular signaling link, as specified by MTP3 to a particular stream. Equally, the SG can map the stream to a particular signaling link to the destination.

If the ISUP function at the MGC uses M3UA, then more flexibility can take place in how streams are allocated in the SCTP association between M3UA at the SG and M3UA at the MGC. Since the choice of signaling link is made by the MTP3 function at the SG, it is not necessary that the

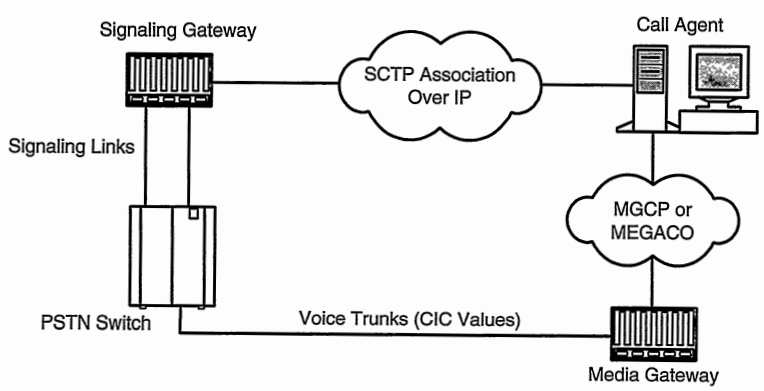


Figure 7-22
Streams allocated according to signaling links or CIC values

streams be allocated according to the signaling link; although they could be. Alternatively, the streams could be allocated according to the DPC/OPC combination or even according to the OPC/DPC/CIC range.

The use of these streams is an important tool to avoid head-of-line blocking and to ensure in-sequence delivery. In-sequence delivery is ensured because each message sent for a given stream contains a stream sequence number. The receiving entity can make sure that messages are transmitted to the user application in the order of stream sequence numbers. Secondly, each stream is processed separately so that message delivery for one stream is not held up while waiting for the next in-sequence message of a different stream.

Types of SCTP Chunks Within an SCTP packet, individual chunks are identified by a chunk ID. Although a chunk ID has 256 possible values and hence 256 different chunk types exist, SCTP chunks can be grouped into four main categories: those that carry SCTP user data, those that carry SCTP control information, those that are reserved by the IETF, and those that carry IETF-defined extensions. The correlation between chunk ID values and chunk types is shown in Table 7-1.

Table 7-1

Chunk ID values and chunk types

ID value	Chunk type
00000000	Payload Data (DATA)
00000001	Initiation (INIT)
00000010	Initiation Acknowledgment (INIT ACK)
00000011	Selective Acknowledgment (SACK)
00000100	Heartbeat Request (HEARTBEAT)
00000101	Heartbeat Acknowledgment (HEARTBEAT ACK)
00000110	Abort (ABORT)
00000111	Shutdown Association (SHUTDOWN)
00001000	Shutdown Acknowledgment (SHUTDOWN ACK)
00001001	Operation Error (ERROR)
00001010	State Cookie (COOKIE ECHO)
00001011	Cookie Acknowledgment (COOKIE ACK)
00001100	Reserved for Explicit Congestion Notification Echo (ECNE)

Motorola - Exhibit 1022 - Page 362

Table 7-1 cont.
Chunk ID values and chunk types

ID value	Chunk type
00001101	Reserved for Congestion Window Reduced (CWR)
00001110	Shutdown Complete (SHUTDOWN COMPLETE)
00001111 to 00111110	Reserved by IETF
00111111	IETF-defined Chunk Extension
01000000 to 01111110	Reserved by IETF
01111111	IETF-defined Chunk Extension
10000000 to 10111110	Reserved by IETF
10111111	IETF-defined Chunk Extension
11000000 to 11111110	Reserved by IETF
11111111	IETF-defined Chunk Extension

Chunk ID values are encoded such that the two highest-order bits specify the action that must be taken if the processing endpoint does not recognize the chunk type. The possible bits and their meanings are as follows:

- **00** Stop processing this SCTP packet and discard it; do not process any further chunks within it.
- **01** Stop processing this SCTP packet and discard it, do not process any further chunks within it, and report the unrecognized parameter in an Unrecognized Parameter Type (in either an ERROR or in an INIT ACK).
- **10** Skip this chunk and continue processing.
- **11** Skip this chunk and continue processing, but report an ERROR chunk using the Unrecognized Chunk Type cause of error.

SCTP Control Chunks The INIT chunk is used to initiate an SCTP association between two endpoints. Unlike many other chunks, the INIT chunk must not share an SCTP packet with any other chunk. In other words, an SCTP packet containing an INIT chunk must contain no other chunks besides the INIT chunk. The INIT chunk is further described later in this chapter.

The INIT ACK chunk is used to acknowledge the initiation of an SCTP association. As is the case for the INIT chunk, the INIT ACK chunk must not share a packet with any other chunk.

The SACK chunk is used to acknowledge the receipt of DATA (payload) chunks and to inform the sender of any gaps in the received chunks. Not every received chunk merits a SACK chunk in response. Instead, the SACK chunk works by indicating where gaps occur. Let us suppose that a receiver has received chunks 1 through 5, plus chunks 8 and 9. The SACK message is used to indicate that chunks 6 and 7 are missing. These are the only chunks that need to be resent. This technique is more efficient that the retransmit mechanism of TCP.

The HEARTBEAT chunk is used to query the reachability of a particular endpoint. Let's assume that no chunks need to be sent from endpoint A to endpoint B during a particular period of time. In that case, endpoint A will send periodic HEARTBEAT messages to endpoint B, just to make sure that endpoint B is still alive. The HEARTBEAT contains sender-specific information. The receiver of the HEARTBEAT chunk should respond with a HEARTBEAT ACK chunk containing heartbeat information copied from the received HEARTBEAT chunk.

The ABORT chunk is sent by an endpoint to end an association abruptly. The ABORT chunk may contain cause information regarding the reason for aborting the association. The ABORT chunk may be multiplexed with other SCTP control chunks into one packet. In such cases, however, the ABORT chunk should be the last chunk in the packet. If it is not the last chunk, then subsequent chunks in the packet are ignored. DATA chunks should not be included in the same packet as an ABORT chunk.

The SHUTDOWN chunk is used for a graceful termination of an association. If a higher-layer application or management application wants to terminate an association, then the endpoint stops sending any new data to the far end. It will wait until all data sent to the far end has been acknowledged and will then send a SHUTDOWN to the far end in order to close the association. The SHUTDOWN will indicate the last data chunk received from the far end. The endpoint may, if necessary, retransmit user data to the far end before sending the SHUTDOWN chunk.

Upon receipt of a SHUTDOWN chunk, an endpoint will make sure that all the user data that it has sent has been acknowledged and, if necessary, retransmit data to the far end. Once it is sure that everything it sent in the past has been received, the endpoint shall send a SHUTDOWN ACK chunk.

Upon receipt of the SHUTDOWN ACK, the sender of the SHUTDOWN shall respond with a SHUTDOWN COMPLETE chunk and will remove all knowledge of the association. When the far end receives the SHUTDOWN COMPLETE chunk, it too will delete all knowledge of the association. At this point, the association is ended.

The ERROR chunk is sent to indicate that the endpoint has detected some error condition. The chunk will include an error cause to provide further information on the type of error. For example, an endpoint may have received a chunk for a nonexistent stream or may have received a chunk that is missing certain mandatory parameters. The receipt of an ERROR chunk does not, in itself, indicate a fatal condition. An ERROR chunk sent on its own may simply enable the receiver to correct the error condition. If the error condition is fatal, then the ERROR chunk can be sent in the same datagram as an ABORT chunk.

The COOKIE ECHO chunk is used only during the initialization of an association. When an endpoint receives an INIT chunk and responds with an INIT ACK chunk, it includes a cookie parameter within the INIT ACK chunk. This parameter contains information specific to the endpoint and to the endpoint's view of the association, a timestamp, and a cookie lifetime value (5 seconds is recommended). When the far end receives the INIT ACK, it copies the cookie information and returns it in a COOKIE ECHO chunk. The COOKIE ECHO chunk can be sent in a packet that also contains DATA chunks. In such a situation, however, the COOKIE ECHO chunk must be the first chunk in the packet.

The COOKIE ACK chunk is sent in response to a COOKIE ECHO chunk. Hence, the COOKIE ACK chunk is used only during the establishment of an association. Since the content of a COOKIE ECHO chunk is the same as what was sent in an INIT ACK, the sender of the INIT ACK can make sure that the initiator of the association has received the cookie information correctly. If the COOKIE ECHO chunk has been received without error within the lifetime specified for the cookie, the receiver of the COOKIE ECHO chunk sends a COOKIE ACK chunk. Otherwise, an ERROR chunk is sent. The COOKIE ACK chunk may be sent in the same datagram as DATA chunks, but it must be the first chunk in the datagram.

Payload Data (DATA) Chunk The DATA chunk is used to carry information to and from the ULP. It has the format shown in Figure 7-23.

SCTP can possibly segment a given user message. This situation could occur if the path MTU is smaller than the size of the message to be sent. The *Beginning* (B) and *End* (E) bits are included because of such potential segmentation. The B bit indicates that this chunk contains the first segment of a user message, and the E bit indicates that the chunk contains the last segment of a user message. If a message is completely contained within one chunk, then both the B and E bits are set to 0. If the message contains more than 2 segments, then the first segment shall have the B bit set to 1

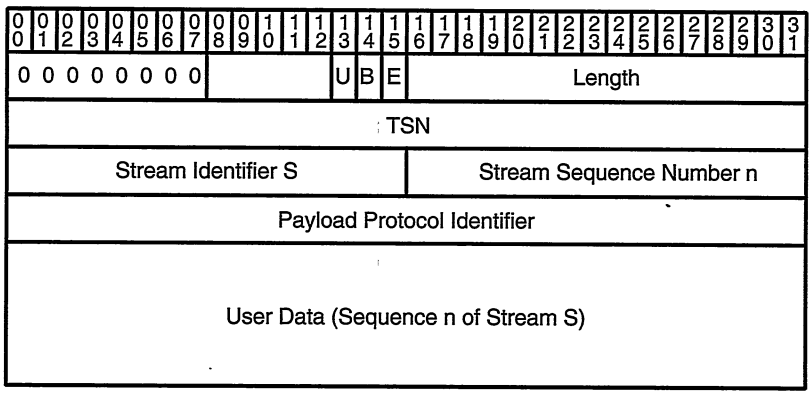


Figure 7-23
Payload data chunk format

and the E bit set to 0, while the last segment shall have the B bit set to 0 and the E bit set to 1. All segments in between shall have both the E and B bits set to 0.

The U bit indicates that this chunk belongs to an unordered data stream. In other words, the order of user messages in the stream is not significant and the value of the stream sequence number should be ignored. In such a case, SCTP passes the data to the appropriate upper layer without any concern for the order of the message. SCTP must still, however, ensure that segmented messages are reassembled before passing the data to the upper layer.

The *transmission sequence number* (TSN) is a 32-bit integer identifying this chunk in the context of the association. It is independent of any stream sequence number and is assigned by SCTP rather than by any user of SCTP. When an endpoint sends an INIT chunk, it includes a TSN value, which corresponds to the first DATA chunk that it plans to send. Thus, the first DATA chunk sent will contain the same value of TSN as in the INIT chunk. Thereafter, the TSN is incremented for each new DATA chunk that the endpoint sends in this association.

The stream identifier (S) is a 16-bit integer, identifying the stream to which the data belongs. The stream sequence number (n) is a 16-bit integer indicating the position of this message within the stream. For a given stream, the value of the stream sequence number begins at 0. It is incremented for each message sent in a given stream. Note that a segmented message shall have the same value of the stream sequence number in each segment.

The payload protocol identifier is passed from the user to SCTP at the sending end and passed from SCTP to the destination user at the receiving

end. It is available to the users for passing further information about the chunk, but it is not examined or acted upon by SCTP.

