SIP Introduction

Jan Janak

SIP Introduction
by Jan Janak

A brief overview of SIP describing all important aspects of the Session Initiation Protocol.

Table of Contents

List of Figures

Chapter 1. SIP Introduction

1.1. Purpose of SIP

SIP stands for Session Initiation Protocol. It is an application-layer control protocol which has been developed and designed within the IETF (http://www.ietf.org). The protocol has been designed with easy implementation, good scalability, and flexibility in mind.

The specification is available in form of several RFCs, the most important one is RFC3261 (http://www.ietf.org/rfc/rfc3261.txt) which contains the core protocol specification. The protocol is used for creating, modifying, and terminating sessions with one or more participants. By sessions we understand a set of senders and receivers that communicate and the state kept in those senders and receivers during the communication. Examples of a session can include Internet telephone calls, distribution of multimedia, multimedia conferences, distributed computer games, etc.

SIP is not the only protocol that the communicating devices will need. It is not meant to be a general purpose protocol. Purpose of SIP is just to make the communication possible, the communication itself must be achieved by another means (and possibly another protocol). Two protocols that are most often used along with SIP are RTP and SDP. RTP protocol is used to carry the real-time multimedia data (including audio, video, and text), the protocol makes it possible to encode and split the data into packets and transport such packets over the Internet. Another important protocol is SDP, which is used to describe and encode capabilities of session participants. Such a description is then used to negotiate the characteristics of the session so that all the devices can participate (that includes, for example, negotiation of codecs used to encode media so all the participants will be able to decode it, negotiation of transport protocol used and so on).

SIP has been designed in conformance with the Internet model. It is an end-to-end oriented signalling protocol which means, that all the logic is stored in end devices (except routing of SIP messages). State is also stored in end-devices only, there is no single point of failure and networks designed this way scale well. The price that we have to pay for the distributiveness and scalability is higher message overhead, caused by the messages being sent end-to-end.

It is worth of mentioning that the end-to-end concept of SIP is a significant divergence from regular PSTN (Public Switched Telephone Network) where all the state and logic is stored in the network and end devices (telephones) are very primitive. Aim of SIP is to provide the same functionality that the traditional PSTNs have, but the end-to-end design makes SIP networks much more powerful and open to the implementation of new services that can be hardly implemented in the traditional PSTNs.

SIP is based on HTTP protocol. The HTTP protocol inherited format of message headers from RFC822 (http://www.ietf.org/rfc/rfc822.txt). HTTP is probably the most successful and widely used protocol in the Internet. It tries to combine the best of the both. In fact, HTTP can be classified as a signalling protocol too, because user agents use the protocol to tell a HTTP server in which documents they are interested in. SIP is used to carry the description of session parameters, the description is encoded into a document using SDP. Both protocols (HTTP and SIP) have inherited encoding of message headers from RFC822 (http://www.ietf.org/rfc/rfc822.txt). The encoding has proven to be robust and flexible over the years.

1.2. SIP URI

SIP entities are identified using SIP URI (Uniform Resource Identifier). A SIP URI has form of sip:username@domain, for instance, sip:joe@company.com. As we can see, SIP URI consists of username part and domain name part delimited by @ (at) character. SIP URIs are similar to e-mail addresses, it is, for instance, possible to use the same URI for e-mail and SIP communication, such URIs are easy to remember.

1.3. SIP Network Elements

Although in the simplest configuration it is possible to use just two user agents that send SIP messages directly to each other, a typical SIP network will contain more than one type of SIP elements. Basic SIP elements are user agents, proxies, registrars, and redirect servers. We will briefly describe them in this section.

Note that the elements, as presented in this section, are often only logical entities. It is often profitable to co-locate them together, for instance, to increase the speed of processing, but that depends on a particular implementation and configuration.

1.3.1. User Agents

Internet end points that use SIP to find each other and to negotiate a session characteristics are called *user agents*. User agents usually, but not necessarily, reside on a user's computer in form of an application--this is currently the most widely used approach, but user agents can be also cellular phones, PSTN gateways, PDAs, automated IVR systems and so on.

User agents are often reffered to as *User Agent Server* (UAS) and *User Agent Client* (UAC). UAS and UAC are logical entities only, each user agent contains a UAC and UAS. UAC is the part of the user agent that sends requests and receives responses. UAS is the part of the user agent that receives requests and sends responses.

Because a user agent contains both UAC and UAS, we often say that a user agent behaves like a UAC or UAS. For instance, caller's user agent behaves like UAC when it sends an INVITE requests and receives responses to the request. Callee's user agent behaves like a UAS when it receives the INVITE and sends responses.

But this situation changes when the callee decides to send a BYE and terminate the session. In this case the callee's user agent (sending BYE) behaves like UAC and the caller's user agent behaves like UAS.

