RADVISION®
an Avaya company

White Paper

SIP Server
Technical Overview

NOTICE

For further information contact RADVISION or your local distributor or reseller.

SIP Server version 2.0, April, 2004
Publication 1
http://www.radvision.com

CONTENTS

1 *SIP Server Technical Overview*

Introduction 1
What is a SIP Server? 1
Registrar Server 3
Redirect Server 4
Proxy Server 6
- Stateful and Stateless Proxies 7
- Request Validation 10
- Address Resolution 12
- Determining the Target-set 12
- DNS Resolution 13
- Stateful Message Forwarding 13
- Symmetric Record-Route 18
- Loose-Routing 19
- Forking 20
- Authentication 23
- Loop detection and Max Forwards 24
- Message Spiraling 25
- Outbound Proxy 25
- Policy 26

B2BUA 27
Events Server 30
- Event Package Name 31
- Subscribe Duration 31
- Authentication 31

Presence Server 32

2 *RADVISION SIP Server Platform*

SIP Server Development Challenges 37
Minimizing Development Time and Cost 38
RADVISION SIP Server Platform 38
Standards Compliance 39
Interoperability 40
Robustness 40
Extensibility 41
Flexibility and Customization 41
Performance and Capacity 42
Maintainability 42
Portability 43
SIP Server Platform Capabilities 43
Standards compliance 43
General Capabilities 44
Proxy Capabilities 45
Registrar Capabilities 45
Redirect Capabilities 46
Back-to-Back User Agent 46
Events Server 47
Non-SIP capabilities 48
Application Layout 48
SIP Server Platform Architecture 51
SIP Server Platform Coding 52
SIP Server Platform Package 52
Operating System Support 53
RADVISION Family of SIP Development Solutions 53

1

SIP SERVER TECHNICAL OVERVIEW

INTRODUCTION

SIP Servers are essential network elements that enable SIP endpoints to exchange messages, register user location, and seamlessly move between networks. SIP Servers enable network operators to install routing and security policies, authenticate users and manage user locations.

SIP Server applications may take many forms, but the SIP standard defines three general types of server functionality that apply to all—Proxy, Redirect and Registrar servers. These standard functionalities can be used according to the needs of the specific implementation. A SIP Server can also function as a Presence Server or a Back-to-Back User Agent (B2BUA). In addition, since SIP defines other event types, such as Winfo and Register, a SIP Server may function as an Events Server for handling the various SIP events.

With the advance of SIP, server logic has become increasingly complex. SIP Servers need to deal with varying network topologies (such as public Internet networks, cellular networks, broadband residential networks), complex routing policies, security and SIP extensions. SIP Servers often need to handle high message/transaction rates and yield real-time performance and scalability, high throughput, and low delay. This chapter discusses the protocol aspects of SIP Server behavior and the usage of the RADVISION SIP Server Platform to address the challenges of effective SIP Server development.

WHAT IS A SIP SERVER?

The SIP baseline specification RFC 3261 (previously RFC 2543bis) divides SIP Server functionality into the following parts:

- SIP Registrar Server—handles location registration messages.
- SIP Redirect Server—returns “contact this address” responses.
- SIP Proxy Server—forwards SIP requests and responses.

In addition to the functionality of the above three SIP Servers, the SIP specification and its related drafts also refers to two other types of SIP Server:

- Back-2-Back User Agent (B2BUA)—acts as UA server to one side and as UA client to the other side.
- Presence Server—provides a Watcher with presence information of a Presentity, as defined in *draft-ietf-simple-presence-07.txt* and in *draft-ietf-impp-cpim-pidf-06.txt.* Also provides Watcher information to Presentities, as defined in *draft-ietf-simple-winfo-package-05.txt*.
- Events Server—in this document, the term "Events Server" is used for the network element, acting as Notifier, which receives SUBSCRIBE messages for different SIP Event types, such as Registration and MWI (Message Waiting Indication). This type of general Notifier functionality is defined in RFC 3265: SIP Specific Event Notification.

Note The document version number may change as newer versions are published.

The authors of SIP have made this functional separation for ease of understanding, and not necessarily as a guideline for implementers. In other words, it is acceptable to have one "box" do more than just one type of server functionality. For example, a SIP Server may benefit from having a routing capability to forward some SIP messages as a proxy and redirect others as a redirect server. There are existing SIP Server implementations that have a few types of functionalities in one product running in the same process space, while others prefer to distribute functionality, with different machines performing different server functions. In fact, the SIP standard does not prohibit a SIP UA (endpoint) from having server capabilities, since UAs may redirect requests or process registrations. It is important to understand that proxy, redirect and registrar are functionalities that may be put to use in any way that benefits the application.

The behavior of SIP Servers is discussed in RFC 3261, but you can find more detailed examples of message flows in “SIP Call Flow Examples” in the RFC 3665 document *draft-ietf-sipping-call-flows-01.txt*, found at the IETF site or one of its mirror sites. (http://www.ietf.org/rfc/rfc3665.txt?number=3665)

Note The document version number may change as newer versions are published.

REGISTRAR SERVER

The SIP standard defines a registrar server as “a server that accepts REGISTER requests and places the information it receives in those requests into the location service for the domain it handles”. REGISTER requests are generated by clients in order to establish or remove a mapping between their externally known SIP address(es) and the address(es) they wish to be contacted at. The REGISTER request can also be used to retrieve all the existing mappings saved for a specific address.

The Registrar processes the REGISTER request for a specific set of domains. It uses a “location service”—an abstract location database—in order to store and retrieve location information. The location service may run on a remote machine and may be contacted using any appropriate protocol (such as LDAP). The SIP standard leaves this decision to the implementation. Some implementations may co-locate the location service and the registrar server on the same machine.

A registrar server may authenticate incoming REGISTER requests using the 401 (Unauthorized) response.

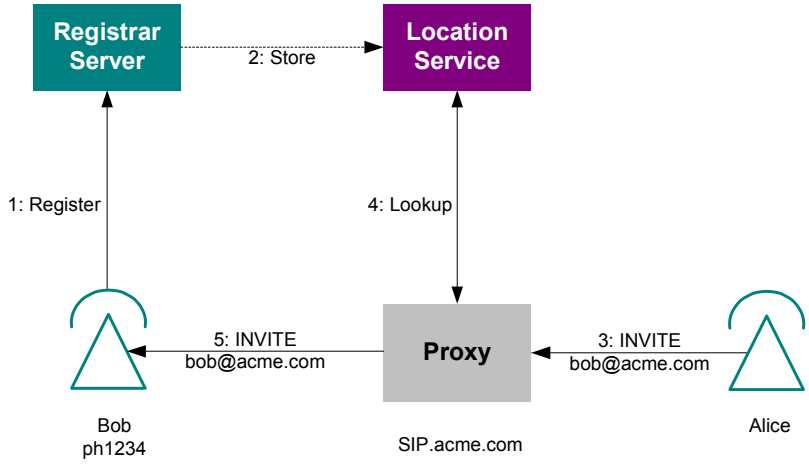

Figure 1-1 *Registration Process*

REDIRECT SERVER

Redirect server functionality is the simplest of the three functionalities. A redirect server receives SIP requests and responds with 3xx (redirection) responses, directing the client to contact an alternate set of SIP addresses. The alternate addresses are returned as Contact headers in the response message.

Table 1-1 shows the 3xx responses that are currently defined by SIP.

Table 1-1 *3xx Responses*

Response	Meaning
300 Multiple Choices	The address in the request was resolved to several choices, each with its own specific location, and the user (or UA) can select a preferred communication end point and redirect its request to that location. This status response is appropriate if the callee can be reached at several different locations and the server cannot, or prefers not, to proxy the request **Note:** 301 and 302 responses can also contain multiple Contact address. The difference is that they convey a more specific reason for the redirection.
301 Moved Permanently	The user can no longer be found at the address specified in the Request-URI (the destination address in the request), and the requesting client should retry at the new address given by the Contact header field.
302 Moved Temporarily	The user is temporarily available at a different address(es). The duration of validity of these addresses may be expressed in the Contact header.
305 Use Proxy	The requested destination address must be accessed through the proxy specified in the Contact field.
380 Alternative Service	The call was not successful, but alternative services are possible. The alternative services are described in the message body of the response. The use of this response code is still not defined in SIP and is for future use.

In most cases, 301 and 302 responses are used by redirect servers. 300 can also be used, although it is more ambiguous to the calling client.

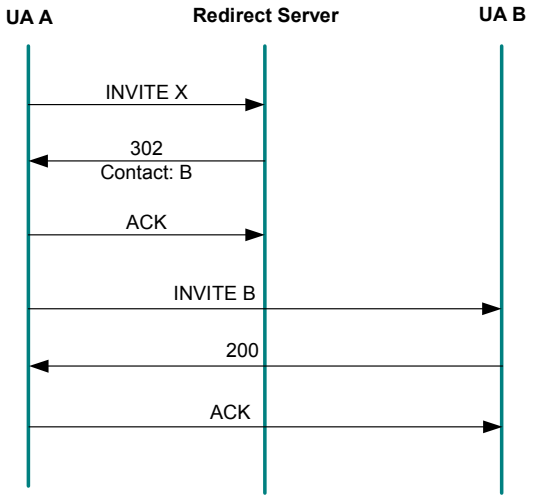

Figure 1-2 *Request Redirection*

The scenario in Figure 1-2 illustrates a redirection scenario. Note that the second INVITE request is generated with the same dialog identifiers, Call-ID, and To and From headers as the first INVITE request, but with a different CSeq value.

Redirection allows servers to push back routing information for a request in a response to the client, thereby aiding in locating the target of the request, while taking themselves out of the loop of further messaging for this transaction. Redirect servers typically are not aware of the state of dialogs (calls, subscriptions), only of the state of the individual transactions they are handling, making them transaction-stateful elements. Redirection is designed as a simple and quick procedure, allowing for redirect servers to be highly scalable and to yield high-performance. Redirect servers are sometimes used as a load balancing devices.

Redirect servers may request user authentication with the use of the 401 response (Proxy Authentication Required) as explained below.

PROXY SERVER

The SIP standard defines SIP proxies as “elements that route SIP requests to User Agent Servers (UAS) and SIP responses to User Agent Clients (UAC). A request may traverse several proxies on its way to a UAS. Each will make

routing decisions, modifying the request before forwarding it to the next element. Responses will route through the same set of proxies traversed by the request in the reverse order."

It is useful to view Proxy Servers as SIP-level routers that forward SIP requests and responses. However SIP proxies employ routing logic that is typically more sophisticated than just automatically forwarding messages based on a routing table. The SIP standard allows proxies to perform actions such as validate requests, authenticate users, fork requests, resolve addresses, cancel pending calls, Record-Route and Loose-Route, and detect and handle loops. The versatility of SIP proxies allows the operator/system administrator to use the proxies for different purposes and in different locations in the network (such as edge proxy, core proxy and enterprise proxy). This versatility also allows for the creation of a variety of proxy policies, such as routing calls only for authenticated users that have no standing debt to the network service provider operating the proxy. Proxies can be placed at the network of the service provider or at the enterprise or SOHO premises. The 3GPP IMS architecture, for example, uses proxies known as Call State Control Functions (CSCF in 3GPP terminology) of different kinds for various purposes—as the first hop server that communicates with the handset; as the element that accesses location repositories and application servers and triggers services; and as the edge element that communicates with proxies in foreign networks.