Establishing an Association The establishment of an association is typically instigated by an SCTP upper-layer protocol, which instructs SCTP to establish the association. The association can be established in advance of any data traffic being sent. For example, if a network administrator provisions an ISUP trunk group on an MGC and the MGC uses M3UA, then the act of defining the trunk group could cause M3UA to request SCTP to establish an association. Similarly, an association could be established when a new SS7 link is brought into service at an SG or when a new CIC range is brought into service.

The process is depicted in Figure 7-24. It begins by sending an SCTP packet containing an INIT chunk. The INIT chunk has the format shown in Figure 7-25.

The Chunk Flags field should be set to 0. The Initiate Tag is a 32-bit integer and is a random number. It must not have the value 0. The end that receives this INIT chunk (the far end) will store the value of the Initiate tag. For every SCTP data packet that the far end sends in this association, it will use the value of the Initiate tag as the Verification tag in the SCTP common header. The *Advertised Receiver Credit Window* (a_rwnd) represents the dedicated buffer space that the sender has allocated for this association. Although the buffer space available to the association may exceed this value during the lifetime of the association, it must never fall below this value. The *Number of Outbound Streams* (OS) and the *Number of Inbound Streams* (MIS) indicate the maximum number of streams that the initiator is willing to send and receive respectively for this association. The

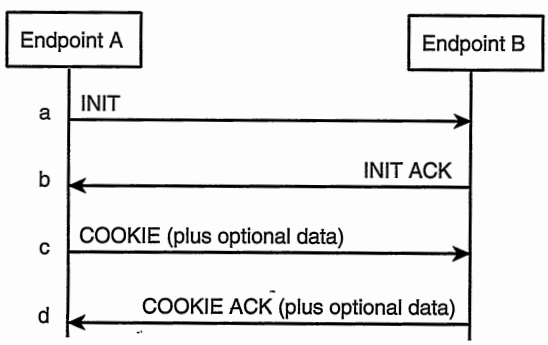


Figure 7-24 SCTP association establishment

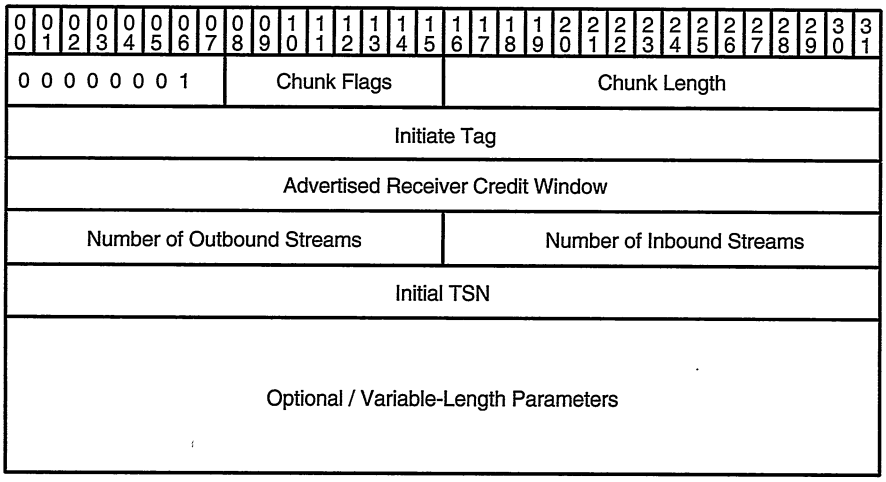


Figure 7-25
The INIT chunk format

value 0 is not allowed for either of these parameters. The Initial TSN is the TSN value that the sender expects to use when sending the first DATA chunk in the association.

The INIT chunk can also contain a number of optional parameters and/or variable-length parameters. The INIT chunk can, for example, contain one or more IPv4 or IPv6 addresses, or it can contain a host name that can be resolved to one or more IP addresses.

When the other endpoint receives the INIT chunk, it responds with an INIT ACK chunk, which has a format identical to the INIT chunk, with the exception that the optional/variable component must include a cookie. Importantly, the Verification tag in the SCTP common header must contain the same value as the received Initiate tag in the INIT chunk.

The INIT ACK chunk can also contain a number of IPv4 or IPv6 addresses and possibly a host name that can be resolved to one or more IP addresses. Given that the INIT ACK chunk also contains numbers of inbound and outbound streams, both ends of the association know the maximum number of streams that the other can send and receive. Each endpoint must respect the other's requirements and should not send more than what the other endpoint can handle.

Upon receipt of the INIT ACK chunk, the initiator of the association sends a COOKIE ECHO chunk, the content of which is copied from the cookie parameter of the received INIT ACK. In order to quickly start the message exchange for which the association is being created, one or more

additional DATA chunks may be included in the packet. The COOKIE ECHO chunk must be the first chunk in the packet, however.

Finally, the receiver of the COOKIE chunk sends a COOKIE ACK chunk. This is just a header containing the CHUNK ID value of 00001011, the chunk flags all set to 0, and a length indicator of 4. In other words, it's just a simple acknowledgement with no additional information. The packet containing the COOKIE ACK chunk can also contain a number of DATA chunks, provided that the COOKIE ACK chunk is the first chunk in the packet.

Transferring Data The reliable transfer of user data is achieved by the use of two SCTP chunks. The first is the DATA chunk described earlier and depicted in Figure 7-23. The second is the SACK chunk, which is an SCTP control chunk and is shown in Figure 7-26.

The easiest explanation of the SACK chunk is by example. Let's assume that an endpoint has transmitted data chunks 1 through 11. Let's also assume that chunks with TSNs 1 through 4 and those with TSNs 7, 8, 10, and 11 have been received. Hence, chunks 5, 6, and 9 are missing. Let's also assume that chunks with TSN 8 and TSN 11 have been received twice. Therefore, the received data appears as depicted in Figure 7-27.

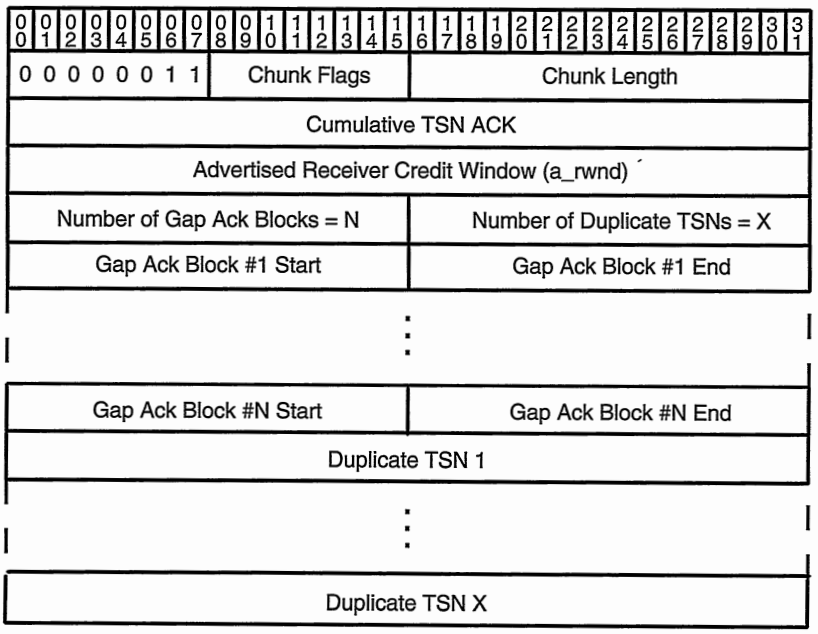


Figure 7-26
The SACK chunk format

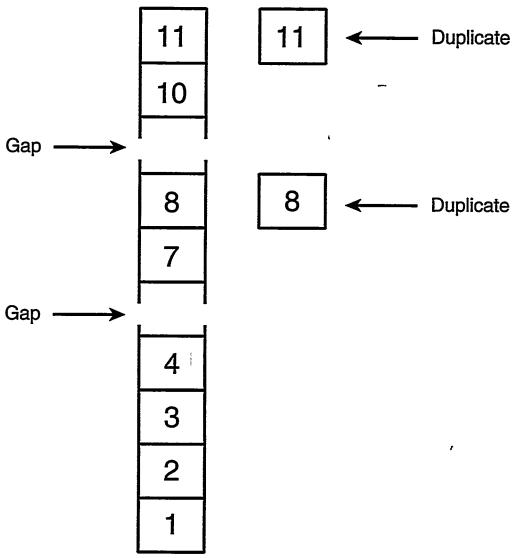


Figure 7-27
Data missing and/or duplicated

In this scenario, the Cumulative TSN ACK field would contain the value 4, as this field indicates the highest TSN value received without any gaps. The number of Gap Ack Blocks (N) indicates the number of fragments received after the unbroken sequence. In our example, N has the value 2 to indicate the block 7 to 8 and the block 10 to 11. The number of duplicate TSNs (X) indicates the number of TSNs that have been received more than once.

The Gap Ack Block number 1 start field indicates the offset of the first segment from the unbroken sequence. This is the difference between the TSN value in the Cumulative TSN ACK field and the lowest TSN value of the first segment. In our example, this is the difference between 4 and the value 7 (i.e., 3). The Gap Ack Block number 1 end field indicates an offset from the Cumulative TSN ACK and the highest TSN in the first block after the Cumulative TSN. This field has the value 4 in our example (the difference between 8 and 4). This process is repeated for each fragment. Thus, for the next fragment, the Gap Ack Block start and end values are 6 and 7 respectively. Finally, we have a listing of the duplicate TSNs. In this case, the values 8 and 11 would be listed.

The *Advertised Receiver Window Credit* (a_rwnd) indicates the updated buffer space of the sender of this Selective ACK. This indicates how much receive buffer space is free at the time the SACK is sent. It enables the remote end to manage the amount of data sent, such that buffer overflow and consequent data loss can be avoided. SCTP specifies procedures based on the a_rwnd and the SACK chunk in general to ensure that congestion is avoided.

SCTP Robustness Robustness is a key characteristic of any carrier-grade network. Robustness means that the network should implement procedures whereby failures or undesired occurrences are minimized. Robustness also means the capability to handle a certain amount of failure in the network without a significant reduction in quality. Furthermore, the network should provide a graceful rather than a drastic degradation in the event of failures or overload. SCTP addresses these issues in a number of ways.

As mentioned previously, SCTP implements congestion control mechanisms to ensure that one endpoint does not flood another with messages. Furthermore, SCTP incorporates Path MTU discovery so that messages are not sent if they are too long to be handled by the intervening transport network.

Recall that the SCTP common header contains source and destination port numbers. Recall also that the INIT and INIT ACK chunks may optionally include one or more IP addresses or a host name that can be resolved to one or more addresses. The inclusion of these parameters enables a given endpoint to be effectively multihomed (to have multiple IP addresses). If the INIT or INIT ACK chunk does not contain any IP addresses, then the IP address shall default to the IP address from which the packet is sent. Greater robustness is offered by having multiple IP addresses, however.

If a given endpoint supplies several IP addresses, then the other end shall choose one of those addresses as the primary destination address. If the primary address fails, then one of the other addresses is chosen as the destination address for subsequent messages. Furthermore, if one or more DATA chunks are transmitted to a given address and not acknowledged within the retransmission timer, then the sender should send the retransmission to one of the other addresses.

SCTP ensures that an endpoint is aware of the reachability of another endpoint through the use of SACK chunks if DATA chunks have been sent and through HEARTBEAT chunks if an association is idle. Therefore, the detection of path or endpoint failure is assured and appropriate action can be taken.