Figure 1-1. UAC and UAS

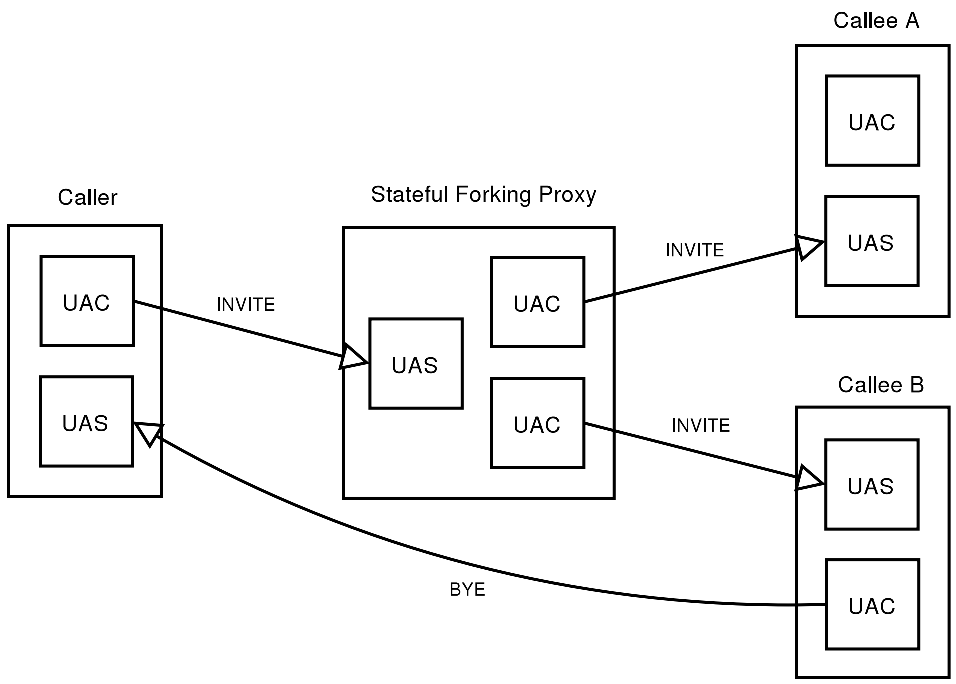


Figure 1-1 shows three user agents and one stateful forking proxy. Each user agent contains UAC and UAS. The part of the proxy that receives the INVITE from the caller in fact acts as a UAS. When forwarding the request statefully the proxy creates two UACs, each of them is responsible for one branch.

In our example callee B picked up and later when he wants to tear down the call it sends a BYE. At this time the user agent that was previously UAS becomes a UAC and vice versa.

1.3.2. Proxy Servers

In addition to that SIP allows creation of an infrastructure of network hosts called *proxy servers*. User agents can send messages to a proxy server. Proxy servers are very important entities in the SIP infrastructure. They perform routing of a session invitations according to invitee's current location, authentication, accounting and many other important functions.

The most important task of a proxy server is to route session invitations "closer" to callee. The session invitation will usually traverse a set of proxies until it finds one which knows the actual location of the callee. Such a proxy will forward the session invitation directly to the callee and the callee will then accept or decline the session invitation.

There are two basic types of SIP proxy servers--stateless and stateful.

1.3.2.1. Stateless Servers

Stateless server are simple message forwarders. They forward messages independently of each other. Although messages are usually arranged into transactions (see Section 1.5), stateless proxies do not take care of transactions.

Stateless proxies are simple, but faster than stateful proxy servers. They can be used as simple load balancers, message translators and routers. One of drawbacks of stateless proxies is that they are unable to absorb retransmissions of messages and perform more advanced routing, for instance, forking or recursive traversal.

1.3.2.2. Stateful Servers

Stateful proxies are more complex. Upon reception of a request, stateful proxies create a state and keep the state until the transaction finishes. Some transactions, especially those created by INVITE, can last quite long (until callee picks up or declines the call). Because stateful proxies must maintain the state for the duration of the transactions, their performance is limited.

The ability to associate SIP messages into transactions gives stateful proxies some interesting features. Stateful proxies can perform forking, that means upon reception of a message two or more messages will be sent out.

Stateful proxies can absorb retransmissions because they know, from the transaction state, if they have already received the same message (stateless proxies cannot do the check because they keep no state).

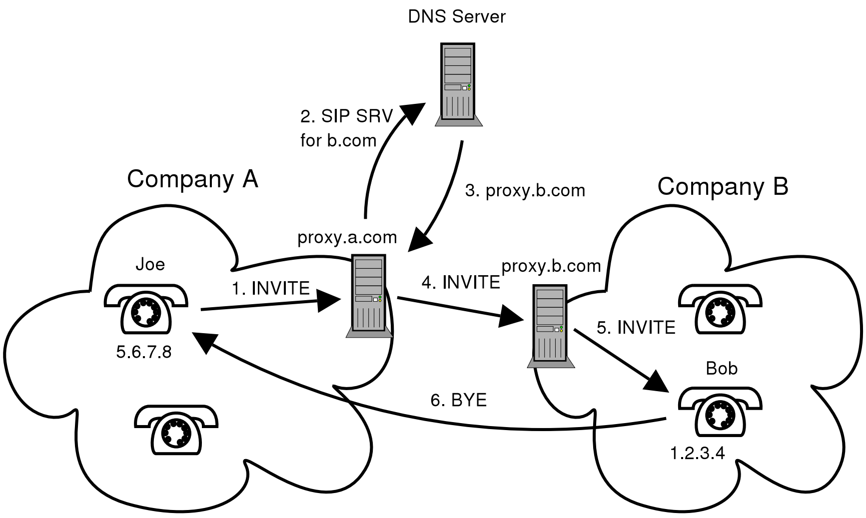
Stateful proxies can perform more complicated methods of finding a user. It is, for instance, possible to try to reach user's office phone and when he doesn't pick up then the call is redirected to his cell phone. Stateless proxies can't do this because they have no way of knowing how the transaction targeted to the office phone finished.