A proxy server is designed to be mostly transparent to UAs. Proxy servers are allowed to change messages only in specific and limited ways. For example, a proxy is not allowed to modify the SDP body of an INVITE. Apart from a few exceptions, proxies cannot generate requests at their own initiative. Therefore a proxy cannot terminate an existing call by generating a BYE request.

Stateful and Stateless Proxies

The SIP specification defines two types of SIP proxies:

- Stateful proxy
- Stateless proxy

Stateless proxy

A stateless proxy is a "simple message forwarder", as described in the SIP standard. When receiving a request, the stateless proxy processes the request much like a stateful proxy, however the stateless proxy forwards the message in a stateless fashion—without saving any transaction context. This means that once the message is forwarded the proxy "forgets" ever handling this message. Stateless forwarding allows for improved performance and scalability, but has some consequences:

- A stateless proxy cannot associate responses with forwarded requests because it retains no knowledge of the requests it has forwarded. Therefore, the proxy application cannot know if a transaction was successful or not.
- A stateless proxy cannot associate retransmissions of requests and responses with the previous instance of these messages. It processes retransmissions exactly as if this is the first copy of the message it received.

Note Stateless proxies need to use a routing algorithm that always routes copies of the same message to the same destination. If this rule is not enforced, a retransmission of an ACK message, for example, may be routed to a different UA than the one that returned the 200 response.

- If the message is lost, the proxy will not retransmit it. Retransmission is the responsibility of stateful UAs or proxies

Because of their high-throughput capabilities, stateless proxies are often used at the core of carrier and service provider networks assisting in forwarding SIP messages on the network. Stateless proxies may also be used as load balancers.

Note that once a proxy decides to return a non-2xx response, the proxy cannot do this statelessly and has to retain a transaction state.

STATEFUL PROXY

When stateful, the proxy processes transactions rather than individual messages. The proxy manages two types of transactions—server transactions to receive requests and return responses, and client transactions to send requests and receive responses. An incoming request is processed by a server transaction and then forwarded downstream by one or more client transactions (there may be more than one in the case of parallel forking, for example). An incoming response is received by the matching client transaction and forwarded back to the server transaction. Associating between client and server transactions and managing the overall state of this request is the responsibility of the Proxy Core Object (*ProxyCoreObj*). The *ProxyCoreObj* chooses the destination address(es)

and instantiates one or more client transaction objects accordingly. The *ProxyCoreObj* also collects the responses from the different client transactions and chooses the response(s) that will be sent upstream via the server transaction.

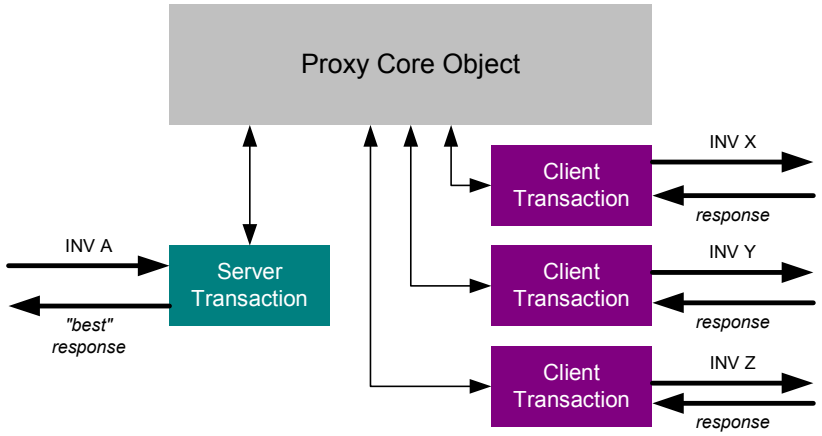

Figure 1-3 *Stateful Proxy Model*

A stateful proxy is aware of the state of transactions and message history, and can therefore perform better-informed processing on incoming messages. For example, a stateful proxy can identify a retransmission of an incoming message and forward the message only in situations that require retransmission forwarding, whereas a stateless proxy cannot identify retransmissions and has to forward every message it encounters. A stateful proxy can also generate retransmissions in cases of message loss. In addition, a stateful proxy can locally process incoming CANCEL requests and generate CANCEL requests as needed. Forking is also more natural for a stateful proxy (especially sequential forking).

Statefulness, however, has the following drawbacks:

- **Memory consumption**

  The stateful proxy retains more memory per processed message than a stateless proxy, and for a longer duration. This has a negative impact on the maximal capacity of the proxy and limits the number of concurrent calls/transactions it can handle. Certain code optimizations which are SIP-specific can compensate for this and bring memory consumption to the level required by high-capacity proxies.

- **Throughput**

  A stateful proxy has to spend more CPU cycles on message processing—mapping messages to transactions, managing transaction state machines, processing transaction timers and associating client and server transactions. This extra processing reduces proxy capacity in terms of performance (maximal number of processed requests per second). For these reasons, developing a high-performance stateful proxy has proved to be non-trivial, and requires special optimizations in the design of the proxy. The ability to customize and to fine tune the proxy through flexible configuration is also crucial for achieving high performance.

- **Implementation complexity**

  A stateful proxy does more than just forward requests. Certain logic needs to be employed in order to deal with actions such as parallel forking (for example, choosing the best response), CANCEL, recursion on 3xx responses, and handling ACK for 2xx responses. The evolution of the SIP standard introduced numerous “hard-to-deal-with” special cases for proxies that require the use of a number of special techniques, such as storing hashed context information in certain request message fields and later retrieving the information from responses. All of these factors make the implementation of the proxy non-trivial and add special cases that have to be tested.

- **Underlying SIP Stack complexity**

  A proxy requires certain flexibilities from the underlying SIP Stack that a UA does not. This is especially true for the Transport and Transaction layers. This requires the SIP Stack to be more modular and export more layers of API than a UA-oriented stack.

REQUEST VALIDATION

Before routing a request, a SIP Server (proxy or redirect) needs to validate the request to make sure it can actually proceed with processing this message. The message has to pass the following validity checks:

- **Reasonable syntax check**

  The request must be well-formed enough to be handled by the server. However, this applies only to specific fields in the message which the server must process. All other parts should not be checked or fixed by the proxy.

- **URI scheme check**

  The URI scheme (for example, “ftp” in ftp:radvision.com) must be URI scheme the proxy understands and knows how to route. If not, the proxy must return a 416 (Unsupported URI Scheme) response.

- **Max-Forwards check**

  Max-Forwards is a message field that indicates how many more hops the message is allowed to traverse. Each proxy that handles the message decrements this number by one (similar to the TTL field in certain protocols). If the message contains a Max-Forwards value of zero, the proxy must return a 483 (Too many hops) response. This mechanism allows preventing a message from going into an endless loop between a set of proxies.

- **Loop Detection (Optional)**

  A proxy may check for loops by executing a loop detection algorithm on the Via list contained in the message. The proxy checks that it did not previously handle this message. If it did previously handle the message, it verifies that the message contains different values in the fields that influence routing decisions (such as Request-URI, From and To). If the proxy identifies a loop condition, it rejects the message with a 482 response code (Loop Detected). This check is now optional because the Max-Forwards field has become mandatory.

- **Proxy-Require**

  The client may indicate certain SIP extensions in the Proxy-Require fields that the proxy must support in order to successfully handle this request. The proxy must inspect this field and verify that it supports all the extensions listed in the field.

- **Authentication**

  If the SIP Server determines it has to authenticate the originator of the message, it has to make sure the message contains credentials that authenticate the user. If the message does not contain credentials or the credentials failed to authenticate the user, the proxy may return a 407 response containing a challenge. The authentication procedure is explained in more detail in the Authentication chapter.

ADDRESS RESOLUTION

Once a proxy has validated an incoming request and decided to forward it, it must determine the destination(s) to which the message is to be forwarded before sending the messages. The proxy does two types of address resolution:

- Determining the target-set—the proxy resolves the request SIP destination address (Request URI) to a set of SIP addresses that are mapped to it in some way.
- DNS resolution—the proxy resolves each of the SIP destination addresses to a transport address of the form:

  {transport_protcol, IP address, port}

Note A SIP request may be forwarded to more than one destination address. An example of such a case is a user that is simultaneously registered at several locations. In this case, the proxy may decide to fork the request either sequentially or in parallel.

DETERMINING THE TARGET-SET

The first process in address resolution, known as obtaining a target-set in the SIP specification, results in producing a set of SIP addresses. Essentially this stage maps from SIP address to SIP addresses. (Some addresses in the target-set may express explicit transport addresses by using the format sip:user@ip_address;transport=xyz.)

A target-set is obtained in one of two ways:

- **Predefined target-set**

  This is the simpler case, where the destination address of the request (Request-URI) is such that the proxy must automatically forward to the destination address without trying to resolve to other addresses. One such case is where the request-URI is in a domain for which the SIP Server is not responsible. For example, a proxy sip:proxy1.acme.com which is responsible for the domain acme.com receiving a request for sip:bob@example.com must proxy the request to sip:bob@example.com.

- **Target-set determined by proxy**

  If the target-set is not dictated by the message, the proxy may employ whatever mechanism it may wish to determine the target-set. Some options are:

  - Accessing a location service updated by a SIP registrar
  - Reading from a database

- Consulting a presence server
- Using other protocols
- Performing algorithmic substitutions on the destination address

While the Request-URI is an important factor in determining the target set, the proxy may also choose to route based on other message fields, or on external parameters, such as time of day, network and server load.

DNS Resolution

Before forwarding a message, the proxy must resolve the message to concrete transport addresses which it can use in sending the message. Proxies, as other SIP entities, use the DNS mechanism described in RFC 3263 (SIP: Locating SIP Servers). Essentially, this is an algorithm that selectively uses SRV, NAPTR, A and AAAA DNS queries to map a given SIP address to a prioritized set of transport addresses of the form:

{transport_protocol, IP_address, port}

Using this advanced DNS scheme is recommended, since it allows building highly available, load-balanced SIP networks with the possibility of dynamic adjustment through DNS tables.

Stateful Message Forwarding

The message flows below illustrate the way a stateful SIP proxy forwards different types of messages. The term “downstream” means in the direction of the server (request direction); “upstream” means in the direction of the client (response direction). For the sake of simplicity, the scenarios show only non-forking proxies that Record-Route.

NON-INVITE REQUEST

The scenario in Figure 1-4 shows Non-INVITE Request-Response message flow through multiple proxies.

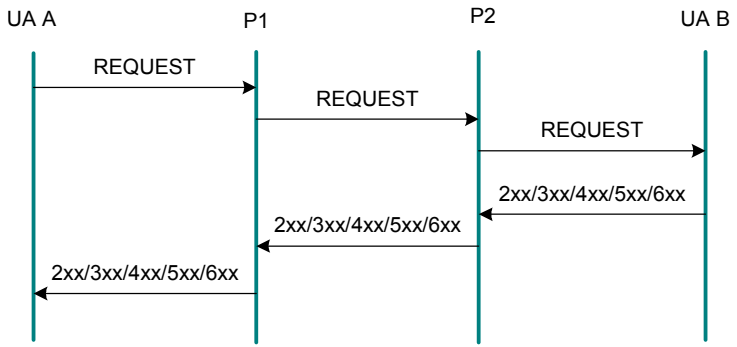

Figure 1-4 *Non-INVITE Request-Response Message Flow*

In the case of non-INVITE requests, such as BYE and REGISTER, proxy functionality is to forward requests and responses as they arrive. The proxy processes all final responses (2xx-6xx) the same way. Retransmitted requests are not forwarded by the proxy, but the proxy may retransmit requests based on its own retransmission timers. Retransmitted responses are forwarded by the proxy.