Motorola - Exhibit 1022 - Page 371

M3UA Operation

The following describes how M3UA over SCTP can be used in a VoIP network to provide seamless interworking with the external SS7 network for ISUP. First, let's become familiar with a number of terms applicable to M3UA. In fact, many of these terms apply to all of the adaptation layers:

- An *application server* (AS) is a logical entity handling signaling for a particular scope. For example, an AS could be a logical entity within a softswitch that handles ISUP signaling for a particular SS7 DPC/OPC/CIC range. Equally, an AS could be a logical entity within a database application that handles signaling for a particular SCCP DPC/OPC/*Subsystem Number* (SSN) combination. An AS contains a set of *application server processes* (ASPs), which are those actual processes operating within the AS. In fact, the AS can be considered as a list of ASPs, some of which are active and some standby.
- An *application server process* (ASP) is a process instance of an AS, acting as an active or standby process. An ASP could be, for example, that process within an MGC that is currently handling ISUP signaling. The ASP has an SCTP endpoint. Therefore, the ASP can be multihomed (spread across multiple IP addresses). In a robust network, there should be at least one active ASP and at least one standby ASP for a given application. A single ASP can serve multiple ASs. For example, an MGC could have several ASs, each corresponding to a unique OPC/DPC combination. That same MGC might have only one ASP handling all ISUP signaling (or two ASPs for redundancy or load sharing).
- A *routing key* is a set of SS7 parameters such as an SLS, DPC, OPC, or CIC range that identifies that signaling for a given AS. For example, if a given AS is to handle ISUP signaling for a particular combination of OPC/DPC/CIC range, then that OPC/DPC/CIC range is the routing key. Within an SG, a particular routing key will point to a particular AS. In other words, a one-to-one relationship exists between a routing key and an AS.
- *Network appearance* is a mechanism for separating signaling traffic between an SG and an ASP, where all the traffic uses the same underlying SCTP association. Imagine, for example, an SG that is an international SG. This SG will have at least one national point code and one international point code. The SG uses a national variant of MTP for communication within the national network and it uses the

Motorola - Exhibit 1022 - Page 372

ITU-T variant of MTP for communication within the international network. For communication between an SG and a call agent/MGC, the messages being exchanged must be placed in the context of the correct network appearance so that the appropriate link set can be used on the non-IP side of the SG.

Signaling Network Architecture In order to ensure carrier-grade service, we must ensure that no single point of failure exists on the network between the SG and ASP. Consequently, SGs should be set up at least in pairs in a manner similar to the arrangement of STPs in the SS7 network. Furthermore, ASPs should be set up in a redundant or load-sharing configuration, spread out over different physical platforms (hosts). Such a robust configuration is shown in Figure 7-28. Note that the requirements for such a robust configuration are not something unique to the usage of M3UA; they apply to any system that needs to be fault tolerant (a requirement for carrier-grade operation).

Each ASP needs to be associated with a point code. The allocation of point codes to ASPs is completely flexible, however. For example, all ASPs connected to a given SG could share the same point code as the SG. In such a case, the combination of SG and ASPs appears to the SS7 network as a single signaling endpoint. Alternatively, all ASPs connected to a given SG could share the same point code, but a different point code from the SG. In such a case, the SG would appear to the SS7 network as an STP and the

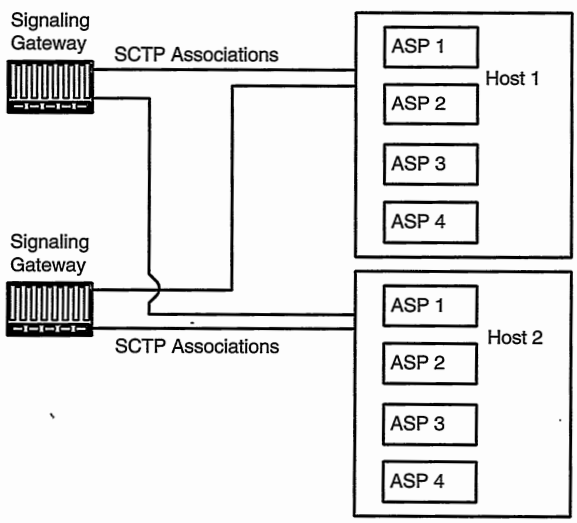


Figure 7-28 Robust signaling architecture

combined ASPs as a single signaling endpoint located behind that STP. Another alternative would be for every ASP to have its own point code or for groups of ASPs to share a point code separate from the point code of the SG. In that case, the SG appears as an STP and each ASP or group of ASPs appears as a unique signaling endpoint.

The options available are dependent only upon the translation and mapping capabilities of the SG. If a given ASP or group of ASPs can communicate with the SS7 network via more than one SG, then the ASP or group of ASPs must have a point code different from those of the two SGs. In such a scenario, the SGs operate as STPs, as shown in Figure 7-29.

M3UA uses a client server model, where one end of the SCTP association acts as the client and initiates the association, while the other end acts as the server. Typically, the ASP will act as the client, with the M3UA application at an SG acting as a server. Product implementations should, however, enable a node to act either as a client or a server.

Services Provided by M3UA As mentioned, M3UA provides an interface between an application such as ISUP and the underlying transport. It enables the application to utilize the services of the real MTP3, which is located at an SG. M3UA accomplishes this task in a transparent manner, so that the application does not realize that the instance of MTP3 being used is at the SG rather than local.

In order to provide services to the upper layer, M3UA must offer the same primitives to the upper layer as are offered by MTP3. These are as follows:

- The *MTP-Transfer request* is sent from the upper layer to M3UA to request that a message be transferred to a particular destination.
- The *MTP-Transfer indication* is used by M3UA to pass an incoming message to the upper layer.

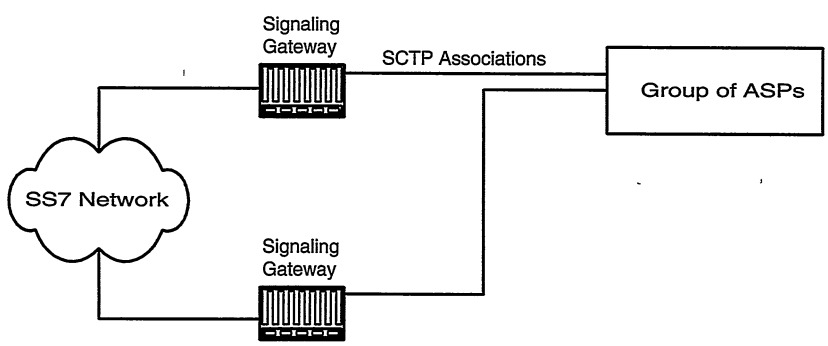


Figure 7-29
SGs acting as STPs

- The *MTP-Pause indication* is sent by M3UA to the upper layer to indicate that signaling to a particular destination should be suspended. This primitive is used, for example, when the destination is not reachable.
- The *MTP-Resume indication* is sent by M3UA to the upper layer to indicate that signaling to a particular destination can resume.
- The *MTP-Status indication* is sent by M3UA to the upper layer to inform the upper layer of some change in the SS7 network such as network congestion or a destination user part becoming unavailable.

Transferring application messages from the application to the SS7 network is a relatively straightforward process. Let's take an example of an ISUP application at an MGC wanting to send a message to the SS7 network. The application issues an MTP-Transfer request to M3UA, and M3UA effectively sends this to SCTP as a DATA chunk to be transmitted on a particular SCTP association and in a particular stream. SCTP ensures that it reaches the SG, where the content of the DATA chunk is passed up to M3UA, which passes it to the interworking function of the SG. The interworking function passes it to MTP3 on the SS7 side of the SG, which takes care of routing the message correctly to the SS7 network.

In the opposite direction, the process is similar. A message arrives at the SG and is passed from MTP3 up to the interworking function and from the interworking function to M3UA. Based on the SS7 parameters of the message, such as the DPC/OPC/CIC range, the appropriate AS and the appropriate ASP are chosen. If more than one ASP is active, then one of the active ASPs is chosen (depending on the load-sharing algorithm). M3UA packages the message for transmission by SCTP as a DATA chunk on the appropriate SCTP association and within the appropriate stream. At the destination, the DATA chunk is passed to M3UA, which passes the information to the application using the MTP-Transfer indication primitive.

M3UA Messages M3UA includes a number of messages between peer M3UA entities. These have the generic format shown in Figure 7-30. This format contains a header followed by the message content. The header is common across all adaptation layers. For M3UA, the protocol version is 0000 0001. The message class is as follows:

- **0** Management messages
- **1** Transfer messages
- **2** *SS7 Signaling Network Management* (SSNM) messages

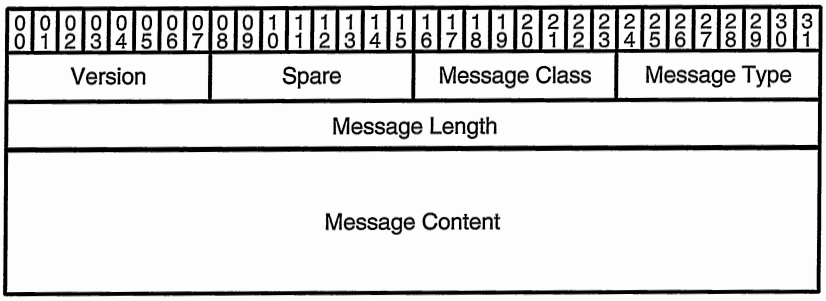


Figure 7-30
Common message format

- **3** *ASP State Maintenance* (ASPSM) messages
- **4** *ASP Traffic Maintenance* (ASPTM) messages
- **5** Reserved for IUA-related Q.921/Q.931 boundary primitive transport messages
- **6** Reserved for M2UA-related user adaptation messages
- **7** Reserved for SUA-related connectionless messages
- **8** Reserved for SUA-related connection-oriented messages
- **9** *Routing Key Management* (RKM) messages
- **10** Reserved for M2UA interface identifier management messages
- **11** Reserved for M2PA messages
- **12 to 127** Reserved by the IETF
- **128 to 255** Reserved for IETF-defined message class extensions

When M3UA passes user information between the SG and MGC, as described previously, it does so by sending M3UA DATA messages. These DATA messages are packaged as SCTP DATA chunks. When sending a DATA message, the message class field has the value 1 and the message type has the value 1 (see Table 7-2).

In addition to packaging user messages in the form of M3UA DATA messages, M3UA includes other messages between M3UA peer entities so that the entities may be able to communicate information regarding the SS7 network in general. For example, if a remote destination becomes unavailable, then the SG will become aware of this fact through SS7 signaling network management messages. The ISUP application at an MGC must also be made aware of the event so that it does not try to send messages that cannot reach their destination. M3UA at the MGC can indicate such an event through the use of the MTP-Pause indication primitive. But M3UA is not MTP3, and SS7 signaling network management messages are not

passed directly from the SG to the MGC. So how does the M3UA at the MGC know that the destination is unavailable? The answer is that the information is passed in other M3UA messages besides the DATA message. Table 7-2 lists the various types of M3UA messages.

Table 7-2
M3UA messages

Message name	Message class	Message class value	Message type value
Error (ERR)	MGMT	00	00
Notify (NTFY)	MGMT	00	01
Data	Transfer	01	01
Destination Unavailable (DUNA)	SSNM	02	01
Destination Available (DAVA)	SSNM	02	02
Destination State Audit (DAUD)	SSNM	02	03
SS7 Network Congestion State (SCON)	SSNM	02	04
Destination User Part Unavailable (DUPU)	SSNM	02	05
Destination Restricted (DRST)	SSNM	02	06
ASP Up (ASPUP)	ASPSM	03	01
ASP Down (ASPDN)	ASPSM	03	02
Heartbeat (BEAT)	ASPSM	03	03
ASP Up Acknowledgment (ASPUP ACK)	ASPSM	03	04
ASP Down Acknowledgment (ASPDN ACK)	ASPSM	03	05
Heartbeat Acknowledgment (BEAT ACK)	ASPSM	03	06

Motorola - Exhibit 1022 - Page 377

Table 7-2 cont.

M3UA messages

Message name	Message class	Message class value	Message type value
ASP Active (ASPAC)	ASPTM	04	01
ASP Inactive (ASPIA)	ASPTM	04	02
ASP Active Acknowledgment (ASPAC ACK)	ASPTM	04	01
ASP Inactive Acknowledgment (ASPIA ACK)	ASPTM	04	02
Registration Request (REG REQ)	RKM	09	01
Registration Response (REG RSP)	RKM	09	02
Deregistration Request (DEREG REQ)	RKM	09	03
Deregistration Response (DEREG RSP)	RKM	09	04

Management Messages The *Error* (ERR) message is used when a message is received unexpectedly or with invalid contents. The ERR message includes an error code and optional additional information regarding the error condition.