Most SIP proxies today are stateful because their configuration is usually very complex. They often perform accounting, forking, some sort of NAT traversal aid and all those features require a stateful proxy.

1.3.2.3. Proxy Server Usage

A typical configuration is that each centrally administered entity (a company, for instance) has it's own SIP proxy server which is used by all user agents in the entity. Let's suppose that there are two companies A and B and each of them has it's own proxy server. Figure 1-2 shows how a session invitation from employee Joe in company A will reach employee Bob in company B.

Figure 1-2. Session Invitation



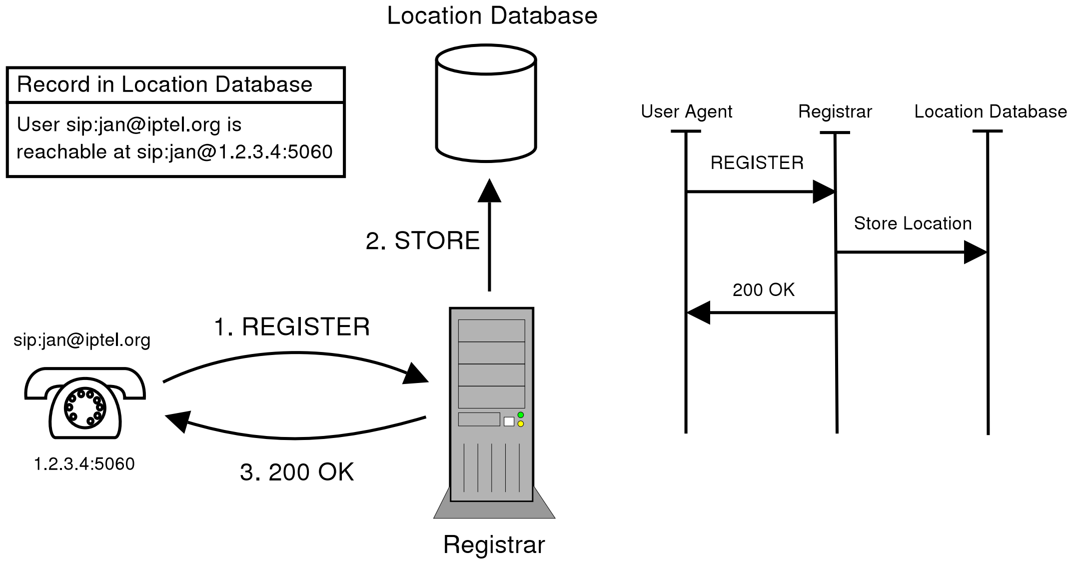
User Joe uses address sip:bob@b.com to call Bob. Joe's user agent doesn't know how to route the invitation itself but it is configured to send all outbound traffic to the company SIP proxy server proxy.a.com. The proxy server figures out that user sip:bob@b.com is in a different company so it will look up B's SIP proxy server and send the invitation there. B's proxy server can be either preconfigured at proxy.a.com or the proxy will use DNS SRV records to find B's proxy server. The invitation reaches proxy.bo.com. The proxy knows that Bob is currently sitting in his office and is reachable through phone on his desk, which has IP address 1.2.3.4, so the proxy will send the invitation there.

1.3.3. Registrar

We mentioned that the SIP proxy at proxy.b.com knows current Bob's location but haven't mentioned yet how a proxy can learn current location of a user. Bob's user agent (SIP phone) must register with a *registrar*. The registrar is a special SIP entity that receives registrations from users, extracts information about their current location (IP address, port and username in this case) and stores the information into location database. Purpose of the location database is to map sip:bob@b.com to something like sip:bob@1.2.3.4:5060. The location database is then used by B's proxy server. When the proxy receives an invitation for sip:bob@b.com it will search the location database. It finds sip:bob@1.2.3.4:5060 and will send the invitation there. A registrar is very often a logical entity only. Because of their tight coupling with proxies registrars, are usually co-located with proxy servers.

Figure 1-3 shows a typical SIP registration. A REGISTER message containing Address of Record sip:jan@iptel.org and contact address sip:jan@1.2.3.4:5060 where 1.2.3.4 is IP address of the phone, is sent to the registrar. The registrar extracts this information and stores it into the location database. If everything went well then the registrar sends a 200 OK response to the phone and the process of registration is finished.