INVITE-ACK

When processing an INVITE request, a proxy typically responds with a 100 (Trying) response to stop INVITE retransmissions at the previous hop. All received 1xx (provisional) responses except 100 are forwarded to the previous hop. If the proxy does not receive a 100, it may retransmit the INVITE request as necessary.

The scenario in Figure 1-5 shows INVITE-non-2xx-ACK message flow through multiple proxies.

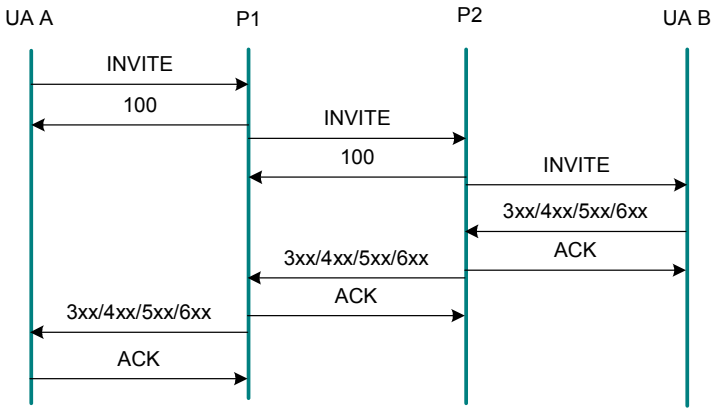

Figure 1-5 *INVITE-non-2xx-ACK Message Flow*

If a non-2xx response is received, the proxy generates the ACK request and forwards the response upstream. Upon reception of a retransmission of the response, the proxy will retransmit the ACK request.

INVITE-200-ACK

The scenario in Figure 1-6 shows INVITE-200-ACK message flow through multiple proxies.

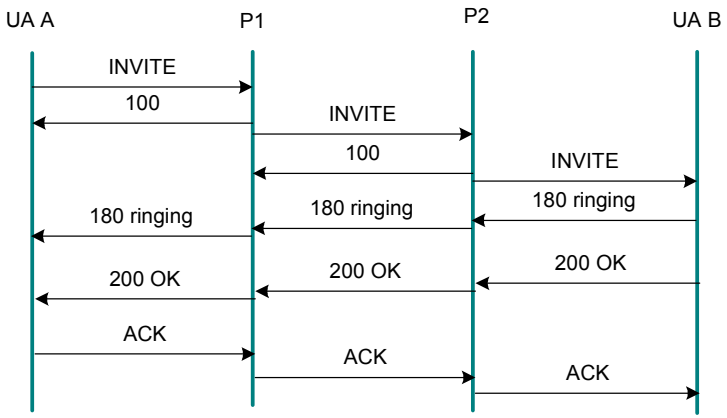

Figure 1-6 *INVITE-200-ACK Message flow*

A 2xx response to an INVITE request presents a special case. For purposes of call-setup robustness, it was decided that reliability (retransmissions) of 200 and ACK messages is to be handled end-to-end, rather than hop-by-hop. This means that upon receiving a 2xx response for an INVITE, the proxy forwards the message (and possibly any retransmission that follows) in a stateless fashion. It does not change state in any of its transactions and it does not generate an ACK request. Only the calling User Agent (UA A in the figure above) is allowed to ACK a 2xx response. The proxies forward the ACK (and possible retransmission) also in a stateless fashion. If either the 2xx or the ACK messages are lost along the way, it is the responsibility of the callee (UA B in the drawing) to retransmit the 2xx until it receives an ACK. This procedure assures that a call is established (and media can start flowing) only when both UAs have completed the handshake.

CANCEL

The scenario in Figure 1-7 shows CANCEL processing.

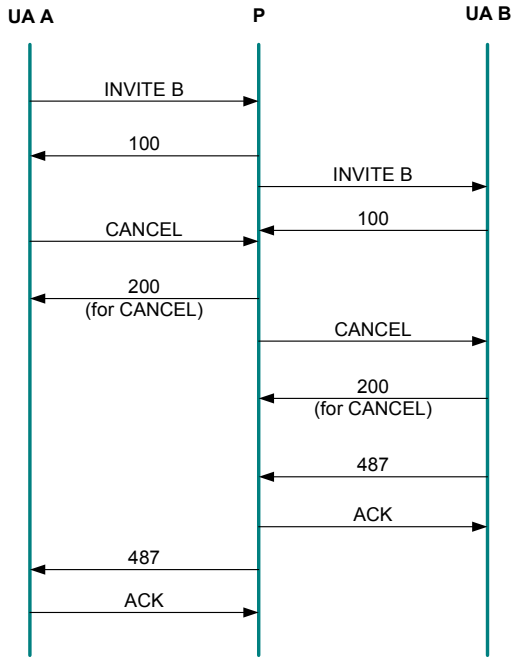

Figure 1-7 *CANCEL Processing*

A stateful proxy may choose to generate a CANCEL request for any pending INVITE request it has previously forwarded (subject to the rules of CANCEL as defined by the SIP standard). This capability is put to use in parallel forking as explained later in Parallel Forking on page 21.

A proxy receiving a CANCEL request must try and match it to an existing INVITE context (*ProxyCoreObj*) and cancel any pending client transactions associated with this INVITE, as illustrated in Figure 1-7 above. If an INVITE context is not found, the proxy must statelessly forward the CANCEL request (it may have statelessly forwarded the associated INVITE previously).

Record-Routing

The scenario in Figure 1-8 shows message proxying without Record-Routing.

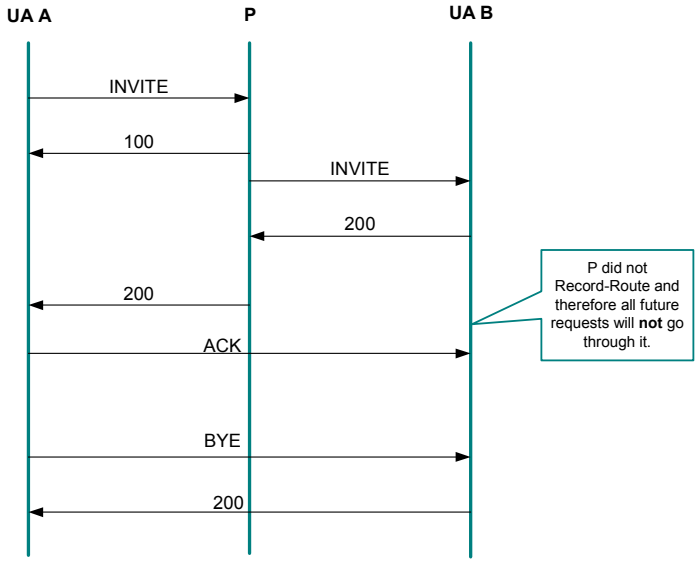

Figure 1-8 *Message Proxying without Record-Routing*

Record-Routing is a SIP mechanism that allows SIP proxies to request being in the signaling path of all future requests that belong to this dialog. A proxy Record-Routes by entering the Record-Route header into the initial dialog-establishing request (currently, INVITE and SUBSCRIBE). The UAS and UAC build route-lists based on the Record-Route headers they find in the request and send each subsequent request with the route list as a set of Route headers.

A proxy that does not Record-Route an INVITE message should not expect to receive any of the following requests sent as part of the *Stack Call-leg*, including the ACK request (as shown in Figure 1-8). Similarly, a proxy that does not Record-Route a SUBSCRIBE request should not expect to see any of the NOTIFY requests sent in the context of the subscription. Exceptions to these rules may arise from the use of outbound proxy or from loose-routing. For more information, see Loose-Routing on page 19 and Outbound Proxy on page 25.

Proxies should normally Record-Route only requests that set up a dialog (currently INVITE and SUBSCRIBE). However, a proxy may add a Record-Route header to any SIP request if it so wishes. SIP UAs do not change their route-lists based on Record-Route headers in requests other than the initial INVITE or SUBSCRIBE. It is recommended that proxies avoid adding Record-Route headers to every request for reasons of processing time and message size.

Selective Record-Routing is highly important because it allows proxies to keep track of some dialogs for their entire duration, while assisting others only in their initial setup phase and then removing themselves from loop. This helps proxies avoid spending resources on routing requests for dialogs that are of no interest to them.

The Record-Routing mechanism was enhanced and fine-tuned numerous times during the evolution of SIP in order to cope with various difficult cases. Today a Record-Routing proxy is required to implement the following functionality:

- Route information pre-processing
- Route information post-processing
- Rewriting Record-Route headers in responses (For example, the case of a proxy that is routing between different domain names.)
- Symmetric Record-Route
- Loose-Routing

Symmetric Record-Route

In some cases, a request is received in the Proxy on network interface X and needs to be sent on network interface Y. In such cases, it is recommended that the proxy adds two Record-Route headers, one for each interface. As result of such behavior of the Proxy, each of the UAs builds the same Route-List (in the opposite order), and each of the UAs will send further requests of the session to the corresponding interface of the Proxy. This also means that the Proxy does not need to analyze the Record-Route in the response.

LOOSE-ROUTING

A Loose-Routing proxy (also known as Loose-Router) is one that follows the procedures defined for Record-Routing in RFC 2543bis-08 and later. These procedures allow a proxy to introduce more hops into the route-list regardless of the final destination of the message. This enables a proxy to route a message through a pre-defined set of other proxies before reaching its final destination. Loose-Routing was added to the SIP specification due to requirements set by wireless 3G standard bodies, such as the 3GPP, in order to allow for "intelligent" routing of messages between visited and home networks.

RECURSION ON 3XX RESPONSES

The scenario in Figure 1-9 shows recursion on 3xx Response

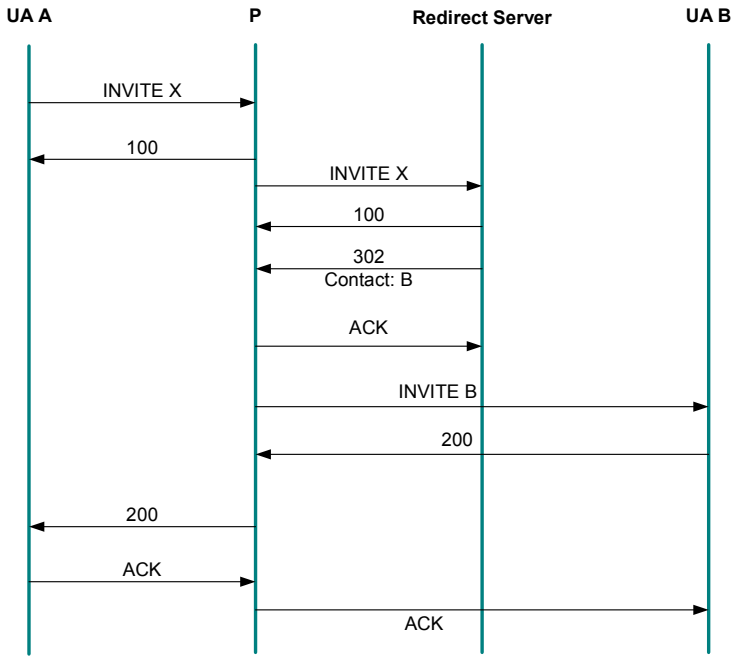

Figure 1-9 *Recursion on 3xx Response*

A proxy, upon receiving a 3xx (Redirection) response, may choose to add the contact address(es) provided in the 3xx response to the target set and possibly generate a new copy of the request to this address(es), as illustrated in Figure 1-9. This process is called recursion on 3xx responses. A proxy may do recursion on 3xx responses only if the Request-URI of the original request indicates an address for which the proxy is responsible.