The *Notify* (NTFY) message is used between M3UA peers to communicate the occurrence of certain M3UA-related events. For example, the NTFY message would be used to communicate a status change in an ASP.

Signaling Network Management Messages M3UA includes the following messages for signaling network management:

- **Destination Unavailable (DUNA)** This message is sent from the SG to all concerned ASPs to indicate that a destination within the SS7 network is not available. The message indicates the affected destination (DPC) and, optionally, the network appearance and routing context (which identifies the routing key). The DUNA might include several destinations, so that one message can be used to indicate the

unavailability of several signaling points. The message allocates 24 bits for each unavailable point code and also includes an 8-bit mask to indicate the unavailability of multiple contiguous point code values. The mask is effectively a wild card indicator and specifies how many digits in the point code value are wild-carded. For example, a network mask of 8 when used with an ANSI point code would indicate that the last 8 bits of the point code are wild-carded. In other words, all point codes in the cluster are unavailable.

The network appearance is used when the SG is logically partitioned across multiple networks and it serves to logically separate the signaling between SG and ASP according to each of the logical portions of the SG. The network appearance is also used to indicate the format of the DPC (14-bit or 24-bit). Therefore, the same DUNA message can be used in ANSI, ITU-T, or other national-specific signaling environments.

DUNA is generated at the SG when it determines from MTP3 network management messages that a destination is unavailable. The DUNA message is transmitted to the ASP, where M3UA uses it to create the primitive MTP-Pause indication, which it issues to the upper-layer protocol (such as ISUP).

- **Destination Available (DAVA)** This message is sent from the SG to all concerned ASPs when a destination becomes reachable. At the ASP, it is mapped to the primitive MTP-Resume indication. It contains the same information as the DUNA with respect to the destinations in question and the network appearance.
- **Destination State Audit (DAUD)** This message is sent from an ASP to an SG when the ASP wants to query the state of SS7 routes to one or more destinations. With the exception of the message-type value, the DAUD has the same format as the DUNA message.
- **SS7 Network Congestion (SCON)** This message is sent from the SG to all concerned ASPs to indicate that the route to one or more SS7 destinations is congested. At an ASP, it is mapped to the primitive MTP-Status indication, which is issued to the upper-layer protocol. The SCON message can also be sent from an ASP to the SG to indicate congestion at the ASP end of the interface. In that case, the SCON message includes a point code for the congested ASP, plus an optional congestion-level indicator.
- **Destination User Part Unavailable (DUPU)** This message is sent from the SG to all concerned ASPs to indicate that a given user part at

a given destination is not available. It indicates the DPC in question and the user part in question, such as ISUP, SCCP, or TUP. At the ASP, the message is mapped to the primitive MTP-Status indication. The MTP-Status indication may be used to indicate congestion in the network or the unavailability of a given destination user part. Although in MTP3 these different events are indicated with the same primitive by using different cause codes, M3UA communicates these events between M3UA peers through the use of two different messages.

- **Destination Restricted (DRST)** This message can be sent from an SG to concerned ASPs to indicate that one or more SS7 destinations are restricted from the perspective of the SG. Since this message has significance only from the perspective of the SG, it might be possible for signaling traffic to be routed to a destination via a different SG. The M3UA layer at the ASP should take such an alternative path if it is available.

ASP State Management Messages M3UA includes the following messages for ASP state management:

- **ASP Up (ASPUP) and ASP Up Acknowledgment (ASPUP ACK)** The ASPUP message is sent from an M3UA layer to its peer to indicate that the M3UA layer is ready to receive ASP management and ASP traffic-related messages for all routing keys that the ASP is configured to handle. Prior to the establishment of an SCTP association between an *SG Process* (SGP) and an ASP, the ASP is considered to be in state ASP-DOWN. Once an SCTP association is established, then the affected ASP will send an ASPUP message. This message will cause the SG to consider the ASP to be in state ASP-INACTIVE, which means that the ASP is "awake" but not yet ready to handle traffic. The SG responds to the ASPUP, with the ASPUP ACK message. If the ASP does not receive that response within a timeout period, then the ASP can resend the ASP UP message.
- **ASP Down (ASPDN) and ASP Down Acknowledgment (ASPDN ACK)** An ASP will send an ASPDN message to an SG when it wants to be removed from service in all ASs. Sending an ASPDN message will convey that no further DATA, SSNM, or ASPTM messages should be sent to the ASP. The ASPDN message can be generated automatically or as a result of some maintenance action. When an SG receives the ASPDN message, it considers the ASP to be in state ASP-DOWN and responds with an ASPDN ACK message.

- **Heartbeat messages (BEAT and BEAT ACK)** The BEAT message is an optional message used by an M3UA entity to ensure that its peer entity is available. The BEAT message contains a Heartbeat Data parameter that is set by the sending node. The parameter could include, for example, a sequence number or timestamp. A node that receives a BEAT message will respond with a BEAT ACK message, which contains the same parameter values as the BEAT message. In other words, the BEAT ACK message is effectively an echo of the BEAT message.

  The heartbeat mechanisms of M3UA are optional. SCTP has its own heartbeat procedure and M3UA does not need to use a heartbeat technique of its own when M3UA operates over SCTP. M3UA can operate over other types of transports, in which case the BEAT and BEAT ACK messages might be required.

ASP Traffic Management Messages M3UA includes the following messages for ASP traffic management:

- **ASP Active (ASPAC) and ASP Active Acknowledgment (ASPACN ACK)** An ASP will send an ASPAC message when it is ready to begin processing traffic. When an SCTP association is established, the ASP will typically send the ASPAC immediately after receiving an ASPUP ACK from an SG. The ASPAC optionally contains a routing context, which identifies the routing keys for which the ASP is to be considered active. The inclusion of the routing context can be necessary if the ASP can serve multiple ASs. Recall that a one-to-one relationship exists between an AS and a routing key and the routing key can correspond to combinations of parameters such as OPC/DPC/CIC range. Recall also that an ASP is a process (and could correspond to a physical processor), and that an ASP could be shared among several ASs. For example, an ASP could be configured to support traffic processing for several combinations of DPC/OPC/CIC range. If the ASP is to actively process traffic for a limited set of those combinations, then the routing context will indicate the applicable combinations. If the routing context is omitted, then the ASP is prepared to handle traffic for all combinations for which it is configured. Upon receipt of an ASPAC message, the SG will respond with ASPAC ACK.
- **ASP Inactive (ASPIA) and ASP Inactive Acknowledgment (ASPIA ACK)** An ASP will send an ASPIA message when it no

longer wants to process traffic for some or all the routing keys applicable to it. This action can be initiated automatically or as a result of some maintenance action. As is the case for the ASPAC message, the ASPIA can include a routing context parameter to indicate those routing keys for which the ASP is to be considered inactive. If the routing context parameter is omitted, then the ASP is to be considered inactive for all applicable routing keys.

Upon receipt of the ASPIA message, the SG marks the ASP as inactive and stops sending any traffic towards it. If the ASP was part of a group of ASPs in load-sharing mode, then the traffic will be distributed to the other ASPs in the group. The SG responds to the ASPIA with ASPIA ACK.

Figure 7-31 and Figure 7-32 show examples of ASP state management and ASP traffic management messages. In Figure 7-32, we also see the use of the NOTIFY message to inform an ASP of the change in status of another ASP in the AS.

Routing Key Management Messages M3UA includes the following messages for routing key management:

- **Registration Request (REG REQ) and Registration Response (REG RSP)** In the previous discussions, we have addressed the fact that a given ASP can be associated with one or more routing keys (such as a DPC/OPC/CIC range). This association must be recognized at both the ASP and the SG, and it will occur as a result of some management or provisioning action. One way to establish the association is to provision the necessary data at both the ASP and the SG. If such provisioning is done manually, then one runs the risk that human error can intervene and lead to a data mismatch between the

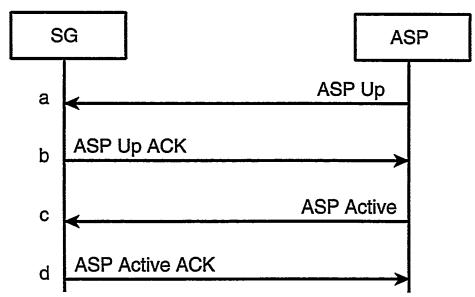


Figure 7-31
The establishment of traffic between an SG and an ASP

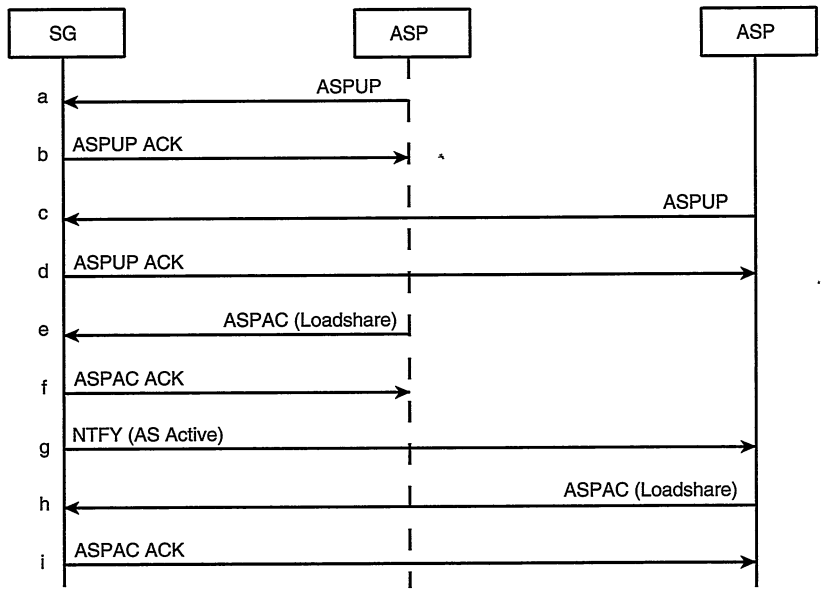


Figure 7-32
The establishment of traffic between an SG and two ASPs (load-sharing)

ASP and the SG. An alternative approach is to provision the necessary data only at an ASP and for the ASP to then communicate the information to the SG, which will ensure information consistency between ASP and SG. Routing key management messages are the means by which such consistency is maintained.

An ASP uses the REG REQ message to register with an SG. The message indicates to the SG that the ASP wants to participate in the traffic handling for the AS identified by a specific routing key. In fact, since a given ASP can serve multiple ASs (corresponding to multiple routing keys), the REG REQ message can include multiple routing keys, each of which specifies a combination such as DPC; DPC and OPC; the DPC, OPC, and CIC range; DPC and the network appearance; and so on. In the case of a successful registration, the SG will respond with a successful REG RSP message that will include a number of registration result parameters corresponding to the number of routing keys included in the REG REQ message and a routing context value associated with a successfully registered routing key. The inclusion of the routing context value means that further communication regarding a given routing key can be referenced by the routing context value rather than having to send the complete description of the routing key (such as DPC/OPC/network appearance/CIC range) every time.

A registration can be unsuccessful in several cases. For example, the SG might not consider the routing key to be valid. Alternatively, the SG might not support the dynamic registration of routing keys. Dynamic registration means that an SG will self-provision a new routing key if it receives a valid, new, and unique routing key in an REG REQ message. In other words, dynamic registration enables routing key data to be provisioned only at the ASP and to be propagated to one or more SGs.

Regardless of the result of registration at the SG, the SG will respond to an REG REQ message with at least one REG RSP message, which indicates the success or failure of the registration. The REG RSP can include multiple registration result parameters. Thus, if a single REG REQ message specifies multiple routing keys, a single REG RSP message can indicate the success or failure of registration for each of the routing keys. In the case of failure, the registration result indicates the reason for failure.