Figure 1-3. Registrar Overview



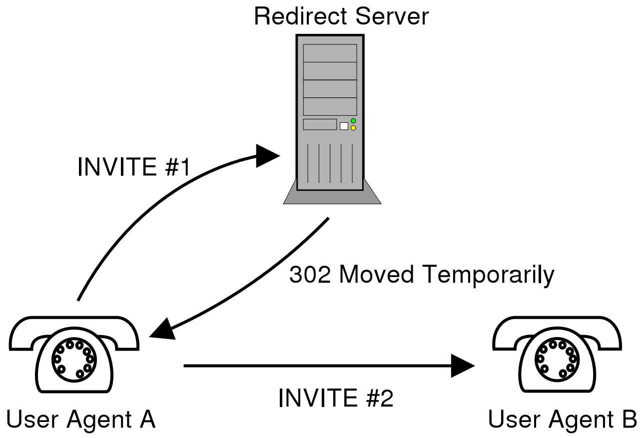
Each registration has a limited lifespan. Expires header field or expires parameter of Contact header field determines for how long is the registration valid. The user agent must refresh the registration within the lifespan otherwise it will expire and the user will become unavailable.

1.3.4. Redirect Server

The entity that receives a request and sends back a reply containing a list of the current location of a particular user is called *redirect server*. A redirect server receives requests and looks up the intended recipient of the request in the location database created by a registrar. It then creates a list of current locations of the user and sends it to the request originator in a response within 3xx class.

The originator of the request then extracts the list of destinations and sends another request directly to them. Figure 1-4 shows a typical redirection.

Figure 1-4. SIP Redirection



1.4. SIP Messages

Communication using SIP (often called signalling) comprises of series of *messages*. Messages can be transported independently by the network. Usually they are transported in a separate UDP datagram each. Each message consist of “first line”, message header, and message body. The first line identifies type of the message. There are two types of messages--*requests* and *responses*. Requests are usually used to initiate some action or inform recipient of the request of something. Replies are used to confirm that a request was received and processed and contain the status of the processing.

A typical SIP request looks like this:

```
INVITE sip:7170@iptel.org SIP/2.0
Via: SIP/2.0/UDP 195.37.77.100:5040;rport
Max-Forwards: 10
From: "jiri" <sip:jiri@iptel.org>;tag=76ff7a07-c091-4192-84a0-d56e91fe104f
To: <sip:jiri@bat.iptel.org>
Call-ID: d10815e0-bf17-4afa-8412-d9130a793d96@213.20.128.35
CSeq: 2 INVITE
Contact: <sip:213.20.128.35:9315>
User-Agent: Windows RTC/1.0
Proxy-Authorization: Digest username="jiri", realm="iptel.org",
  algorithm="MD5", uri="sip:jiri@bat.iptel.org",
  nonce="3cef753900000001771328f5ae1b8b7f0d742da1feb5753c",
  response="53fe98db10e1074
b03b3e06438bda70f"
Content-Type: application/sdp
Content-Length: 451

v=0
o=jku2 0 0 IN IP4 213.20.128.35
s=session
c=IN IP4 213.20.128.35
b=CT:1000
```

```
t=0 0
m=audio 54742 RTP/AVP 97 111 112 6 0 8 4 5 3 101
a=rtpmap:97 red/8000
a=rtpmap:111 SIREN/16000
a=fmtp:111 bitrate=16000
a=rtpmap:112 G7221/16000
a=fmtp:112 bitrate=24000
a=rtpmap:6 DVI4/16000
a=rtpmap:0 PCMU/8000
a=rtpmap:4 G723/8000
a=rtpmap: 3 GSM/8000
a=rtpmap:101 telephone-event/8000
a=fmtp:101 0-16
```

The first line tells us that this is INVITE message which is used to establish a session. The URI on the first line--sip:7170@iptel.org is called *Request URI* and contains URI of the next hop of the message. In this case it will be host iptel.org.

A SIP request can contain one or more Via header fields which are used to record path of the request. They are later used to route SIP responses exactly the same way. The INVITE message contains just one Via header field which was created by the user agent that sent the request. From the Via field we can tell that the user agent is running on host 195.37.77.100 and port 5060.

From and To header fields identify initiator (caller) and recipient (callee) of the invitation (just like in SMTP where they identify sender and recipient of a message). From header field contains a tag parameter which serves as a dialog identifier and will be described in Section 1.6.

Call-ID header field is a dialog identifier and it's purpose is to identify messages belonging to the same call. Such messages have the same Call-ID identifier. CSeq is used to maintain order of requests. Because requests can be sent over an unreliable transport that can re-order messages, a sequence number must be present in the messages so that recipient can identify retransmissions and out of order requests.

Contact header field contains IP aaddress and port on which the sender is awaiting further requests sent by callee. Other header fields are not important and will be not described here.

Message header is delimited from message body by an empty line. Message body of the INVITE request contains a description of the media type accepted by the sender and encoded in SDP.

1.4.1. SIP Requests

We have described how an INVITE request looks like and said that the request is used to invite a callee to a session. Other important requests are:

- *ACK*--This message acknowledges receipt of a final response to INVITE. Establishing of a session utilizes 3-way hand-shaking due to asymmetric nature of the invitation. It may take a while before the callee accepts or declines the call so the callee's user agent periodically retransmits a positive final response until it receives an ACK (which indicates that the caller is still there and ready to communicate).