FORKING

After processing an incoming request and building a target-set for it, the proxy may choose to forward the request to multiple addresses. This process is called forking and a proxy that supports it is called a forking proxy. Forking allows the implementation of features such as simultaneously searching (ringing) for a user in multiple UA devices (desktop phone and cellular phone, for example), finding the first available agent in a call-center, and Forward-on-Busy.

A proxy may choose to fork in several ways:

- Parallel forking—the proxy forwards copies of the request to multiple destinations simultaneously.
- Sequential forking—the proxy forwards the request to one target address at a time, waiting for a final response (failure) before moving on to the next address.
- Mixed forking—the proxy may choose to forward requests to some target addresses in parallel while doing sequential forwarding for others.

PARALLEL FORKING

The scenario in Figure 1-10 (below) shows parallel forking.

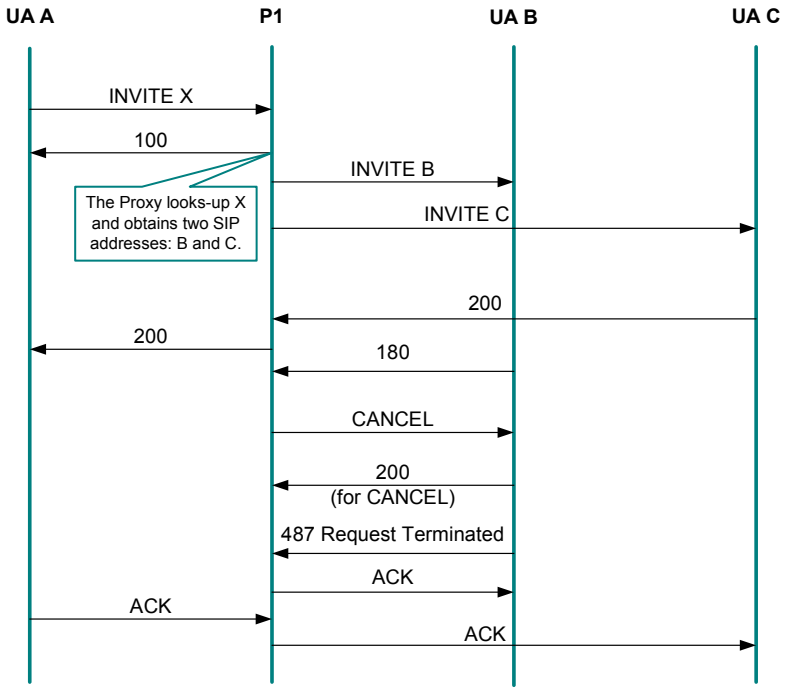

Figure 1-10 *Parallel Forking*

Parallel forking, as illustrated in Figure 1-10, is a more time-effective way to search for a user. This is because parallel forking attempts to reach the user in multiple locations simultaneously. However, parallel forking introduces additional complexity into the work of the proxy. A parallel forking proxy has to handle multiple concurrent client transactions and possibly collect multiple final responses from them. The proxy has to choose the "best" final response and forward it upstream. Picking the best response is done according to an algorithm provided in the SIP standard. Some responses have higher preference then others, and some responses should never be forwarded upstream. For example, a 200 (OK) response is "better" than a 404 (Not Found) response. Upon receiving a 2xx (Success) or 6xx (Global Failure) response at one of the client transactions, the proxy has to cancel the remainder of the pending requests (using CANCEL) and forward the final response upstream.

When parallel forking, a proxy also may have to do aggregation of challenges if it receives multiple 401 (Unauthorized) or 407 (Proxy Authentication Required) responses. In this case, the proxy must collect the challenges from all responses and forward them upstream.

A parallel forking proxy may have to do redirection response aggregation if multiple 3xx responses are received.

SEQUENTIAL FORKING

The scenario in Figure 1-11 shows sequential forking.

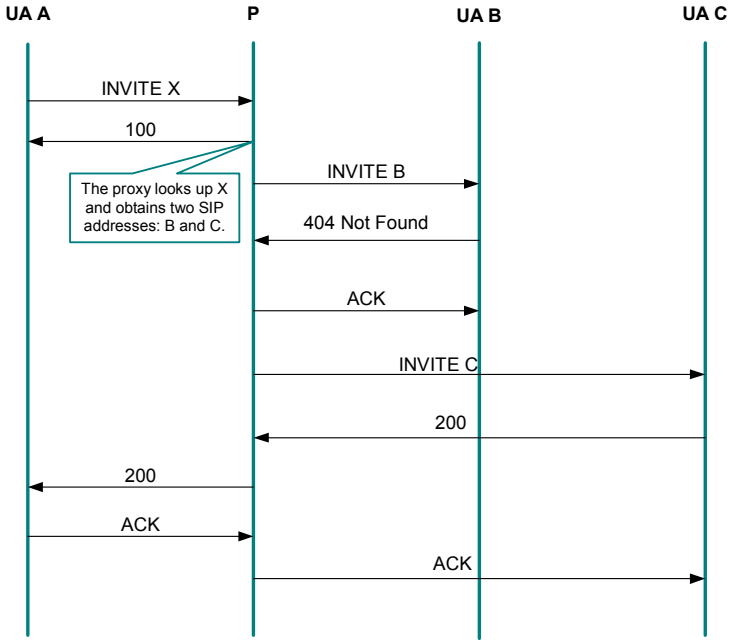

Figure 1-11 *Sequential Forking*

AUTHENTICATION

The scenario in Figure 1-12 shows authentication.

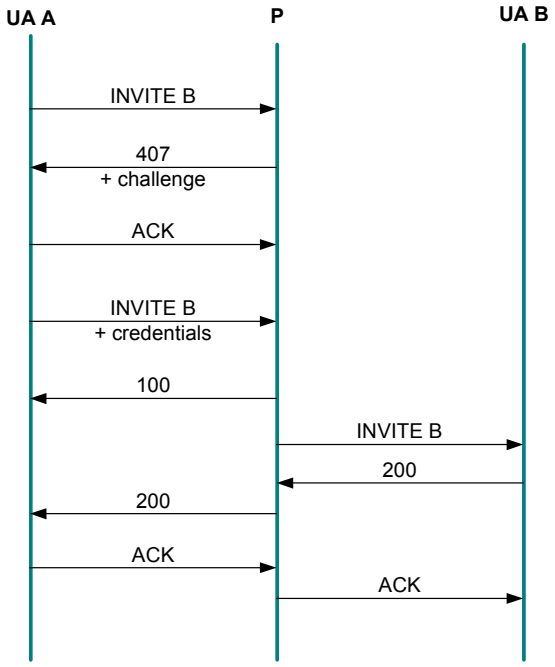

Figure 1-12 *Authentication*

When a UAC sends a request to a proxy server, the proxy server may decide to authenticate the originator before the request is processed. The proxy can challenge the originator to by returning a 407 response (Proxy Authentication Required) with a Proxy-Authenticate header containing the challenge. The client can re-send the request with a Proxy-Authorization header which provides the credentials that match the challenge. A client may provide the credentials also before being challenged in order to avoid the delay and extra processing of the 407 response (the credentials may be built according to cached challenges). Both challenge and credentials are built using a cryptographic hash so that certain values, such as password, are not sent in the clear.

Authentication by proxy is useful for the following:

- Verifying that the originator of the request is indeed an authorized user entitled to receive services (avoiding service theft)

- Asserting that certain message fields were not altered by a third party

SIP authentication also allows for multi-level authentication by different proxies along the signaling path.

Loop Detection and Max Forwards

Figure 1-13 shows an example of an illegal loop.

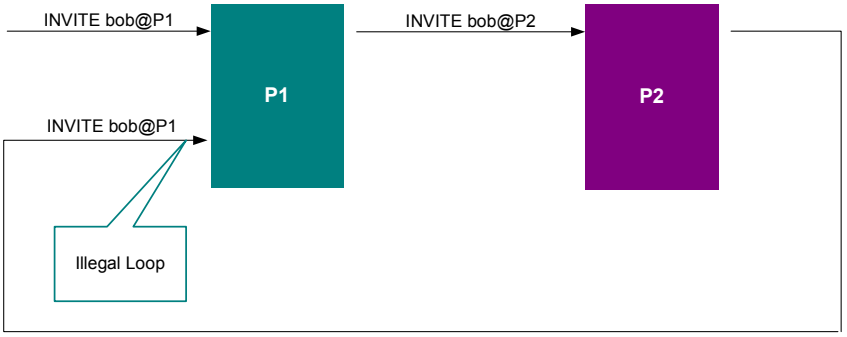

Figure 1-13 *Illegal Loop*

A loop is a situation where a request that arrives at a proxy is forwarded, and later arrives back at the same proxy. An illegal loop occurs if the second time it arrives, the message has the same values in fields that affect the routing decision (Request-URI and any other fields the proxy may take into account). In this case the proxy will forward the request the same way it did the first time and every other proxy along the loop path may do the same. This causes an error condition where the message is looped through a set of proxies for an undefined number of times, consuming network and proxy resources, but never reaching its final destination.

The SIP specification handles loops in two ways:

- Max Forwards (mandatory)
- Loop detection (optional)

Max Forwards

The Max-Forwards header field serves to limit the number of hops a request can transit on the way to its destination. It consists of an integer that is decremented by one at each hop. If the Max-Forwards value reaches 0 before the request reaches its destination, it will be rejected with a 483 (Too Many Hops) error response. The default initial value for Max-Forwards is 70.

This solution requires minimal processing by proxies and UACs, but has the disadvantage of stopping the loop only after the message has been forwarded enough times to exhaust the Max-Forwards value (70 hops by default).

Loop Detection

A proxy can optionally check for loops by employing a special loop detection algorithm. The algorithm affects the way the proxy builds the Via-branch field and mandates the proxy to do certain validations of the Via list in incoming requests. Loop detection requires some extra processing per message, but guarantees immediate detection of the loop as the proxy receives the message for the second time.

Message Spiraling

Figure 1-14 shows an example of message spiraling (Legal Loops).