- **Deregistration Request (DEREG REQ) and Deregistration Response (DEREG RSP)** The DEREG REQ and DEREG RSP messages are simply the opposite of the corresponding registration messages. An ASP sends a DEREG REQ to an SG when it wants to deregister a given routing key. The message contains a list of routing contexts that the ASP wants to deregister. The SG will respond with a DEREG RSP indicating the success or failure of each routing context.

  The deregistration of a routing key by an ASP does not necessarily mean that the routing key will be deleted at the SG. Multiple ASPs may support a given routing key in a load-sharing or active/standby arrangement. If, however, a deregistration occurs for the last ASP handling a given routing key, then the SG can choose to also delete the routing key itself.

M2UA Operation

The previous discussion focuses on a situation where the call agent/MGC implements a function such as ISUP, but it does not implement any of the lower layers of the SS7 stack. In particular, MTP3 is not implemented at the MGC. The direct consequence of this approach is that the call agent does not have the same view of the SS7 network as an entity that implements MTP3. In particular, it does not send or receive signaling network management messages.

If it were necessary (or desirable) for the call agent to be more involved in signaling network management, then one option is to implement MTP3 at the MGC and use M2UA over SCTP to access the MTP2 functions at an SG. Although this gives the MGC more visibility of the SS7 network, another consequence is that the MGC can be considered more tightly coupled to the SG. In fact, the SG becomes a remote signaling terminal that is logically part of the MGC from an SS7 perspective.

The MTP3 function at the MGC is a normal MTP3 implementation that includes routing and distribution capabilities and that needs to know about the details of all signaling links that it might use. The main purpose of the SG is to take care of the MTP2 functions, which means that the SG acts as a signaling terminal for the MTP3 application at the MGC.

M2UA uses concepts similar to those used by M3UA. These include the concept of an AS and an ASP. It also includes similar messages, such as ASPUP, ASPDN, ASPAC, ASPIA, NTFY, BEAT, ERR, and the applicable acknowledgements. These messages are used in exactly the same way as used in M3UA, with the difference that the ASPs in M2UA perform different functions to those in M3UA. In M3UA, the ASP can be related to something like a DPC/OPC/CIC range or some other set of characteristics; in M2UA, the ASP is an instance of MTP3 at a node such as an MGC.

The messages sent between M2UA peers use the same format of message used by M3UA. The difference is the Message Type field of the common header. Those messages that are M2UA-specific have different message class values (values 6 and 10) to those messages that are used for other adaptation layers. The management, ASP state management, and ASP traffic management messages are common across all adaptation layers. The following section describes the M2UA-specific messages.

MTP2 User Adaptation (MAUP) Messages (Message Class 6) Unlike M3UA where the M3UA message immediately follows the common header, M2UA utilizes an M2UA header that applies to all *MTP2 User Adaptation* (MAUP) messages. In other words, a given MAUP message begins with the common header, followed by the M2UA header, followed by the message itself. The M2UA header contains an interface identifier. This identifier identifies the physical interface at the SG on which a given signaling message is sent or received. In practical terms, the interface identifier indicates a specific signaling link. The following are the MAUP messages:

- *DATA* (message type value 01) is used to carry an MTP2-user *protocol data unit* (PDU). The DATA message can optionally contain a Correlation ID. When an M2UA peer receives a DATA message, it can

respond with a DATA ACKNOWLEDGE message (message type value 15) to indicate successful receipt of the message. That message contains the same value of Correlation ID. This process can be useful in the event of SCTP peer congestion or SCTP association failure.

- *ESTABLISH REQUEST* (message type value 02) is used to establish a link at the SG or to indicate that a link has been established. In setting up signaling links, the establishment of those links at the SG may take place in advance of management action on the MGC to establish links. In such a case, when the MGC sends an establish request to the SG, the SG may simply return the ESTABLISH CONFIRM (message type value 03).
- *RELEASE REQUEST* (message type value 04) is used by the MGC to request that the SG take a particular signaling link out of service. The request will include the reason why the link should be released. Once the link has been taken out of service, the SG responds with a RELEASE CONFIRM (message type value 05). It is also possible for the SG to autonomously take a link out of service, which would occur, for example, in the case of a hardware failure at the SG. In such a case, the SG would send a RELEASE INDICATION (message type value 06) to the MGC.
- *STATE REQUEST* (message type value 07) is sent from an MGC to the SG to cause the SG to perform some action on a particular signaling link. The actions include tasks such as requesting normal link alignment or flushing transmit and retransmit buffers. Upon completion of the action at the SG, the SG responds with a STATE CONFIRM (message type value 08). It is also possible for the SG to autonomously send a STATE INDICATION (message type value 09) to the MGC to indicate the condition on a link. The message might be sent upon the occurrence of a remote processor outage or a change in the physical state of the link itself.
- *CONGESTION INDICATION* (message type value 14) can be sent from an SG to an ASP (at an MGC, for example) to indicate the congestion status and discard status of an SS7 link. When the MSU buffer fill increases above an onset threshold, decreases below an abatement threshold, or crosses a discard threshold in either direction, the SG will send a CONGESTION INDICATION message when it supports SS7 MTP2 variants that support multiple congestion levels.

  The SG will send the message only when a change in either the discard level or the congestion level can actually be reported, meaning that the

value is different from that of the previously sent CONGESTION INDICATION message. In addition, the SG will limit the frequency of CONGESTION INDICATION messages.

A number of messages are used during link changeover. In such an event, it is necessary for the ASP to retrieve certain information from the SG, particularly information that is stored at the SG for retransmission on a link. DATA RETRIEVAL REQUEST (message type value 10) is used during link changeover procedures. This message is sent by the ASP to request the BSN or to retrieve PDUs from the transmit and retransmit queues. The SG responds with the DATA RETRIEVAL CONFIRM (message type value 11). It is also possible for the SG to just send a PDU to the ASP from the transmit or retransmit queue, in which case the SG will use a DATA RETRIEVAL INDICATION (message type value 12). Multiple RETRIEVAL INDICATION messages may be sent from the SG to the ASP in the case of link changeover. When the final PDU from the retransmit queue is being sent, the SG will use the DATA RETRIEVAL COMPLETE INDICATION (Message type value 13) instead. This message should be sent only once in a given changeover event.

Interface Identifier Management (IIM) Messages (Message Class 10) Much like M3UA, M2UA uses ASP state management messages and ASP traffic management messages. The difference is that an ASP in M2UA will handle the processing of traffic for one or more signaling links, while an ASP in M3UA will have traffic for one or more routing keys (such as OPC/DPC/CIC range). In other words, when an M2UA ASP becomes active, then M2UA at an SG will send the ASP traffic related to a specific signaling link. In order to do so, the SG must know which links a given ASP is to handle. The SG can know this information through data provisioning at the SG or through registration of an ASP with an SG. In other words, an ASP can register with an SG to inform the SG of which links are to be associated with a given ASP.

This process is similar to the routing key management process of M3UA. In fact, the same message names are used: REG REQ, REG RSP, DEREG REQ, and DEREG RSP. The REG REQ message is sent from an ASP to an SG to associate one or more link keys with the ASP. The link key includes a *Signaling Data Terminal Identifier* (SDTI) and a *Signaling Data Link Identifier* (SDLI). The REG REQ message also includes a *Local-LK-Identifier*. This parameter is used to correlate a REG REQ message and its response. An SG will respond to an REG REQ with an REG RSP, which contains the

same value of Local-LK-Identifier and a registration result to indicate the success or failure of the registration.

An ASP sends the DEREG message to deregister a particular interface, that is, to indicate that the ASP no longer wants to handle traffic for that interface. The reply to the DEREG REQ is the DEREG RSP.

Although the *Interface Identifier Management* (IIM) messages of M2UA have the same names and serve similar purposes to the equivalent messages of M3UA, they are not the same. The registration and deregistration messages of M2UA have a different message class value (10) compared to the M3UA routing key management messages (message class value 9). Unlike the M2UA MAUP messages, the IIM messages do not include the M2UA common header. The IIM messages do, of course, include the SCTP common header.

M2PA Operation

As described previously, M2PA effectively enables IP-based SS7 links between an SG and another node (such as an MGC), whereas M2UA simply enables the MGC to have a remote SS7 terminal, with IP used on the interface between the MGC and SG. M2PA also enables IP-based SS7 signaling between two IP signaling points without traversing the traditional SS7 network.

As mentioned earlier in this chapter, three main types of SS7 messages are used in a traditional SS7 network: MSUs, which carry actual user payload; LSSUs, which convey link status information; and FISUs, which are sent when no other data is waiting to be sent. Since M2PA effectively enables IP-based SS7 links, it offers similar types of messaging, with the exception of the FISU for which no M2PA equivalent exists. The reason why M2PA has no FISU equivalent is because the IP network is a shared resource. Constantly sending such messages in an IP network would be wasteful of shared resources. Moreover, the SCTP BEAT message ensures the correct operation of the IP-based links, and the M2PA messages for data transfer include acknowledgment functions. Consequently, two M2PA messages are available: the User Data message and the Link Status message.

Like other adaptation layers, M2PA establishes SCTP associations between M2PA peers. M2PA-initiated SCTP associations use two streams in each direction. One stream is used for the sending of User Data messages, while the other is used for sending Link Status messages.

The User Data message of M2PA is simply an SS7 MSU with a few of the standard SS7 fields excluded. The message contains the *Length Indicator*

(LI), *Service Information Octet* (SIO), and *Signaling Information Field* (SIF). The Flag, *Backward Sequence Number* (BSN), *Backward Indicator Bit* (BIB), *Forward Sequence Number* (FSN), *Forward Indicator Bit* (FIB), and *Check bits* (CK) are not included. The LI is included in the User Data message because some national SS7 variants use the two spare bits of the LI as a priority indicator.

The Link Status message is sent between M2PA peers to indicate the current state of the link and is analogous to the SS7 LSSU. In much the same way as the SS7 LSSU contains a "status" field, the M2PA Link Status message contains a state parameter. The values for the state parameter are as follows:

- **1** Alignment
- **2** Proving Normal
- **3** Proving Emergency
- **4** Ready
- **5** Processor Outage
- **6** Processor Outage Ended
- **7** Busy
- **8** Busy Ended
- **9** Out of Service
- **10** In Service

M2PA Link Alignment Link alignment is the process by which a link is brought into service in a controlled manner, so that one end does not try to send user traffic before the other end is ready. A common situation for link alignment is when a new link is being established.

Once an SCTP association is established, M2PA sends the Link Status Out of Service message to its peer. The actual link activation process is started by MTP3, which sends a Start request to M2PA. This request causes M2PA to send a Link Status Alignment message to its peer and start a timer (typically 10 seconds). A Link Status Alignment message is expected in return, and if it is not received, then M2PA informs MTP3 that the link is out of service. Assuming, however, that a Link Status Alignment message is received from its M2PA peer before the timeout, M2PA begins the link-proving process.

To begin the link-proving process, each M2PA usually sends a Proving Normal Link Status message to its peer at a regular rate. M2PA can, however, use the Proving Emergency Link Status message if requested by

MTP3. The proving duration is typically 10 seconds, but it can be as little as 400 milliseconds in the case of emergency proving.

Once the proving period has ended, M2PA sends Link Status Ready messages to its peer at regular intervals. Upon receipt of a Link Status Ready message in return, or a User Data message, the link is considered to be in service and M2PA sends an In Service message to its peer. The link is now ready for traffic.

In addition to the Link Alignment messages just described, other messages are used to convey fault conditions. The Processor Outage message indicates a failure at a layer above M2PA (such as MTP3) and the Processor Outage Ended message indicates that the failure condition has ceased. The Busy message conveys an indication of receiver congestion to the far end, so that the far end can stop sending messages. The Busy Ended message indicates that the congestion condition has ceased and enables user traffic to flow again.

Interworking SS7 and VoIP Architectures

The primary thrust of this chapter has been to describe how a VoIP network can interwork with an external SS7 network. Finally, we consider any differences that the architecture of the VoIP network might cause—specifically whether the network follows the softswitch or the H.323 model.