- *BYE*--Bye messages are used to tear down multimedia sessions. A party wishing to tear down a session sends a BYE to the other party.
- *CANCEL*--Cancel is used to cancel not yet fully established session. It is used when the callee hasn't replied with a final response yet but the caller wants to abort the call (typically when a callee doesn't respond for some time).
- *REGISTER*--Purpose of REGISTER request is to let registrar know of current user's location. Information about current IP address and port on which a user can be reached is carried in REGISTER messages. Registrar extracts this information and puts it into a location database. The database can be later used by SIP proxy servers to route calls to the user. Registrations are time-limited and need to be periodically refreshed.

The listed requests usually have no message body because it is not needed in most situations (but can have one). In addition to that many other request types have been defined but their description is out of the scope of this document.

1.4.2. SIP Responses

When a user agent or proxy server receives a request it send a reply. Each request must be replied except ACK requests which trigger no replies.

A typical reply looks like this:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.168.1.30:5060;received=66.87.48.68
From: sip:sip2@iptel.org
To: sip:sip2@iptel.org;tag=794fe65c16edfdf45da4fc39a5d2867c.b713
Call-ID: 2443936363@192.168.1.30
CSeq: 63629 REGISTER
Contact: <sip:sip2@66.87.48.68:5060;transport=udp>;q=0.00;expires=120
Server: Sip EXpress router (0.8.11pre21xrc (i386/linux))
Content-Length: 0
Warning: 392 195.37.77.101:5060 "Noisy feedback tells:
    pid=5110 req_src_ip=66.87.48.68 req_src_port=5060 in_uri=sip:iptel.org
    out_uri=sip:iptel.org via_cnt==1"
```

As we can see, responses are very similar to the requests, except for the first line. The first line of response contains protocol version (SIP/2.0), reply code, and reason phrase.

The *reply code* is an integer number from 100 to 699 and indicates type of the response. There are 6 classes of responses:

- *1xx* are *provisional* responses. A provisional response is response that tells to its recipient that the associated request was received but result of the processing is not known yet. Provisional responses are sent only when the processing doesn't finish immediately. The sender must stop retransmitting the request upon reception of a provisional response.

  Typically proxy servers send responses with code 100 when they start processing an INVITE and user agents send responses with code 180 (Ringing) which means that the callee's phone is ringing.

- *2xx* responses are *positive final* responses. A final response is the ultimate response that the originator of the request will ever receive. Therefore final responses express result of the processing of the associated request. Final responses also terminate transactions. Responses with code from 200 to 299 are positive responses that means that the request was processed successfully and accepted. For instance a 200 OK response is sent when a user accepts invitation to a session (INVITE request).

  A UAC may receive several 200 messages to a single INVITE request. This is because a forking proxy (described later) can fork the request so it will reach several UAS and each of them will accept the invitation. In this case each response is distinguished by the tag parameter in To header field. Each response represents a distinct dialog with unambiguous dialog identifier.

- *3xx* responses are used to redirect a caller. A redirection response gives information about the user's new location or an alternative service that the caller might use to satisfy the call. Redirection responses are usually sent by proxy servers. When a proxy receives a request and doesn't want or can't process it for any reason, it will send a redirection response to the caller and put another location into the response which the caller might want to try. It can be the location of another proxy or the current location of the callee (from the location database created by a registrar). The caller is then supposed to re-send the request to the new location. 3xx responses are final.
- *4xx* are *negative final* responses. a 4xx response means that the problem is on the sender's side. The request couldn't be processed because it contains bad syntax or cannot be fulfilled at that server.
- *5xx* means that the problem is on server's side. The request is apparently valid but the server failed to fulfill it. Clients should usually retry the request later.
- *6xx* reply code means that the request cannot be fulfilled at any server. This response is usually sent by a server that has definitive information about a particular user. User agents usually send a 603 Decline response when the user doesn't want to participate in the session.