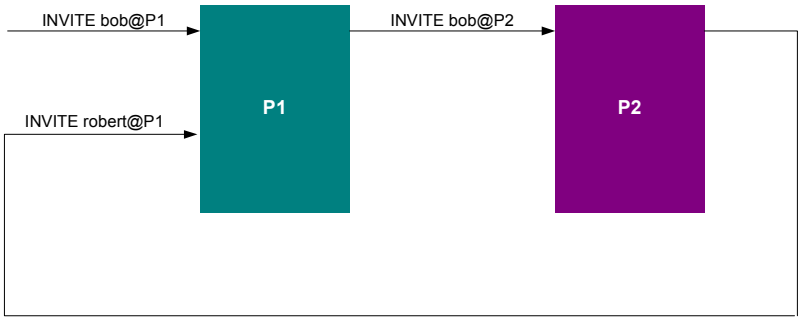

Figure 1-14 *Spiral Example*

A spiral is a SIP request that is routed to a proxy, forwarded onwards, and arrives once again at that proxy, but with a different set of values in the fields that affect the routing decision. The proxy will route the spiraled request to different target addresses the first and second time it processes it, therefore an endless loop is not created. A spiral is not an error condition, unlike a loop. Identifying spirals and telling them apart from loops presents a challenge for loop-detecting proxies and requires the use of special techniques.

Outbound Proxy

Outbound proxy is a proxy that receives requests from a client regardless of the destination of the messages (Request-URI) the client is sending. Simple clients may choose to send every outbound message via an outbound proxy. Typically, a UA is manually configured with an outbound proxy, or can learn about one through auto-configuration protocols.

Outbound proxies are important because they allow the creation of simple UAs that do not have to be concerned with making routing decisions and DNS queries. This is especially important in wireless devices which typically have limited capabilities and resources but may roam to foreign networks.

A proxy that is suitable for use as an outbound proxy is one that is willing to accept requests even though they are not intended for its domain and that do not belong to a dialog that it has Record-Routed.

POLICY

As explained above, SIP Servers have a wide array of possible ways to handle incoming messages. A SIP Server has the freedom to make the following decisions:

- Where to route the request and based on which location information source (location service/database/presence info/ other)
- Proxy/Redirect/Reject
- Stateful/Stateless forwarding
- Record-routing
- Forking—parallel/sequential/none
- Authentication
- Loop detection

These decisions can be made based on many factors, such as:

- Message destination address (Request-URI)
- Domain(s) managed by this server
- Type of request (method)
- Other message fields: From, To, Date, Priority …
- External parameters, such as time of day

In this document, the collective set of rules that govern proxy routing decision making is called Server Policy. Server Policies are typically set by the service providers or administrators that install and configure the SIP Servers.

Because of the large number of input variables and output decisions that are available for Server Policies, SIP Servers are highly versatile and flexible elements. This is of key importance for deployment of SIP-based networks. Server Policy can be as complex or as simple as required by network topology, user profiles, traffic load, security risk levels, and other factors specific to the environment in which the server operates.

Below are examples of some simple policies that may be employed by proxies:

- Forward all requests targeted to domain acme.com based on location service records. Redirect all other requests to sip:routing_proxy.sp.com
- Authenticate all incoming INVITE messages except those intended for 911 emergency services
- Statefully forward (and fork if necessary) requests with destination address sip:bob_smith@acme.com, unless they are from sip:alice_smith@worldcom.com, in which case forward to sip:bob_smith@voicemail.acme.com.
- Switch from stateful to stateless forwarding modes when system resources are low.

B2BUA

In addition to Proxy, Registrar and Redirect servers, the SIP specification also defines a fourth type of server—Back-2-Back UA (B2BUA). A B2BUA is a logical entity that receives a request, processes it as a User Agent server (UAS) and, in order to determine how the request should be answered, acts as a User Agent client (UAC) and generates requests. A B2BUA must maintain call state and actively participate in sending requests and responses for dialogs in which it is involved. The B2BUA is similar in many ways to a Proxy server, but has tighter control over the dialog and does not have the limitations of the SIP standard on the Proxy (for example, a Proxy cannot disconnect a call or alter the messages). While handling request and response messages, the B2BUA forwards requests and responses at a dialog level from side to side. In some respects, a B2BUA is similar to an H.323 gatekeeper in routed mode.

RFC 3261 does not define special functionality for B2BUA, but rather defines it as a concatenation of a UAC and UAS.

Figure 1-15 illustrates a common dialog creation scenario.

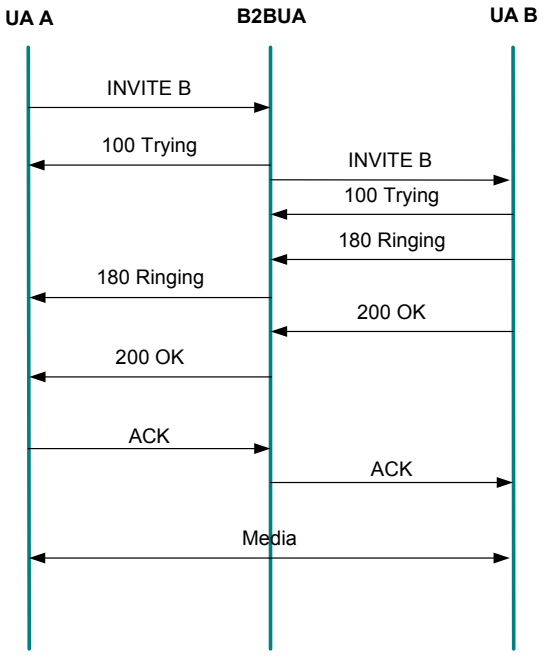

Figure 1-15 *Dialog Creation*

Since a B2BUA acts as a UAS to one side and a UAC to the other side, it has tight control over the call. A B2BUA hides the identity of the initiator of the call from the destination and enables header modification and SDP manipulation (codecs, media IP+Port, an so on). In addition, it is preferable that B2BUA implementation allows the application to initiate actions, such as disconnecting one side of the dialog and even initiating a dialog to one side. Such B2BUA capabilities allow it to act as a Third Party Call Control (3PCC), as defined in *draft-ietf-sipping-3pcc-06.txt*.

Figure 1-16 illustrates a B2BUA that acts as a 3PCC according to Scenario 4 of this draft. This scenario is recommended for cases in which it is not clear that UA A will immediately answer the call, as would happen if this UA is an automatic machine. If UA A is not such a machine, UA B may timeout waiting for the ACK with SDP 2'. The B2BUA cannot send this ACK to UA B before UA A answered with the 200 OK including SDP 2.

This scenario also has some drawbacks, as specified in the draft.

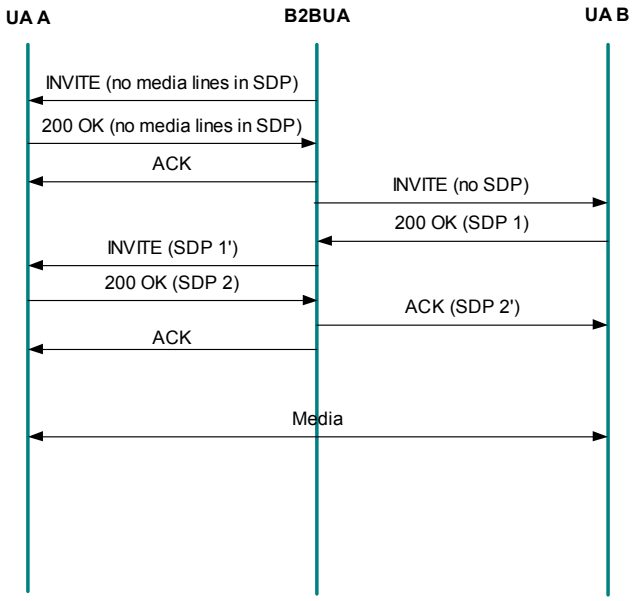

Figure 1-16 *3PCC*

A B2BUA is beneficial to environments with one or more of the following requirements:

- Having the network initiate dialogs or modifying a dialog state.
- Having the network disconnect calls (for example if the caller runs out of pre-paid minutes) or modify calls (for example changing codecs in a call setup process or during a call session).
- Having Third Party Call Control (3PCC) for connecting a call between two UAs by a controller.
- Giving the network tight control over Class 5 features (PBX functionality).
- Having the network change messages in some way that is forbidden in a SIP Proxy (modify/add/remove headers or body, encrypt/decrypt the message or parts of it, compression, and so on).
- Hiding the users or network that is “behind” the SIP Server.

- Keeping close track of a dialog state (for example, for billing purposes).
- systems.

Some of the applications that require B2BUA functionality are:

- 3G Call State Control Functions (CSCF)—the 3GPP has defined 3 types of CSCF proxies that act as a B2BUA in some functions
- Implementation of IP PBX services in the network
- Call Center applications
- Service Creation Platforms
- FW/NAT Application Level GW (ALG) or FW/NAT traversal system in DMZ

Providing such tight control requires call-stateful implementation as in PSTN switches. The consequences are:

- Retains more state (memory per call) therefore has lower capacity and scalability compared to a Proxy server.
- Call establishment/tear down processing is more complex, therefore the maximal performance (CPS) is lower.
- Breaks SIP End-to-End architecture and therefore S/MIME will not work.
- Unlike a Proxy, a B2BUA is a single point of failure, therefore it is highly recommended to add high-availability to such B2BUA is a natural addition to a SIP Server, since it completes the set of standard server functionalities and provides functionality that is much needed by many types of server applications.

EVENTS SERVER

An Events Server is a general implementation of specific event notification, as discussed in RFC 3265, and provides a complete implementation of a Notifier. This RFC defines a general, SIP-extensible framework that allows an entity to subscribe for notifications on the state change of other entities. The Notifier in this architecture is responsible for receiving the SIP SUBSCRIBE request, validating it, and creating a subscription object that also includes the Event package name and the event header ID parameter if one is present. Additionally, the Notifier is responsible for collecting the resource state and sending NOTIFY messages to the Subscribers. The Notifier may involve other entities (some not

in the scope of SIP) to collect resource state information. Validation of a SUBSCRIBE request should be an automatic function that does not require user input, since non-INVITE transactions are limited by Timer F with a duration of 64*T1 (default T1 is 500ms according to RFC 3261). The validation of a SUSCRIBE request may include the following:

Event Package Name

The Notifier may return a 489 (bad event) response if it does not understand the event package that was specified in the Event header value, or is not capable of processing this event package. The Notifier may also create a subscription and return 200 (OK) or 202 (accepted) according to its policy if it cannot immediately determine its ability to process the event package. (For example, when user input is required or other entities need to be involved in such processing).

Subscribe Duration

The Notifier may check that the Expires header value is not too short. If it is too short, it may return a 423 (interval too small) response according to the rules specified in RFC 3265. This response will contain a Min-Expires header to notify the subscriber about the minimum expected duration of the subscription.

Authentication

The Notifier may also choose to perform authentication and authorization according to the application policy. This is an action usually recommended, given the information that is provided in the NOTIFY body.

After the subscription was validated and a subscription object was created, the Notifier must send a NOTIFY message. The message may contain a body expressing the current state of the resources, according to the event package that was processed and according to the application policy. SIP uses this mechanism for the implementation of various functionalities and defines various event packages in related drafts and RFCs. Examples of such event packages are:

- Presence—provides information about the willingness and ability of a user to communicate with other users on the network.
- Winfo—the set of users subscribed to a particular resource for a specific event package, and the state of their subscriptions is called Watcher information (Winfo). A user can subscribe to be notified about the subscription state of such users. This package

is a template package and can be implemented with any event package of the *foo.winfo* template (where "*foo*" can be any SIP event package) including *presence.winfo*, or *winfo.winfo*.