Interworking Softswitch and SS7

The concepts behind Sigtran are aimed at the reliable transfer of signaling information. Many of the adaptation layers are SS7 specific and lend to interworking between SS7 and the softswitch architecture. One reason is that the softswitch architecture includes a number of MGs placed close to the sources and sinks of media in the PSTN (at the edges of the IP network). In the distributed softswitch architecture, however, the MGCs that control the MGs may be far removed from the gateways themselves. Thus, having SGs at the edge of the IP network makes sense.

The interworking is typically achieved by the use of at least two SGs that terminate SS7 links and pass SS7 messages to the actual applications at an MGC. The architecture is shown in Figure 7-33 and the protocols used include SCTP, M3UA, and M2PA or M2UA, as previously discussed.

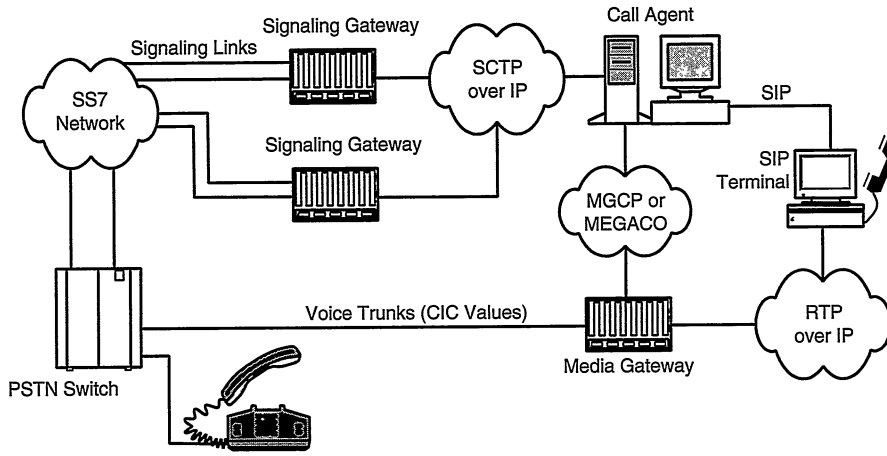


Figure 7-33 Interworking SS7 and softswitch architecture

Figure 7-33 shows the use of SCTP only for transporting SS7-related signaling. The underlying transport for SIP and MGCP/MEGACO is not shown. Although these could be TCP or UDP (UDP only for MGCP), one could easily imagine a situation where SCTP is used. In fact, an Internet draft already exists that addresses the use of SCTP as a transport for SIP.

ISUP Encapsulation in SIP SCTP and the associated adaptation layers provide the capability to convey SS7 application protocol information from an endpoint in the PSTN to an MGC in the center of the VoIP network. The MGC then has access to the signaling information needed to route calls in the VoIP network. SS7 application protocols themselves might or might not be used internally in the VoIP network. Often, protocols such as ISUP are mapped to equivalent VoIP protocols, such as SIP. In fact, SIP-ISUP mapping is the most common, and we have already discussed that topic in Chapter 5, "The Session Initiation Protocol (SIP)." Such mapping works very well if one end of the call is a VoIP node and the other is a PSTN node. A problem may occur, however, when the softswitch network provides a transit function between a circuit-switched network at the originating end and a circuit-switched network at the terminating end.

Because protocols are designed in different ways, one rarely finds a direct match between the messages and parameters of one protocol and those of another. In addition, interworking has to address the differences between state machines, timers, and so on. The result, although workable, is often less than perfect. The reason is because protocol A might be slightly

better in one aspect, while protocol B might be better in another. Consequently, interworking situations often lead to a lowest common denominator result where the functionality offered is limited to that supported by both protocols. For example, SIP header fields enable the inclusion of information that simply does not exist in ISUP. One example of this is the Retry-after header field that just does not map to an equivalent in ISUP.

Equally, ISUP provides information that does not easily map to SIP headers (if at all). Often, such information might need to be mapped to the closest SIP equivalent or perhaps be discarded. This situation is not so bad if a call is placed from a PSTN user to a SIP user, because the SIP user agent server would not be able to interpret such information in any case. Many networks exist today that provide long-distance service using VoIP, however. These networks connect to the PSTN at numerous points. Long-distance calls from one PSTN user to another can transit via the VoIP network, entering the VoIP network at one point and leaving again at another. If ISUP is converted to SIP at the point of ingress and SIP is converted back to ISUP at the point of egress, the result may be that the ISUP messages leaving the network may be different from those that entered the network, as shown in Figure 7-34.

In order to counteract this problem, SIP enables the message body to encapsulate an ISUP message. Of course, SIP messages and responses are used between MGCs and, where appropriate, those messages must contain an SDP message body in order for the MGCs to describe the media to be sent between the MGs they control. Some of the messages can also contain an ISUP message in binary form within the message body. Therefore, the message body can have multiple parts: an SDP session description used by the MGCs and MGs, and an encapsulated ISUP message carried transparently across the network.

Consider the scenario depicted in Figure 7-35. The incoming IAM is mapped to a SIP INVITE that contains an SDP session description as well as the original ISUP message. At the egress point, the SDP information is

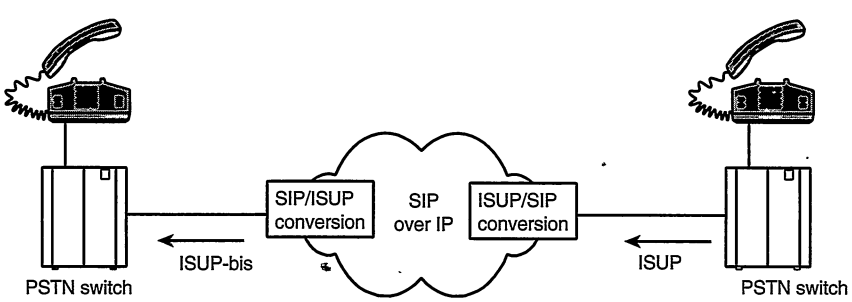


Figure 7-34
SIP in a transit network can be nontransparent to ISUP signaling.

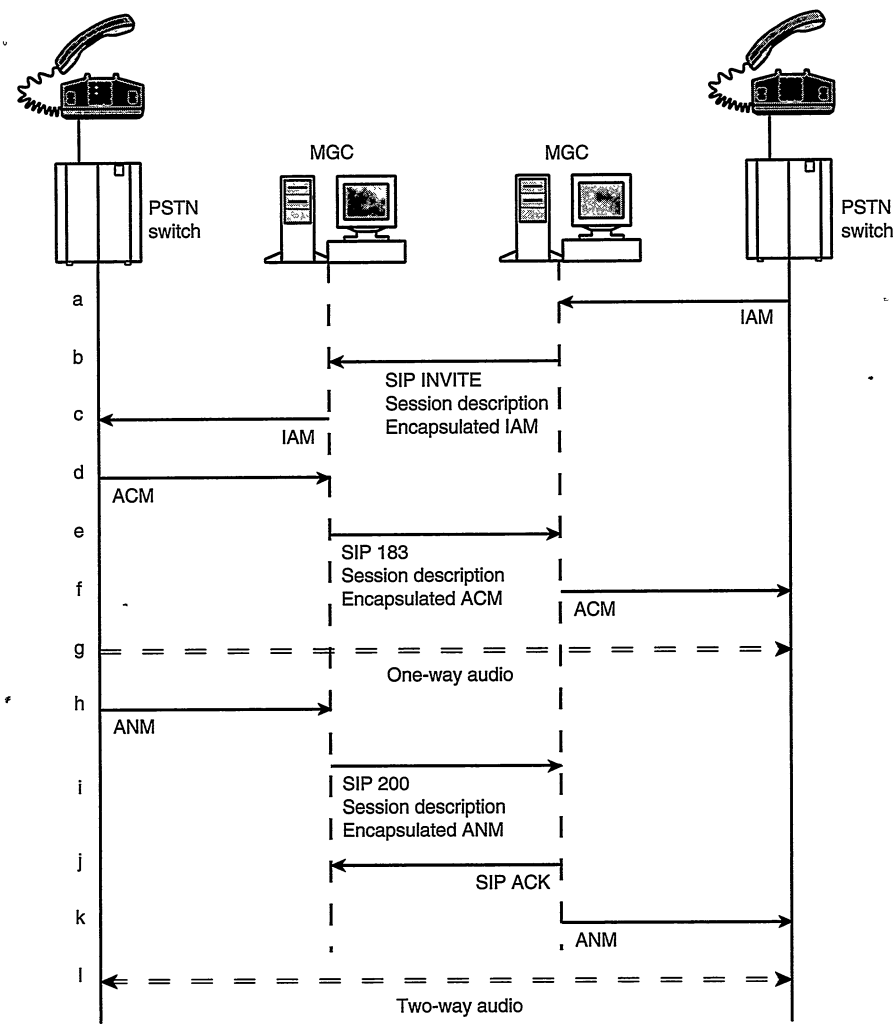


Figure 7-35
SIP encapsulation of ISUP messages

used to set up the RTP session between the gateways, and the ISUP information is extracted to send to the PSTN. When the call is through-connected, the received ACM is mapped to the SIP 183 (session progress) response, which contains a temporary SDP media description, and the message also contains an encapsulated ISUP ACM to pass to the originating PSTN node. A similar process occurs when the call is answered and an ANM message is returned from the called end. Note that the figure does not include SIP messages for the reliable delivery of provisional responses. We can assume, however, that reliable delivery would be applied (the use of the

Provisional Response ACK [PRACK] method). Even though the 183 response is considered provisional, it is important that it be delivered reliably.

In order to encapsulate an ISUP message with a SIP message body, the Content-Type: header field of a SIP message should not indicate application/sdp. Instead it should have the following format:

```
Content-type: multipart/mixed; boundary = SDP-ISUP-boundary
```

The Content-type indicates that the message body contains different sets of information in different formats. The boundary field indicates the boundary between one description and the other within the message body. Its usage is described in RFC 2046, "Multipart Internet Mail Extensions (MIME) Part Two: Media Types."

Within the message body, the beginning of each different payload is indicated by "--boundary," where boundary is as specified in the Content-Type: header. Therefore, if a SIP message were to contain a Content-type field as shown previously, each payload within the message body would be indicated by the string "--SDP-ISUP-boundary." After the last payload is another line with the format "-- boundary--." In our example, the last line would be "--SDP-ISUP-boundary--."

An SIP INVITE using ISUP encapsulation might look something like the following:

```
INVITE SIP: 9725551234@telco2.net SIP/2.0
From: SIP: 2145556789@telco1.net; tag=abcd12345
Call-ID: 123456789@telco1.net
Content-Length: xxx
Content-Type: multipart/mixed; boundary = SDP-ISUP-boundary
MIME-Version: 1.0

-- SDP-ISUP-boundary
Content-Type: application/sdp

v=0
o=collins 22334455 123 IN IP4 444.333.222.111
s=
c= IN IP4 444.333.222.110
t=0 0
m=audio 5555 RTP/AVP 0

-- SDP-ISUP-boundary
Content-Type: application/ISUP; version = ANSI
Content-Encoding: binary
1A 00 01 00 60 00 0A 03 06 0D 03 80 90 A2 07
03 10 18 27 85 31 48 0A 07 03 11 12 74 66 69
53 EA 01 00 00
-- SDP-ISUP-boundary--
```

Interworking H.323 and SS7

Interworking between an H.323 architecture and SS7 could work in a similar manner to interworking between softswitch and SS7. The approach would be to use SGs that terminate SS7 signaling. The SS7 application information would be carried using Sigtran from the SS7 gateways to one or more H.323 gateways.

Equally, however, the gateway itself could simply terminate SS7 links directly. In fact, such an approach has merit. In the H.323 architecture, the gateway itself contains a great deal of the application logic required for call setup, and the gateway is also the entity that performs the media conversion. The fact that the gateway holds the application logic means that it must send and receive call control signaling information. The fact that the gateway performs the media conversion means that it should be placed at the edge of the IP network. Since the gateway is at the edge of the network, it is likely easier and more efficient for the gateway itself to terminate SS7 links from the PSTN directly. The architecture would be as shown in Figure 7-36.