In addition to the response class the first line also contains *reason phrase*. The code number is intended to be processed by machines. It is not very human-friendly but it is very easy to parse and understand by machines. The reason phrase usually contains a human-readable message describing the result of the processing. A user agent should render the reason phrase to the user.

The request to which a particular response belongs is identified using the CSeq header field. In addition to the sequence number this header field also contains method of corresponding request. In our example it was REGISTER request.

1.5. SIP Transactions

Although we said that SIP messages are sent independently over the network, they are usually arranged into *transactions* by user agents and certain types of proxy servers. Therefore SIP is said to be a *transactional protocol*.

A transaction is a sequence of SIP messages exchanged between SIP network elements. A transaction consists of one request and all responses to that request. That includes zero or more provisional responses and one or more final responses (remember that an INVITE might be answered by more than one final response when a proxy server forks the request).

If a transaction was initiated by an INVITE request then the same transaction also includes ACK, but only if the final response was not a 2xx response. If the final response was a 2xx response then the ACK is not considered part of the transaction.

As we can see this is quite asymmetric behavior--ACK is part of transactions with a negative final response but is not part of transactions with positive final responses. The reason for this separation is the importance of delivery of all 200 OK messages. Not only that they establish a session, but also 200 OK can be generated by multiple entities when a proxy server forks the request and all of them must be delivered to the calling user agent. Therefore user agents take responsibility in this case and retransmit 200 OK responses until they receive an ACK. Also note that only responses to INVITE are retransmitted !

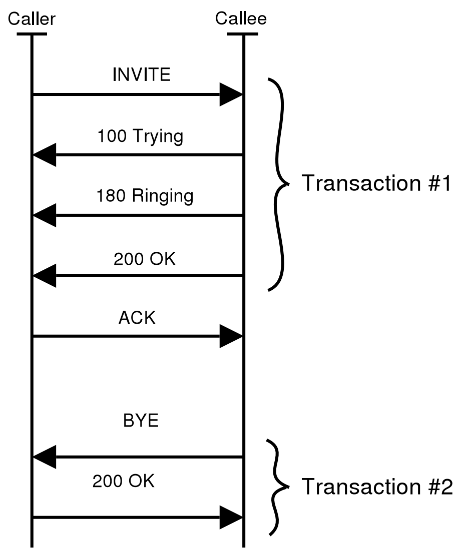
SIP entities that have notion of transactions are called *stateful*. Such entities usually create a state associated with a transaction that is kept in the memory for the duration of the transaction. When a request or response comes, a stateful entity tries to associate the request (or response) to existing transactions. To be able to do it it must extract a unique transaction identifier from the message and compare it to identifiers of all existing transactions. If such a transaction exists then it's state gets updated from the message.

In the previous SIP RFC2543 (http://www.ietf.org/rfc/rfc2543.txt) the transaction identifier was calculated as hash of all important message header fields (that included To, From, Request-URI and CSeq). This proved to be very slow and complex, during interoperability tests such transaction identifiers used to be a common source of problems.

In the new RFC3261 (http://www.ietf.org/rfc/rfc3261.txt) the way of calculating transaction identifiers was completely changed. Instead of complicated hashing of important header fields a SIP message now includes the identifier directly. Branch parameter of Via header fields contains directly the transaction identifier. This is significant simplification, but there still exist old implementations that don't support the new way of calculating of transaction identifier so even new implementations have to support the old way. They must be backwards compatible.

Figure 1-5 shows what messages belong to what transactions during a conversation of two user agents.

Figure 1-5. SIP Transactions

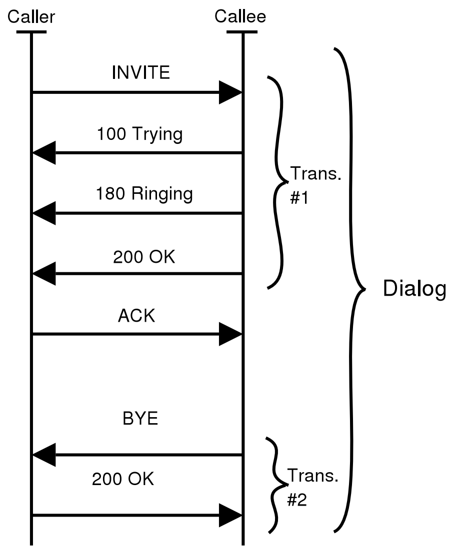


1.6. SIP Dialogs

We have shown what transactions are, that one transaction includes INVITE and it's responses and another transaction includes BYE and it responses when a session is being torn down. But we feel that those two transactions should be somehow related--both of them belong to the same *dialog*. A dialog represents a peer-to-peer SIP relationship between two user agents. A dialog persists for some time and it is very important concept for user agents. Dialogs facilitate proper sequencing and routing of messages between SIP endpoints.

Dialogs are identified using Call-ID, From tag, and To tag. Messages that have these three identifiers same belong to the same dialog. We have shown that CSeq header field is used to order messages, in fact it is used to order messages within a dialog. The number must be monotonically increased for each message sent within a dialog otherwise the peer will handle it as out of order request or retransmission. In fact, the CSeq number identifies a transaction within a dialog because we have said that requests and associated responses are called transaction. This means that only one transaction in each direction can be active within a dialog. One could also say that a *dialog is a sequence of transactions*. Figure 1-6 extends Figure 1-5 to show which messages belong to the same dialog.