For example, a Presentity that subscribes to its Presence Watcher information will receive notifications about Watchers subscribing to their Presence status and when their subscription status changes.

Using this general framework of an Events Server, servers can be built to handle specific event packages, such as a Presence Server.

PRESENCE SERVER

Presence is a service that allows a party to know the ability and willingness of the other party to participate in a call even before the call attempt is made. A user interested in receiving presence information (a Watcher) for another user (Presentity) can subscribe to his/her presence status and receive Presence status notifications from the Presence system.

The IETF SIPMPLE WG is developing a set of specifications for the implementation of a Presence system using SIP. They are working within a general IETF requirements framework for Presence and IM, which is called Common Presence and Instant Messaging (CPIM).

According to SIMPLE, the Presence system is composed of these components:

- One or more Presence User Agents (PUA)—these are the client devices/software the Presentity uses and that collects Presence data (for example, phone, cell-phone, PDA, soft PC client, and geo-location system).
- One or more Presence Agents (PA)—this is a new type of SIP entity. The PA is responsible for:
  - Receiving and handling Presence subscriptions from Watchers.
  - Receiving and aggregating Presence data from the PUAs (either through SIP or via other means).
  - Composing the Presence status from the fragments of presence data.
  - Notifying all subscribed Watchers when the Presence status of a Presentity in which they are interested changes.
- SIP Proxies—forward Subscription requests and notifications between the Watcher and Presence Agent (basically regular proxies that are not aware of Presence).

- Presence Server—a physical entity that processes SUBSCRIBE requests either as a presence agent (and handles them locally) or as a proxy (and forwards them to other entities)

SIMPLE defines a general type of server called Presence Server (PS) which can act either as a PA or a Proxy, as needed. Essentially this is a Proxy which also has PA capability.

Figure 1-17 illustrates a common Presence implementation, Presence entities and the relation between them in the network.

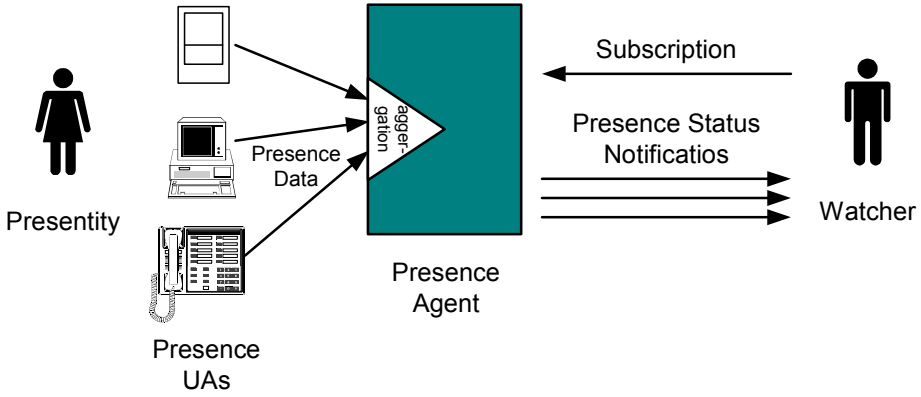

Figure 1-17 *Presence Topography*

Figure 1-18 illustrates a common scenario between Watchers, Presentities and the Presence Agent.

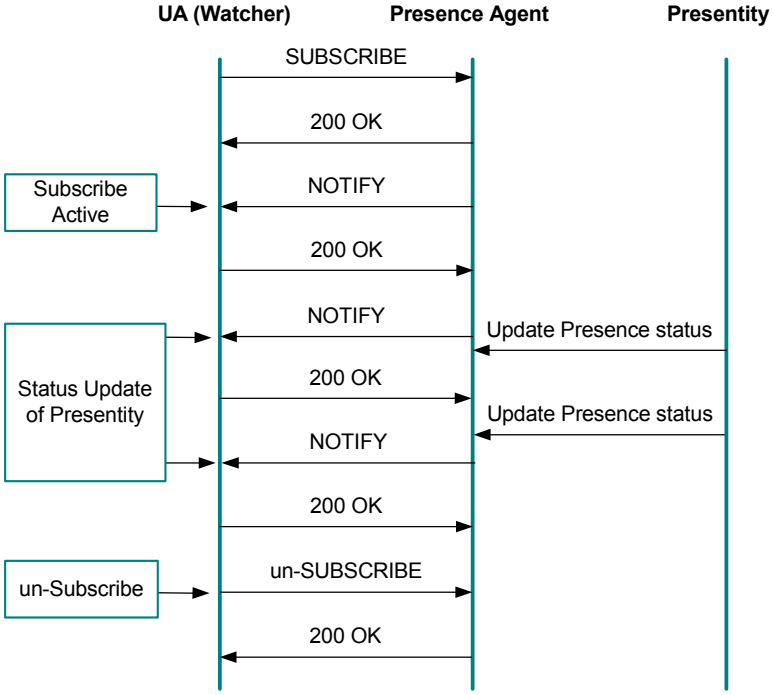

Figure 1-18 *Watchers, Presentities and Presence Agent*

Multiple Watchers can subscribe to receive updates of the status of the Presentity. Each subscription will result in a NOTIFY message generated by the Presence Agent to report an active subscription. The Presence Agent will generate the status information of the Presentity from different sources (REGISTER messages sent by the Presentity, PUBLISH message, or any other source of information not necessarily in the scope of SIP). A change in the status of the Presentity will result in the generation of a NOTIFY message by the Presence Agent to all Watchers according to application policy.

Both a Subscriber (Watcher) and Presence Agent can terminate (un-SUBSCRIBE) a subscription. A Subscriber can send a SUBSCRIBE within the dialog with an Expires header of zero. (An Expires header indicates the duration of the subscription). This causes an immediate termination of the subscription. The Presence Agent then generates a NOTIFY request with the

most recent state. This, in fact, is a way of retrieving Presentity status without actually subscribing to the Presence Agent, and is referred to in the Standard as a Fetcher or Poller.

The Presence Agent can terminate the subscription by sending a NOTIFY request with a Subscription-State header, indicating that the subscription has been terminated.

2

RADVISION SIP SERVER PLATFORM

SIP SERVER DEVELOPMENT CHALLENGES

With the world-wide adoption of SIP, numerous vendors, research organizations and academic institutions are choosing to develop their own SIP servers. While no two SIP server projects are exactly alike, the set of requirements put forward by customers are often the same. RADVISION has identified the following as key technical requirements for a SIP server project:

- Standards compliance
- Interoperability
- Robustness
- Extensibility
- Flexibility and customization
- High performance and capacity
- Maintainability
- Portability

Often SIP servers replace or supplement telephony switches and private branch exchange (PBX) systems so customer requirements are especially strict. Failing to fulfill some or all of these requirements presents a major risk to the commercial success of the product. For example, shipping a non-standard or non-interoperable SIP server may result in lost accounts or in having to do an overall recall or upgrade.

Naturally the regular project constraints also apply when developing SIP servers, namely:

- Minimizing development time
- Minimizing cost of development

Minimizing Development Time and Cost

The need for a fast and effective development process is clear. Often development time has a direct affect on time-to-market and the ability to meet customer deadlines. Development of a SIP Server application typically involves implementation of these layers:

- SIP Stack layer—has to have extra flexibility and customizability in the way transactions are processed because of the special needs of the proxy and the SIP Server in general.
- SIP Server layer—implements the standard SIP Server functionality.
- Application layer—implements all other aspects of the application (for example, service engine, billing module and database access).

Developing all three layers from scratch requires multiple man-years even for a basic type of SIP Server. Using an existing SIP Stack may save some time, but only if the SIP Stack was already enhanced to support proxy implementation. Testing and verifying standards compliance and interoperability are also resource consuming activities and often require purchasing third-party equipment for lab testing.

The RADVISION SIP Server Platform was proven to radically shorten development time and cost. The SIP Server Platform provides the following functionality with a minimal amount of extra coding:

- Out-of-the-box complete proxy, redirect and registrar servers
- B2BUA
- Events Server (for the implementation of Presence and other SIP events)

For example, developing a full-featured proxy/registrar/redirect server using the SIP Server Platform requires as few as two man-months to complete. The SIP Server Platform allows developers to focus on the differentiating features of the application they are developing and thus reduce both the time and cost of development and improve the quality of the product.

RADVISION SIP Server Platform

The RADVISION SIP Server Platform is a comprehensive framework for the development of any type of SIP Server application. The SIP Server Platform includes an implementation of fully-standard server functionality that is controllable through multi-level APIs. SIP Server developers can easily and seamlessly integrate the SIP Server Platform into their application and customize it according to project needs, thus adding SIP Server capabilities to their implementation quickly and effectively.

The SIP Server Platform was designed to specifically address the requirements listed above. This section explains how.

STANDARDS COMPLIANCE

Standards compliance is a key requirement for any communication element. Without it the chances of interoperating with other vendors' equipment are small. Assuring compliance with the SIP standards is complex for the following reasons:

- **Extensive standardization activity**

  The SIP related working groups (SIP, SIPPING, SIMPLE, XCON and so on) within the IETF are extremely active. Each year they discuss many proposed changes and enhancements to the baseline standard and to its extensions. Researching and keeping up-to-date with this activity is time consuming and requires extra resources.

- **SIP still on the move**

  The SIP standard has evolved and changed numerous times in the last few years and is still not fully stable. A SIP server that complies with the standard today is likely to require updates when future changes in the standard occur. There are two main reasons for this continuous change:

  - SIP is being adapted to new environments such as 3rd-Generation (3G) wireless and cable TV networks
  - "Broken" protocol behavior is being fixed.

    Changes to fulfill these requirements are not always backwards compatible.

- **Complexity**

  The functionally defined in the SIP standard for proxy server, registrar server, and redirect server is extensive in scope and not trivial in implementation. A standard SIP server is anything but simple. Some of this functionality is optional, but the bulk of it is not. Building a SIP Server from scratch (even with a SIP Stack in place) is a task that requires careful design and many man-years in implementation.

RADVISION is a leading VoIP technology vendor and is very active in researching and shaping the protocols. The SIP Server Platform, as all other technology products, is kept closely in-tune with the standard, and product updates are regularly shipped with SIP standard updates.

INTEROPERABILITY

Interoperability is a top requirement, especially interoperability with market leaders equipment. However being standards-compliant alone does not guarantee interoperability, since different vendors may interpret the standard differently. Interoperability is mostly achieved through:

- **Implementation philosophy**—leniency when processing incoming messages and strictness when encoding outgoing messages.
- **Extensive interoperability testing**—the most important SIP interoperabilty venue is the SIP Interoperability Testing (SIPit) and SIMPLEt events that take place a number of times each year. In-lab interoperabilty testing is also crucial.

The RADVISION SIP Server Platform is tested on a regular basis in SIPit and SIMPLEt events where interoperability is verified against dozens of vendors at a time, including market leaders. In addition the SIP Server is tested for interoperability in the RADVISION QA lab with equipment bought from select endpoint and server vendors.

ROBUSTNESS

SIP servers are often required to display the same high-level of robustness and fault tolerance required from high-end telecom and datacom equipment. SIP servers are expected to run continuously for long periods of time with no restart. They are expected not to crash even under extreme loads or error conditions. Robustness is typically achieved through correct design and exhaustive testing. A SIP server typically needs to undergo different cycles of testing such as load, stress, and error-condition testing in order to minimize the chances of crashes or faults at the customer site.