In fact, interworking SS7 with H.323 is in some ways an easier proposition than interworking SS7 and softswitch (at least for ISUP). The reason is because a close relationship exists between ISUP and Q.931. After all,

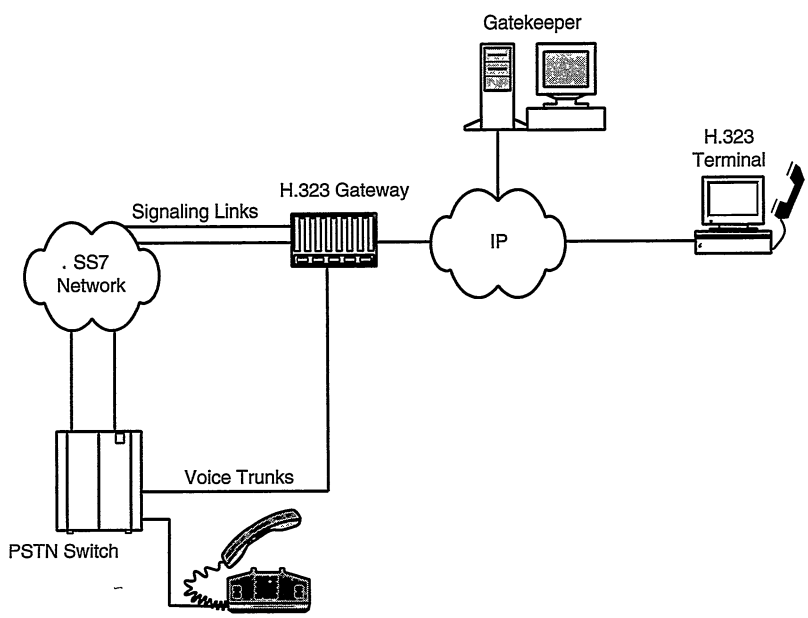


Figure 7-36
Interworking SS7 and H.323 architecture

ISUP is the ISDN User Part, the interswitch network protocol corresponding to the ISDN access protocol Q.931. Consequently, mapping is easy between Q.931 messages and ISUP messages. For example, the setup message in Q.931 maps to IAM in ISUP, and ACM in ISUP maps to the Connect message in Q.931. Therefore, if an H.323 network were to receive a call from the PSTN using ISUP, the signaling would be similar to that shown in Figure 7-37.

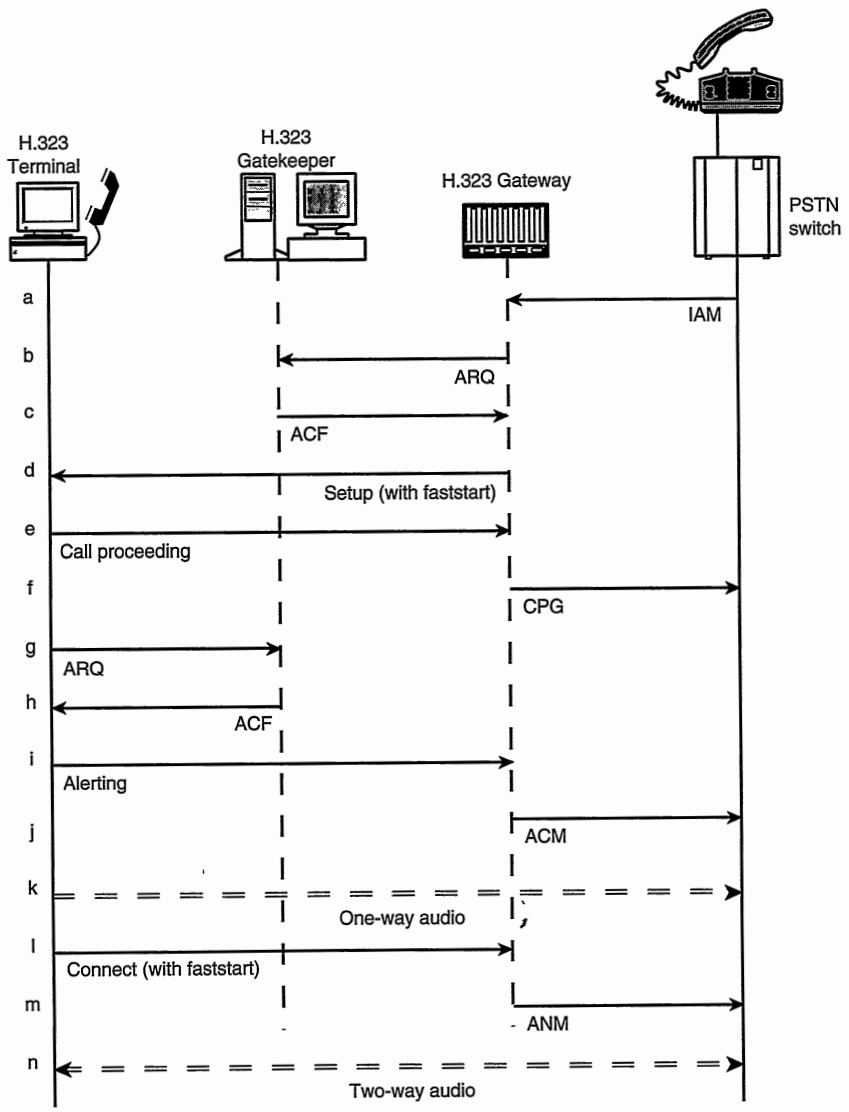


Figure 7-37
ISUP/H.323 interworking

The example assumes the use of the H.323 Fast Connect procedure. If the called H.323 terminal did not support that procedure, then the Connect message would not include a faststart element. In that case, the arrival of the Connect message at the gateway would not immediately lead to an ANM back to the PSTN. Rather, the logical channel procedures of H.245 would first be used to establish the media path between the gateway and the called terminal.

Motorola - Exhibit 1022 - Page 397

CHAPTER 8

Quality of Service (QoS)

Motorola - Exhibit 1022 - Page 398

To many, *quality of service* (QoS) is one of the biggest, if not *the* biggest, issue facing *Voice over IP* (VoIP). QoS (or the lack thereof) is one reason why many of the VoIP solutions on the market today that provide voice over the Internet are free services. The philosophy behind such offerings is that customers can hardly complain about poor quality when they are being provided with a free service. Such a philosophy is fine if the business model is based upon advertising or some other revenue stream rather than paying subscribers. Such VoIP providers may choose to continue providing free service by sending voice over the Internet and not worrying about QoS. For many others, however, the massive revenue generated by the *Public Switched Telephone Network* (PSTN) operators every year is very attractive. Competitors to traditional local and long-distance providers would like to win some of that business, and they see VoIP as a means to do so. The big hurdle is QoS.

Not only do new carriers see VoIP as a means to compete with traditional carriers, but both existing and new carriers would like to have just a single multifunctional network to carry both voice and data. The support of both voice and data traffic on a single network means lower cost. Again, however, the challenge is in ensuring a high-quality service and making sure that one type of traffic does not consume excessive network resources to the detriment of other types of traffic.

This chapter focuses on a number of technical solutions for providing QoS in a VoIP network. The various solutions are described separately, followed by a description of how they may be combined to complement each other. First, however, it is worth considering what QoS means and why it is such an issue for VoIP networks.

The Need for QoS

QoS is a collective measure of the level of service delivered to a customer. It can be considered as the level of assurance from a particular application that the network can meet its service requirements. From a technical perspective, QoS can be characterized by several performance criteria such as availability (low downtime), throughput, connection setup time, the percentage of successful transmissions, and the speed of fault detection and correction. In an *Internet Protocol* (IP) network, QoS can be measured in terms of bandwidth, packet loss, delay, and jitter. In order to provide a high QoS, the IP network needs to provide assurances that, for a given session or

Motorola - Exhibit 1022 - Page 399

a set of sessions, the measurement of these characteristics will fall within certain bounds.

Unfortunately or fortunately (depending on your viewpoint), IP is by nature a best-effort service and offers no guarantees. For this reason, the *Transmission Control Protocol* (TCP) was developed as a layer above IP to ensure error-free, in-sequence delivery of information. Unfortunately, TCP provides this service at the expense of some delay, which is not a big problem for typical IP applications such as e-mail, web browsing, and file transfer. The picture changes significantly, however, when real-time applications such as voice are added to the mix. These applications are time sensitive to the extent that delays and packet loss can very easily make a service unusable. When it comes to voice, delay and jitter are of particular significance. Consequently, TCP cannot be used, which is why the *User Datagram Protocol* (UDP) is used instead (as described in Chapter 2, "Transporting Voice by Using IP").

Using UDP for transporting voice is fine, provided that packet loss is relatively low and the network has relatively little congestion. Traffic in an IP network can be bursty and unpredictable, however, which can lead to a situation where one application consumes network resources for a period of time (however brief), forcing packets from other applications to wait and/or be discarded. Again, TCP can take care of this for non-real-time applications, but for other applications that use UDP, the application itself needs to take care of retransmission.

Unfortunately, however, retransmission is not an option for voice because it introduces delay. If just a few packets are lost, some voice-coding schemes can handle the interruption by replaying previous packets or by extrapolating from previous packets. In the case of significant packet loss, however, voice-coding schemes will be unable to cope, which means that a speaker will have to repeat what he or she just said. Such a situation is hardly desirable, and certainly not one that a paying customer will tolerate. Therefore, we need to ensure that significant delay or packet loss is avoided.

Why should we be concerned about all these issues? The short answer is money. The long answer is that high-quality service is imperative if an operator is to attract and retain paying subscribers. If a provider is to charge for voice service, no matter how low the price, then the quality of that voice service must at least match that provided by traditional circuit-switched network operators. When it comes to quality, however, circuit switching has a distinct advantage. Because a dedicated two-way transmission pipe is established between the parties in a call and because no buffering of packets or packet loss takes place, the quality is excellent. High quality is to be

expected, because circuit switching was designed specifically for voice from the outset.

The problem with circuit switching is that it is ill suited to other forms of communication. IP, on the other hand, is well suited to non-real-time communication and can also be made suitable for real-time communication, provided that QoS solutions are in place. Fortunately, solutions do exist. The primary thrust of this chapter is to describe these technical solutions.

End-to-End QoS

Before delving into the technical solutions for QoS, it should be pointed out that QoS is not just a technical matter on one operator's network. In general, a customer does not know or care about how a provider gets a call from one place to another, provided that the call gets there and meets the customer's quality expectations. This means that QoS must be end to end and requires the support of all networks in the chain. A call might originate on one provider's network and terminate on another provider's network, perhaps traversing other networks in between. Each of the networks must cooperate to ensure that the quality provided matches what is expected. This requirement raises the issue of *Service Level Agreements* (SLAs) between different operators. SLAs are agreements in which operators make commitments to each other regarding the type and QoS to be offered to each other and the penalties involved if such commitments are not met. We will discuss SLAs again later in the section "The Need for SLAs."

Meanwhile, however, we must again remember that VoIP and voice over the Internet are not necessarily the same thing. We must recognize that the Internet is a huge collection of different networks and is not managed by any one entity. For those parts that are actively managed, they are not necessarily managed in the same way or according to the same set of criteria. Although SLAs may be possible between certain VoIP carriers, too many networks are involved on the Internet to think that it will be possible to place a call from anywhere to anywhere else over the Internet and always experience superior quality. Many operators readily agree that a particular QoS can be provided in managed networks, but shake their heads at the suggestion that QoS is a realistic expectation for voice over the Internet. For this reason, this book makes a significant distinction between VoIP and voice over the Internet.

This does not mean that QoS is only achievable when a call stays within a single network. Rather, SLAs at least offer the potential of ensuring quality across multiple networks. Until all *Internet service providers* (ISPs) offer the same quality commitments and implement equivalent technical solu-

tions, however, anywhere-to-anywhere voice over the Internet is likely to remain a best-effort, low-quality service.