Figure 1-6. SIP Dialog



Some messages establish a dialog and some do not. This allows to explicitly express the relationship of messages and also to send messages that are not related to other messages outside a dialog. That is easier to implement because user agent don't have to keep the dialog state.

For instance, INVITE message establishes a dialog, because it will be later followed by BYE request which will tear down the session established by the INVITE. This BYE is sent within the dialog established by the INVITE.

But if a user agent sends a MESSAGE request, such a request doesn't establish any dialog. Any subsequent messages (even MESSAGE) will be sent independently of the previous one.

1.6.1. Dialogs Facilitate Routing

We have said that dialogs are also used to route the messages between user agents, let's describe this a little bit.

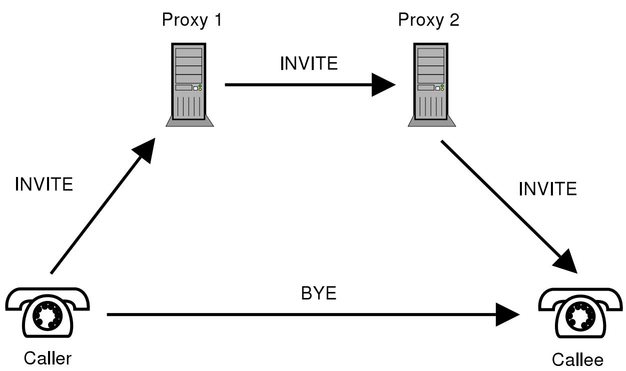
Let's suppose that user sip:bob@a.com wants to talk to user sip:pete@b.com. He knows SIP address of the callee (sip:pete@b.com) but this address doesn't say anything about current location of the user--i.e. the caller doesn't know to which host to send the request. Therefore the INVITE request will be sent to a proxy server.

The request will be sent from proxy to proxy until it reaches one that knows current location of the callee. This process is called routing. Once the request reaches the callee, the callee's user agent will create a response that will be sent back to the caller. Callee's user agent will also put Contact header field into the response which will contain the current location of the user. The original request also contained Contact header field which means that both user agents know the current location of the peer.

Because the user agents know location of each other, it is not necessary to send further requests to any proxy--they can be sent directly from user agent to user agent. That's exactly how dialogs facilitate routing.

Further messages within a dialog are sent directly from user agent to user agent. This is a significant performance improvement because proxies do not see all the messages within a dialog, they are used to route just the first request that establishes the dialog. The direct messages are also delivered with much smaller latency because a typical proxy usually implements complex routing logic. Figure 1-7 contains an example of a message within a dialog (BYE) that bypasses the proxies.

Figure 1-7. SIP Trapezoid



1.6.2. Dialog Identifiers

We have already shown that dialog identifiers consist of three parts, Call-Id, From tag, and To tag, but it is not that clear why are dialog identifiers created exactly this way and who contributes which part.

Call-ID is so called *call identifier*. It must be a unique string that identifies a call. A call consists of one or more dialogs. Multiple user agents may respond to a request when a proxy along the path forks the request. Each user agent that sends a 2xx establishes a separate dialog with the caller. All such dialogs are part of the same call and have the same Call-ID.

From tag is generated by the caller and it uniquely identifies the dialog in the caller's user agent.

To tag is generated by a callee and it uniquely identifies, just like From tag, the dialog in the callee's user agent.

This hierarchical dialog identifier is necessary because a single call invitation can create several dialogs and caller must be able to distinguish them.

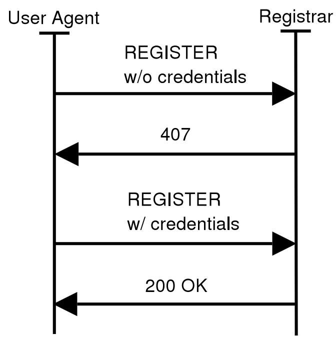
1.7. Typical SIP Scenarios

This section gives a brief overview of typical SIP scenarios that usually make up the SIP traffic.

1.7.1. Registration

Users must register themselves with a registrar to be reachable by other users. A registration comprises a REGISTER message followed by a 200 OK sent by registrar if the registration was successful. Registrations are usually authorized so a 407 reply can appear if the user didn't provide valid credentials. Figure 1-8 shows an example of registration.

Figure 1-8. REGISTER Message Flow

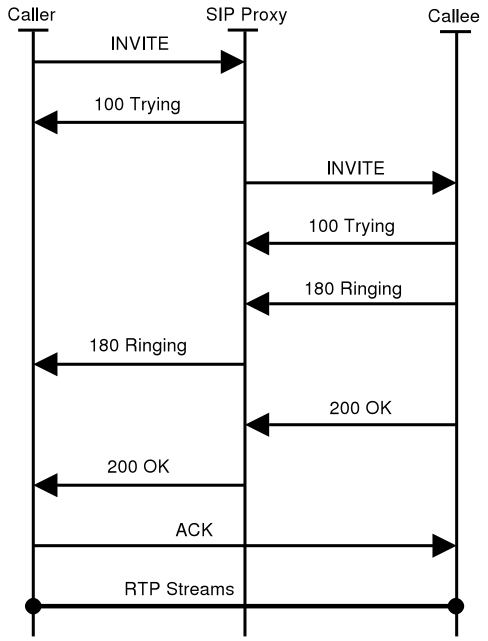


1.7.2. Session Invitation

A session invitation consists of one INVITE request which is usually sent to a proxy. The proxy sends immediately a 100 Trying reply to stop retransmissions and forwards the request further.

All provisional responses generated by callee are sent back to the caller. See 180 Ringing response in the call flow. The response is generated when callee's phone starts ringing.

Figure 1-9. INVITE Message Flow



A 200 OK is generated once the callee picks up the phone and it is retransmitted by the callee's user agent until it receives an ACK from the caller. The session is established at this point.