The RADVISION SIP Server Platform is a robust standard implementation. The robustness of the SIP Server Platform is verified in the following ways:

- **Extensive testing and QA**—Each new version of the SIP Server Platform undergoes extensive testing and QA before being released to the market.
- **Based on the RADVISION SIP Stack**—the SIP Server Platform is built using the award winning and widely used RADVISION SIP Stack. The SIP Stack is sold separately as part of the RADVISION SIP Toolkit package, where it undergoes independent testing. This dual testing (at Stack level and at SIP Server level) helps further assure robustness in the SIP Server Platform.

- **Interoperability testing**—when testing interoperability the robustness of the product is also tested since many crashes occur due to erroneous protocol implementation by other vendors. Also, in the SIPit and SIMPLEt events, the SIP Server Platform typically takes part in advanced server testing which further aids in detecting and fixing bugs.
- **Used by multiple implementors world wide**—the SIP Server Platform is sold to developers world wide, who typically do their own system testing in various environments. This helps to further assure that the SIP Server Platform is fully robust and stable in any condition.

EXTENSIBILITY

SIP is designed for extensibility—the protocol can be extended in many different ways. The IETF already defined multiple extensions to the baseline protocol and others are sure to come. The implications on SIP applications are that any type of SIP element, endpoint or server, needs to be designed with extensibility in mind. The direct implications on proxy and redirect servers are:

- The servers must be able to process unknown types of messages even without understanding the request type (method), response code, certain headers in the message, or the body of the message.
- The servers must be able to process and respond to the Proxy-Require header.
- Server design and implementation should allow for future extensions without requiring major code re-writes. This typically requires a modular multi-layer design and implementation.

The SIP Server Platform was conceived and implemented as a development tool for any type of SIP server. As a result, it can be extended in many ways including message processing, message structure, state changes, policy and authentication. Nearly every aspect of the behavior of the server is overridable and customizable by the application to allow for maximum extensibility.

FLEXIBILITY AND CUSTOMIZATION

SIP servers may be deployed in different environments which have varying constraints. The ability to write a server that can be tailored and configured to match the constraints of a network without code changes is a key advantage. In many cases network administrators will want to do some of the customization and fine-tuning on their own at the customer site.

The SIP Server Platform accommodates this requirement through its extensive multi-level API, its rich set of configuration parameters, and its open and modular design. Many aspects of server behavior are customizable, such as domain names, DNS usage, location database access, memory consumption, and automatic behavior when processing messages. The SIP Server Platform enables the application to decide on any aspect of message processing, such as routing decisions, forking, and stateless/stateful authentication. The SIP Server Platform features multiple levels of APIs and callbacks that enable the application developer to have fine control even over the most low-level behavior, such as the order of headers in an outgoing message.

PERFORMANCE AND CAPACITY

High performance and capacity are prime requirements for carrier-class and large enterprise types of equipment.

SIP server performance is typically measured in calls per second (CPS), transactions per second (TPS) or messages per second (MPS). Capacity is typically measured in maximum number of concurrent transactions (sometimes called “logical ports”).

The SIP Server Platform is optimized for both types of requirements. It yields extremely high CPS, TPS and MPS rates (benchmarking documents are available). This is due to the high performance capabilities of the underlying Stack as well as to the use of highly efficient ANSI C code in the message and transaction processing engines inside the SIP Server Platform.

The SIP Server Platform achieves high capacity through tight control over resource usage, especially memory consumption. The memory consumed by a transaction in progress is minimal, thus the number of concurrent transactions that a machine can sustain is high and can be further up scaled by installing more memory.

MAINTAINABILITY

Maintenance is an important (and often costly) phase in the life cycle of any SIP server application. This is typically due to:

- Standard changes, which frequently still occur in SIP.
- Customer requirement changes—most SIP networks today are still “1st generation” and therefore the customers often do not have full advanced knowledge of their requirements.
- Defects and interoperability issues detected in the product.

Using the SIP Server Platform can improve the overall maintainability of the project and can reduce the time and resources spent on maintenance for the following reasons:

- RADVISION does its own maintenance of the SIP Server Platform including alignment with the latest standards and bug fixes. These versions are made available to customers under the maintenance agreement and free them from the need to update their code with SIP standard changes.
- The SIP Server Platform is provided entirely in source code for enhanced readability and has extensive and customizable logging capabilities for error tracking and debugging. This makes detecting and fixing bugs related to use of the SIP Server Platform easier and faster.
- The SIP Server Platform introduces a layered and object oriented design that improves modularity and maintainability.

Portability

In some cases, the platform on which the SIP server application runs may change over time. Being able to port your code to a new operating system or platform in a fast and straight-forward way is an advantage.

The RADVISION SIP Server Platform is built in a cross-platform way so that all its code is entirely operating system-agnostic except for an internal operating system abstraction layer. The SIP Server Platform was already ported to multiple operating systems (including embedded/RTOS) and is expected to be ported to others in the future. The API that the SIP Server Platform exposes to the application is the same regardless of the underlying operating system. Therefore, migrating to a new operating system does not necessitate any code changes.

SIP Server Platform Capabilities

The following sections present the capabilities of the RADVISION SIP Server Platform.

Standards compliance

The SIP Server Platform complies with the following IETF specifications:

- RFC 3261—SIP: Session Initiation Protocol (baseline SIP spec)
- RFC 3263—locating SIP Servers
- Draft-ietf-simple-presence-10
- Draft-ietf-simple-winfo-package-05
- Draft-ietf-simple-winfo-format-04
- Draft-ietf-impp-cpim-pidf-05
- Other specifications—to be announced

General Capabilities

The general capabilities of the SIP Server Platform are as follows:

- TLS—a security protocol that is typically layered on top of connection-oriented transports, such as TCP. TLS allows client/server applications to communicate over TCP in a way that is designed to prevent eavesdropping, tampering, or message forgery.
- Persistent Connection—in many cases, a SIP Server can handle multiple messages, transactions, and dialogs with a single SIP Server or Gateway and therefore, a single TCP connection can be reused avoiding TCP handshake overhead (and even more so in TLS connections).
- Multi-homed Host—the SIP Server Platform can be configured to listen to multiple IP addresses and to send a message via a specific socket. This feature can be used to enable the support of heterogeneous networks, such as combinations of IPv4 and IPv6 and different network segments.
- Symmetric Record-Route—In some cases, a request is received in the Proxy on network interface X and needs to be sent on network interface Y. In such cases, it is recommended that the proxy adds two Record-Route headers, one for each interface. As result of such behavior of the Proxy, each of the UAs builds the same Route-List (in the opposite order), and each of the UAs will send further requests of the session to the corresponding interface of the Proxy. This also means that the Proxy does not need to analyze the Record-Route in the response.
- Internal Multithreading (Configurable)
- DNS—The SIP Server Platform supports both basic (A records) and advanced (SRV and NAPTR records) DNS queries as defined in RFC 3263 (Locating SIP Servers).
- IPv6 support—the SIP Server Platform can listen on IPv6 sockets, accept TCP connections, and send and receive IPv6 packets (UDP and TCP).[1]

1. This capability is not recommended for operating systems that do not have native support for IPv6. Also in some cases, using IPv6 may prohibit full use of advanced DNS.

- Authentication—the SIP Server Platform support digest-MD5 (HTTP) authentication as specified in the standard. This can be extended to support other kinds of authentication.
- Address resolution—the SIP Server Platform supports the full procedure for address resolution including:
  - Determining a target-set, both:
    - Predefined
    - Defined by proxy based on location service or any other means
  - DNS resolution

PROXY CAPABILITIES

The SIP Server Platform proxy features complete SIP proxy functionality with the following capabilities:

- Stateful forwarding
- Stateless forwarding
- Forking—parallel/sequential/mixed
- Record-Routing
- Loose-Routing
- CANCEL processing and forwarding
- Recursion on 3xx responses
- Loop detection
- Max Forwards check
- Working as outbound proxy
- Message validation
- Authentication

REGISTRAR CAPABILITIES

The SIP Server Platform supports all standard registrar functionality:

- Accept and Validate REGISTER messages
- Read location mappings from location service
- Apply registration logic
- Update location service
- Remove location mappings that have expired from the location service
- Authentication

Redirect Capabilities

The SIP Server Platform provides full redirect server functionality:

- Receiving and processing incoming responses
- Address resolution
- Returning 3xx response with one or more Contact addresses
- Authentication

Back-to-Back User Agent

The Back-to-Back User Agent (B2BUA) is an optional add-on module that is sold separately. The B2BUA has tighter control over the dialog. The module can function as one of the following:

- Transparent B2BUA for implementation of FW/NAT traversal and anonymous applications
- Third Party Call Control (3PCC) for development of PBX and Class 5 switch applications, including a complete state machine implementation allowing easy development of 3PCC and transfer functionalities

The transparent B2BUA provides seamless functionality for:

- Connecting and disconnecting dialogs
- Handling CANCEL requests
- Handling BYE requests
- Handling re-INVITE requests
- Handling general requests (REFER, SUBSCRIBE, PRACK, and so on)
- DNS functionality
- Store/Restore of B2B Transparent objects for high availability

Additional functionality is provided when the B2BUA functions also as a 3PCC. The 3PCC enables more control of the application over dialog creation and provides means of dialog state modification. This flexibility enables the implementation of Class 5 features, such as transfer, forward, hold, call park and pick-up for the development of PBX applications. The application can initiate dialogs out of the scope of a specific B2B Transparent object, and after a dialog has been connected, the application can add it into a B2B Transparent object. The following are examples of possible functionality that the application can implement:

- Create and connect dialogs and add them to a B2B Transparent object
- Remove connected dialogs from a B2B Transparent object

- Disconnect one side of a dialog
- Initiate a request/response on a given B2B Transparent object

EVENTS SERVER

The Events Server is an optional add-on module that is sold separately. The Events Server is a general implementation of specific event notification, as discussed in RFC 3265, and provides a complete implementation of a Notifier. This module provides a general SIP-extensible framework exposing APIs to the application that allow handling of any type of SIP event package (including user-defined packages) and building of a general purpose SIP Events Server. In addition to this general purpose framework, the add-on module provides the specific implementation of two SIP event packages, Presence Server and Winfo.

PRESENCE SERVER

The Presence Server module allows a party to know the ability and willingness of other parties to participate in a call even before an attempt has been made. The Presence Server is responsible for handling Presence SUBSCRIBE requests with event package “presence” from Watchers, and enables the application to notify them about the Presence status of the Presentities. The Presence Server works in coordination with the Presence Agent, to receive updated Presentity status according to REGISTER requests monitored by the Presence Agent, and may work similarly with other user applications which can update the Presence Server of Presentity status through its API functions. Such presence status information can be collected from SIP PUBLISH requests or other means out of the scope of SIP. The Presence Server API also enables sending Presence Notification messages with Presence documents for active Presence subscriptions. The Presence documents are sent as the body of the Notification message, and include Presence information about the requested Presentity.