It's Not Just the Network

Not only is QoS not confined to a single operator's network, but it is not confined to network issues in general. In other words, a quality service means a lot more than just good voice quality. In the competitive environment of telecommunications, low prices and advanced features may attract customers, but the best way to retain customers is to provide a consistently high service across all facets of the business. Given that a potentially high cost may be associated with acquiring a customer in the first place, the last thing a provider wants is to see a customer leave and go to a competitor. Therefore, QoS is provided not only by technical solutions in the network, but it is also provided through superior customer service, rapid service provisioning, 100 percent accurate billing, clear and concise product descriptions, and so on. Although this is a technical book that focuses on network solutions, we must remember that those network solutions provide only one piece of the puzzle when it comes to overall service quality.

Overview of QoS Solutions

A number of solutions have been developed to address the QoS issue. These solutions approach the problem from various angles. One approach is to ensure that resources for a given session are available and reserved for that session prior to establishing the session. Conceptually, this approach has certain similarities to circuit switching, where the bandwidth needed for a call is reserved for that call before the called party's phone rings.

Another approach is to categorize traffic into different classes or priorities with higher-priority values assigned to real-time applications, such as voice, and lower-priority values assigned to non-real-time traffic, such as e-mail. In some ways, this approach is easier to implement, but it requires that no one application can totally shut another out. Just because there is voice on a network that supports voice and data applications does not mean that the data never gets a reasonable share of resources. Therefore, the need exists to implement fair resource allocation techniques.

Finally, perhaps the most common technique today is overprovisioning the network, that is, provisioning greater bandwidth than the traffic

demand would require. After all, a lack of bandwidth is often the reason for packet delay or loss. Bandwidth costs money, however, and needs to be utilized efficiently.

More Bandwidth

Allocating more bandwidth might sound like a very simplistic and expensive approach to QoS—simple because it does not require major system development, but expensive because it means significant overbuild. This overbuild would need to exist so that network resources would be available in times of traffic bursts. Unfortunately, the additional bandwidth would remain unused for most of the time. That method would be a very inefficient way of solving the QoS problem, but it does have some merit and should not be dismissed completely.

We live in an age where huge advances continue to be made in squeezing bandwidth from facilities that appeared to have reached their limit. Not that long ago, a 9,600-baud modem was considered almost a utopian situation for a modem connection. Now, 56 Kbps modems are standard for dial-up access, and *Digital Subscriber Line* (DSL) technology means that homes can have several megabits per second in each direction.[1] And this is just in the access part of the network. In the core of the network, we now have *Dense Wave Division Multiplexing* (DWDM) offering the capability to support tens of gigabits per second and hundreds of gigabits per second in the future on a single optical fiber. Consequently, as these new transmission techniques have become available, the cost of bandwidth has been decreasing and will likely continue to decrease.

On the other hand, however, Moore's Law says that computing power doubles roughly every 18 months, which means that applications are demanding bandwidth at an ever-increasing rate. In addition, the number of Internet users continues to increase. The result is that the demand for bandwidth is increasing rapidly. Although we currently have a glut of transmission capacity (a good deal of excess dark fiber currently exists), that situation will not last forever. Historically, bandwidth availability and bandwidth demand have tended to move almost in lock-step. This is simply a supply and demand issue. If bandwidth is scarce, then applications that

[1]DSL has many variants. The most common is *Asymmetric DSL* (ADSL), which can theoretically provide up to 640 Kbps uplink and up to 9 Mbps downlink. Whether a given customer experiences these rates is a different issue with commercial as well as technical considerations.

Motorola - Exhibit 1022 - Page 403

require high bandwidth are either not developed because they would be impossible or too expensive to use, or they are deployed only in a limited scope. Once more bandwidth becomes available, however, a major barrier to the deployment of such applications is removed and application developers can proceed with implementing new ideas. They can do so without worrying about a bandwidth demand that might have been unconscionable a short time earlier.

Therefore, excess bandwidth is often needed, if only to support additional traffic as demand continues to grow. Although additional bandwidth is a necessity for a network that is required to support voice in addition to the data traffic that may have been carried all along, excess bandwidth is not by itself a complete solution to the QoS issue. Mechanisms need to be in place to manage the available bandwidth so that it is used most effectively to support the services provided.

QoS Protocols and Architectures

The following is a high-level overview of the QoS solutions developed within the *Internet Engineering Task Force* (IETF). The details of these solutions are provided in subsequent sections of this chapter.

Resource Reservation Resource reservation techniques for IP networks are specified in RFC 2205 on the *Resource Reservation Protocol* (RSVP), which is part of the IETF integrated services suite. RSVP is a protocol that enables resources to be reserved for a given session (or sessions) prior to any attempt to exchange media between the participants. Of the solutions available, RSVP is the most complex, but it is also the solution that comes closest to circuit emulation within the IP network. RSVP provides strong QoS guarantees, significant granularity of resource allocation, and significant feedback to applications and users.

RSVP currently offers two levels of service. The first level is *guaranteed*, which comes as close as possible to circuit emulation. The second level is *controlled load*, which is equivalent to the service that would be provided in a best-effort network under no-load conditions.

Basically, RSVP works as depicted in Figure 8-1. A sender first issues a PATH message to the far end via a number of routers. The PATH message contains a *traffic specification* (TSpec), which provides details of the data that the sender expects to send in terms of bandwidth requirement and packet size. Each RSVP-enabled router along the way establishes a *path state* that includes the previous source address of the PATH message (that

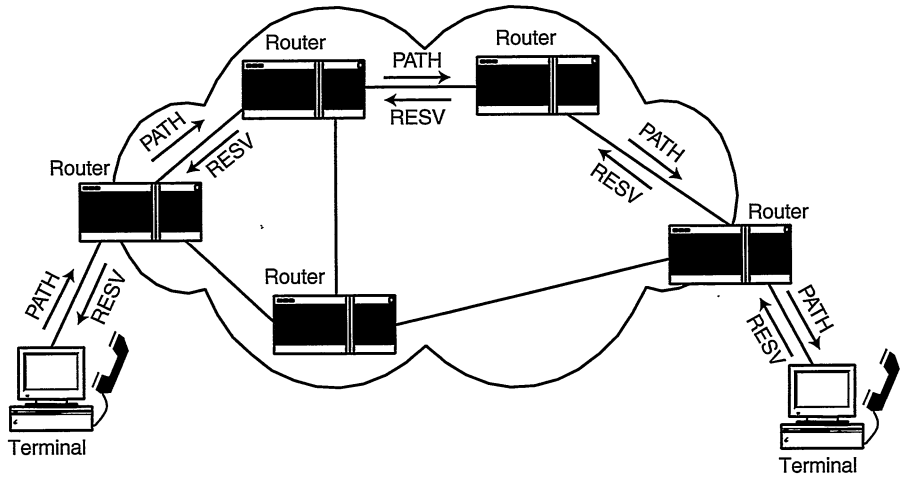


Figure 8-1
RSVP used for resource reservation

is, the next hop back towards the sender). The receiver of the PATH message responds with a *reservation request* (RESV) message that includes a flowspec. The flowspec includes a TSpec and information about the type of reservation service requested, such as controlled-load or guaranteed service.

The RESV message travels back to the sender along the same route that the PATH message took (but in reverse). At each router, the requested resources are allocated, assuming that they are available and that the receiver has the authority to make the request. Finally, the RESV message reaches the sender with a confirmation that resources have been reserved.

One interesting point about RSVP is that reservations are made by the receiver, not by the sender of data. This approach accommodates multicast transport, where there may be large numbers of receivers and only one sender.

Note that RSVP is a control protocol that does not carry user data. The user data (voice) is transported later using the *Real-Time Transport Protocol* (RTP). This occurs only after the reservation procedures have been performed. The reservations that RSVP makes are soft, which means that they need to be refreshed on a regular basis by the receiver(s).

Differentiated Service (DiffServ) *Differentiated Service* (DiffServ) is a relatively simple means for prioritizing different types of traffic. The DiffServ protocol is described in RFC 2475, "An Architecture for Differentiated Services." Basically, DiffServ makes use of the IPv4 *Type of Service* (ToS) field and the equivalent IPv6 Traffic Class field. The portion of the ToS/Traffic Class field used by DiffServ is known as the DS field. The field

is used in specific ways to mark a given stream as requiring a particular type of forwarding. The type of forwarding to be applied is known as *per-hop behavior* (PHB), of which DiffServ defines two types: *Expedited Forwarding* (EF) and *Assured Forwarding* (AF).

RFC 3246 defines EF as a service in which a given traffic stream is assigned a minimum departure rate from a given node, one that is greater than the arrival rate at the same node provided that the arrival rate does not exceed a pre-agreed maximum. This technique ensures that queuing delays are removed. Since queuing delays are a major cause of end-to-end delay and the main cause of jitter, this technique ensures that delay and jitter are minimized. In fact, EF can provide a service that is equivalent to a virtual leased line.

RFC 2597 defines AF as a service in which packets from a given source are forwarded with a high probability provided that the traffic from that source does not exceed some pre-agreed maximum. AF defines four classes, with each class allocated a certain amount of resources (buffer space and bandwidth) within a router. Within each class, a given packet can have one of three drop rates. At a given router, if there is congestion within the resources allocated to a given AF class, then the packets with the highest drop rate values will be discarded first so that packets with a lower drop rate value receive some protection. In order to work well, it is necessary that the incoming traffic does not have packets with a high percentage of low drop rates. After all, the purpose is to ensure that the highest-priority packets get through in the case of congestion and that cannot happen if all the packets have the highest priority.

Label Switching Label switching is something that has seen significant interest in the Internet community, and significant effort has been placed behind the definition of a protocol called *Multiprotocol Label Switching* (MPLS). Like DiffServ, MPLS marks traffic at the entrance to the network. The primary function of the marking is not to allocate priority within a router, however, but to determine the next hop in the path from the source to destination.

MPLS involves the attachment of a short label to a packet in front of the IP header. This procedure is effectively similar to inserting a new layer between the IP layer and the underlying link layer of the *Open System Interconnection* (OSI) model. The label contains all the information that a router needs to forward a packet. The value of a label can be used to look up the next hop in the path and forward the packet to the next router. The difference between this approach and standard IP routing is that the match is exact and is not a case of looking for the longest match (the match with the

longest subnet mask). This process enables faster routing decisions within routers.

The MPLS label identifies something called a *Forwarding Equivalence class* (FEC). All packets that belong to a given FEC are treated equally for the purposes of forwarding. All packets in a given stream of data, such as a voice call, will have the same FEC and receive the same forwarding treatment. Therefore, we can ensure that the forwarding treatment applied to a given stream can be set up such that all packets from A to B follow exactly the same path. If that stream has a particular bandwidth requirement, then that bandwidth can be allocated at the start of the session. Thus, we can ensure that a given stream has the bandwidth that it needs and the packets that make up the stream arrive in the same order as transmitted. Hence, a higher QoS is provided. Not only can we ensure that a given session can be given the required bandwidth and that all packets in a session are forwarded the same way, but we can apply the same technique to a traffic aggregate. For example, we can apply the same FEC to all voice packets between two *media gateways* (MGs) that are geographically separate.

Consider a VoIP network that provides long-distance voice service. Such a network would have numerous gateways in different locations. Provided that we have an understanding of the traffic demand between a given pair of media gateways, we can allocate a specific path (with specific resources) through the IP network to handle the traffic between those gateways. Also, if the traffic between the gateways does not exceed expectations, then the traffic will always have access to the necessary network resources to ensure that QoS requirements are met. This approach (whereby we allocate specific paths and network resources to specific types of traffic) is known as *traffic engineering*. In fact, MPLS is effectively a traffic-engineering tool and it ensures QoS by enabling us to ensure that the network is correctly engineered to meet our QoS objectives.

QoS Policies

Given that mechanisms exist to apply higher QoS to certain streams compared to others, the issue arises as to which QoS levels should be applied to which types of traffic. This raises the issue of QoS policies. Although protocols such as RSVP, DiffServ, and MPLS provide the mechanisms to differentiate traffic and allocate resources to specific types of traffic, QoS policies specify how those mechanisms are used.

One important implication of QoS is the fact that certain users or types of traffic will receive better service than others. In many cases, the QoS for