1.7.3. Session Termination

Session termination is accomplished by sending a BYE request within dialog established bye INVITE. BYE messages are sent directly from one user agent to the other unless a proxy on the path of the INVITE request indicated that it wishes to stay on the path by using record routing (see Section 1.7.4.

Party wishing to tear down a session sends a BYE request to the other party involved in the session. The other party sends a 200 OK response to confirm the BYE and the session is terminated. See Figure 1-10, left message flow.

1.7.4. Record Routing

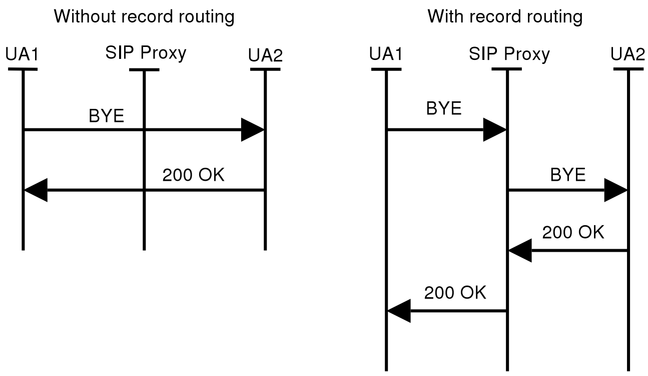
All requests sent within a dialog are by default sent directly from one user agent to the other. Only requests outside a dialog traverse SIP proxies. This approach makes SIP network more scalable because only a small number of SIP messages hit the proxies.

There are certain situations in which a SIP proxy need to stay on the path of all further messages. For instance, proxies controlling a NAT box or proxies doing accounting need to stay on the path of BYE requests.

Mechanism by which a proxy can inform user agents that it wishes to stay on the path of all further messages is called *record routing*. Such a proxy would insert Record-Route header field into SIP messages which contain address of the proxy. Messages sent within a dialog will then traverse all SIP proxies that put a Record-Route header field into the message.

The recipient of the request receives a set of Record-Route header fields in the message. It must mirror all the Record-Route header fields into responses because the originator of the request also needs to know the set of proxies.

Figure 1-10. BYE Message Flow (With and without Record Routing)



Left message flow of Figure 1-10 show how a BYE (request within dialog established by INVITE) is sent directly to the other user agent when there is no Record-Route header field in the message. Right message flow show how the situation changes when the proxy puts a Record-Route header field into the message.

1.7.4.1. Strict versus Loose Routing

The way how record routing works has evolved. Record routing according to RFC2543 (http://www.ietf.org/rfc/rfc2543.txt) rewrote the Request-URI. That means the Request-URI always contained URI of the next hop (which can be either next proxy server which inserted Record-Route header field or destination user agent). Because of that it was necesarry to save the original Request-URI as the last Route header field. This approach is called *strict routing*.

Loose routing, as specified in RFC3261 (http://www.ietf.org/rfc/rfc3261.txt), works in a little bit different way. The Request-URI is no more overwritten, it always contains URI of the destination user agent. If there are any Route header field in a message, than the message is sent to the URI from the topmost Route header field. This is significant change--Request-URI doesn't necessarily contain URI to which the request will be sent. In fact, loose routing is very similar to IP source routing.

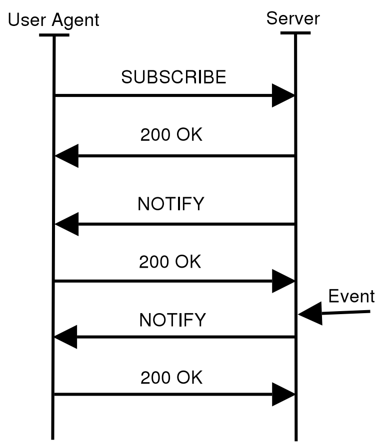
Because transit from strict routing to loose routing would break backwards compatibility and older user agents wouldn't work, it is necesarry to make loose routing backwards compatible. The backwards compatibility unfortunately adds a lot of overhead and is often source of major problems.

1.7.5. Event Subscription And Notification

The SIP specification has been extended to support a general mechanism allowing subscription to asynchronous events. Such evens can include SIP proxy statistics changes, presence information, session changes and so on.

The mechanism is used mainly to convey information on presence (willingness to communicate) of users. Figure 1-11 shows the basic message flow.

Figure 1-11. Event Subscription And Notification



A user agent interested in event notification sends a SUBSCRIBE message to a SIP server. The SUBSCRIBE message establishes a dialog and is immediately replied by the server using 200 OK response. At this point the dialog is established. The server sends a NOTIFY request to the user every time the event to which the user subscribed changes. NOTIFY messages are sent within the dialog established by the SUBSCRIBE.

Note that the first NOTIFY message in Figure 1-11 is sent regardless of any event that triggers notifications.

Subscriptions--as well as registrations--have limited lifespan and therefore must be periodically refreshed.

1.7.6. Instant Messages

Instant messages are sent using MESSAGE request. MESSAGE requests do not establish a dialog and therefore they will always traverse the same set of proxies. This is the simplest form of sending instant messages. The text of the instant message is transported in the body of the SIP request.

Figure 1-12. Instant Messages