WINFO MODULE

The Winfo module handles subscriptions of the winfo template. The Winfo module can handle an incoming Subscribe request with the winfo template and any event package. For example, the *presence.winfo* and *foo.winfo* Event headers are both with the winfo template, and can be handled by the Winfo module (*presence* and *foo* are the event packages). The Winfo module offers the application a high-level interface for handling incoming Winfo SUBSCRIBE requests and for initiating Notify messages with Winfo documents, by implementing draft-ietf-simple-winfo-package-05. The Winfo module uses the XML Encoder library for creating Winfo documents according to draft-ietf-simple-Winfo-format-04. For more information, see the *XML Encoder* chapter located in the *SIP Events Server Programmer and Reference Guide*.

The Winfo module is implemented above the Events layer. The Winfo API allows the application to decide how to handle a specific Winfo SUBSCRIBE request according to its policy, to control Notify requests sent to the Winfo subscribers, and to create Winfo documents.

NON-SIP CAPABILITIES

The SIP Server Platform includes some non-SIP functionality in the form of Server Components. For more information, see Server Components on page 49.

Currently the following functionality is supported:

- Location service (database)
- User/password database
- Cryptographic algorithms needed for security
- Presence Agent—part of the optional Presence Server add-on module
- Others—to be announced later

In addition, XML Encoder library is included in the SIP Server Platform. For more information, see the XML Encoder on page 50.

APPLICATION LAYOUT

Figure 2-1 illustrates the layout of the a SIP Server application using the SIP Server Platform.

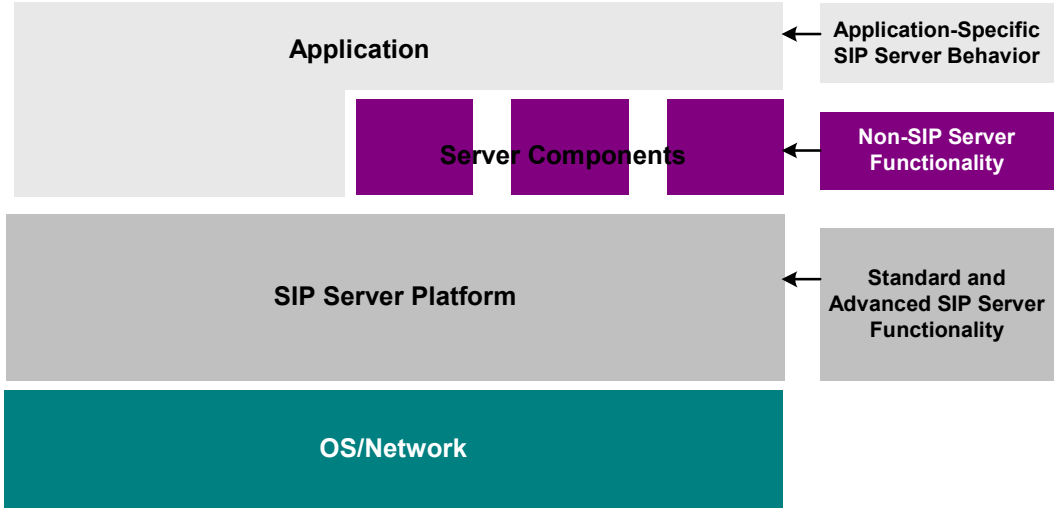

Figure 2-1 *SIP Server Application Layout*

The SIP Server Platform development model breaks the SIP Server application into four parts:

Operating system (OS) and networking layer

This layer is typically provided by a third party.

SIP Server Platform

The SIP Server Platform implements standard SIP Server functionality as defined in the SIP standard.

Server Components

Server components implement functionality that is not SIP-specific, but is required to complete the SIP Server.

Server components are replaceable plug-in modules. They have well-defined interfaces with the SIP Server Platform and with the application. Anyone can develop new implementations to server components as long as these implement the appropriate interfaces.

The SIP Server Platform comes complete with default implementations for server components. These are provided as reference implementations only. Application developers are encouraged to replace or enhance them as they see fit.

Currently the following server components are included:

- Location database—implements the interface to the location service (the storage place of SIP location mappings).

  The SIP server uses this interface to read and to write location mappings as part of the address resolution process (proxy and redirect servers) and the registration process (registrar). Implementations may vary according to the type of location database used. Possible implementations include LDAP client, SQL client, and memory-based local database. The default implementation provided with the SIP Server Platform is of a memory-based, highly efficient database.

- Security—the Security component implements all non-SIP aspects of security, such as cryptographic algorithms and user/password databases.

  The default implementation provided with the SIP Server Platform implements MD5-hash and memory-based user/password database.

- Presence Agent—the Presence Agent monitors REGISTER requests received by the SIP Server, concludes Presentity status, and initiates NOTIFY requests to the relevant Watchers using the Presence Server API. A user can update this default implementation and add other sources from which to conclude Presentity status.
- Other server components will be defined in the future as more capabilities are added to the SIP Server Platform.

XML Encoder

The XML Encoder is a library that can be used for creating XML documents. Many drafts defined by the SIMPLE workgroup are based on XML. This library enables the application to create any XML document that can be set as the body of Notify messages. The Presence Agent and the Winfo modules use this library for creating the relevant documents according to the drafts.

Note The Presence Agent and XML Encoder is part of the optional Events Server add-on module.

Application

The application layer implements all the application-specific behavior. This is where innovative services, value-added functionality and differentiated features are implemented while utilizing the powerful and versatile SIP capabilities of the underlying layers.

The SIP Server Platform together with its default Server Components provide extensive out-of-the-box SIP server functionality. Implementing any of the following servers can be done with minimal coding:

- All-in-one SIP proxy, registrar and redirect server with simple routing policy and co-located location service
- Stateless load-balancer (proxy)
- Stand-alone registrar with local location database
- Redirect Server
- Back-to-Back UA
- Events Server
  - Presence Server
  - Winfo module

- General SIP Events interface

SIP Server Platform Architecture

Figure 2-2 illustrates the SIP Server Platform architecture.

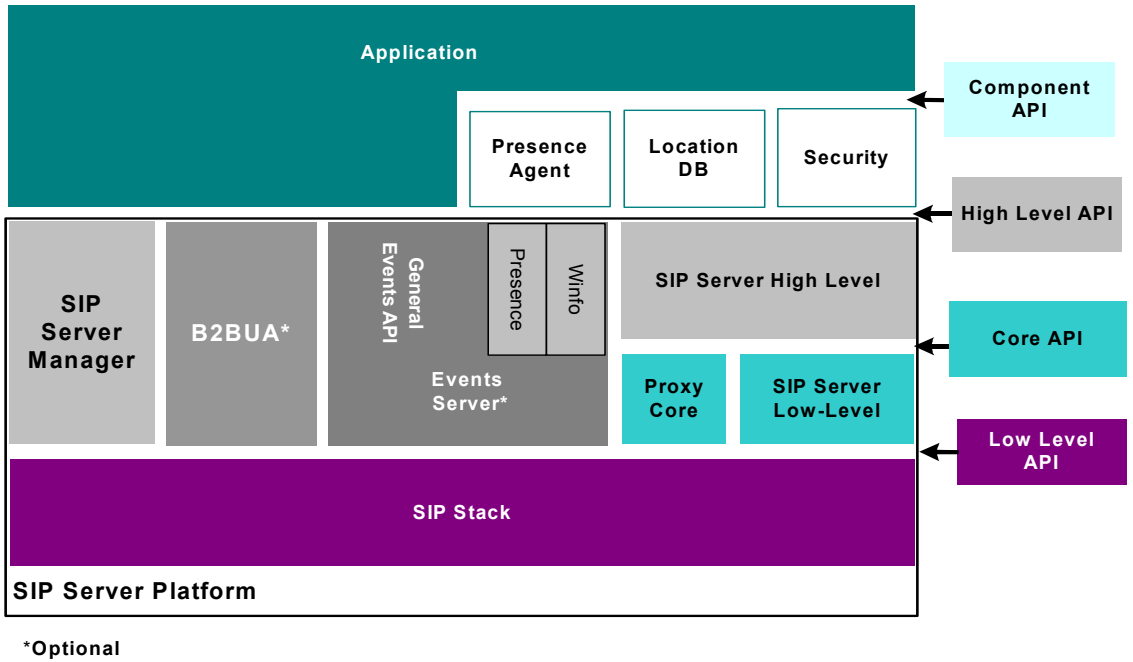

Figure 2-2 *Platform Architecture*

The SIP Server Platform has multiple internal layers. This allows for enhanced modularity and flexibility. Each layer exposes its API functions. Application developers can mix and match the level of API they choose to use according to application needs.

The following API layers are exposed:

- **Component API**—server components also expose a limited API that enables the application to use them. This is a non-SIP API.

- **High Level API**—the High Level API provides higher abstraction on top of the low level modules. This level contains such objects as Registration and allows configuring, initializing and shutting-down the entire SIP Server Platform.
- **Core API**—the Core API exposes the *ProxyCoreObj* (PCO), which is the low level processing object that matches server and client transactions and executes forwarding. The Core API is also used for working with state machines.
- **Low level API**—this API partially exposes the underlying Stack. This is mostly for purposes of working with messages (the messaging layer is located in the underlying Stack).

Note The underlying Stack cannot be used by the application for purposes other than SIP Server development and only as specified in product documentation.

SIP Server Platform Coding

The coding guidelines used by the SIP Server Platform are as follows:

- The SIP Server Platform is provided entirely in source code.
- The SIP Server Platform was designed and coded based on object-oriented (OO) methodology.
- For improved readability and maintainability, the SIP Server Platform was written using strict coding conventions covering naming, internal comments, code structure and project layout. The code is carefully documented.
- The SIP Server Platform is implemented in ANSI C.

SIP Server Platform Package

The SIP Server Platform package includes the following:

- SIP Server Platform libraries in source code
- Server components in source code
- SIP Server add-on modules' libraries and source code (optional)
- Documentation in PDF, HTML and CHM format (For a list of all the SIP Server Platform documentation, see the chapter, *About this Manual*.)
- Fully functional Test Application with graphical user interface given in source code

- Sample code—short, internally-documented, simple and compliable programs that demonstrate the usage of the APIs in source code

Operating System Support

Currently the following operating systems are supported:

- Solaris
- Windows NT, 2000, XP
- Linux Redhat
- VxWorks

The SIP Server Platform will be ported by RADVISION to additional operating systems in the future.

RADVISION Family of SIP Development Solutions

The RADVISION SIP Server Platform is part of a complete family of SIP technology products RADVISION provides for SIP developers.

The other products include:

- RADVISION SIP Toolkit—RADVISION's award winning development tool including SIP, SDP, RTP/RTCP Stacks.
- RADVISION ProLab™ SIP Test Manager—a complete SIP lab testing solution.
- RADVISION Media Device Framework—a framework for the development of IP phones and residential gateways.

RADVISION®
an Avaya company

WP-DI-SIPTECHNICAL-0612-A

About Radvision

Radvision, an Avaya company, is a leading provider of video conferencing and telepresence technologies over IP and wireless networks. We offer end-to-end visual communications that help businesses collaborate more efficiently. Together, Radvision and Avaya are propelling the unified communications evolution forward with unique technologies that harness the power of video, voice, and data over any network. www.radvision.com

USA/Americas
T +1 201 689 6300
F +1 201 689 6301
infoUSA@radvision.com

EMEA
T +44 20 3178 8685
F +44 20 3178 5717
infoUK@radvision.com

APAC
T +852 3472 4388
F +852 2801 4071
infoAPAC@radvision.com