Introduction:

The Session Initiation Protocol (SIP) is an application-layer control protocol that can establish, modify and terminate multimedia sessions or calls. These multimedia sessions include multimedia conferences, distance learning, Internet telephony and similar applications. SIP can invite both persons and "robots", such as a media storage service. SIP can invite parties to both unicast and multicast sessions; the initiator does not necessarily have to be a member of the session to which it is inviting. Media and participants can be added to an existing session. SIP can be used to initiate sessions as well as invite members to sessions that have been advertised and established by other means. Sessions can be advertised using multicast protocols such as SAP, electronic mail, news groups, web pages or directories (LDAP), among others.

Histroy:

SIP has its origins in late 1996 as a component of the Mbone set of utilities and protocols. The Mbone, or multicast backbone, was an experimental multicast network overlayed on top of the public Internet. It was used for distribution of multimedia content, including talks and seminars, broadcasts of space shuttle launches, and IETF meetings. One of its essential components was a mechanism for inviting users to listen in on an ongoing or future multimedia session on the Internet. Basically - a session initiation protocol. Since its approval in early 1999 as an official standard, the Session Initiation Protocol has gained tremendous market acceptance for signaling communications services on the Internet.
Despite its historical strengths, SIP saw relatively slow progress throughout 1996 and 1997. That’s about when interest in Internet telephony began to take off. People began to see SIP as a technology that would also work for VoIP, not just Mbone sessions. The result was an intensified effort towards completing the specification in late 1998, and completion by the end of the year. It received official approval as an RFC (Request for Comments, the official term for an IETF standard) in February and issuance of an RFC number, 2543, in March.

From there, industry acceptance of SIP grew exponentially. Its scalability, extensibility, and - most important - flexibility appealed to service providers and vendors who had needs that a vertically integrated protocol, such as H.323, could not address. Among service providers MCI (particularly MCI's Henry Sinnreich, regarded as the Pope of SIP) led the evangelical charge. Throughout 1999 and into 2000, it saw adoption by most major vendors, and announcements of networks by service providers. Interoperability bake-offs were held throughout 1999, attendance doubling at each successive event. Tremendous success was achieved in interoperability among vendors. Other standards bodies began to look at SIP as well, including ITU and ETSI TIPHON, IMTC, Softswitch Consortium, and JAIN.

Premise:

As an Mbone tool (and as a product of the IETF), SIP was designed with certain assumptions in mind. First was scalability:
Since users could reside anywhere on the Internet, the protocol needed to workwide-area from day one. Users could be invited to lots of sessions, so the protocol needed to scale in both directions. A second assumption was component reuse: Rather than inventing new protocol tools, those already developed within the IETF would be used. That included things like MIME, URLs, and SDP (already used for other protocols, such as SAP). This resulted in a protocol that integrated well with other IP applications (such as web and e-mail).
Interoperability was another key goal, although not one specific to SIP. Interoperability is at the heart of IETF's process and operation, as a forum attended by implementers and operational experts who actually build and deploy the technologies they design. To these practical-minded standardizers, the KISS (Keep It Simple Stupid) principle was the best way to help ensure correctness and interoperability.

Operation breakdown

- establishing connection
- adding parties
- changing session parameters
- terminating multimedia communications

- User location: determination of the end system
- User capabilities: determination of the media and parameters
- User availability: determination of the willingness for communications
- Call setup: "ringing", setting call parameters at called and calling party

As the name implies, the session initiation protocol (SIP) is about initiation of interactive communications sessions between users. SIP also handles termination and modifications of sessions as well. SIP actually doesn't define what a session is; this is described by content carried in SIP messages. Most of SIP is about the initiation part, since this is really the most difficult aspect. Initiating a session requires determining

where the user to be contacted is actually residing at a particular moment. A user might have a PC at work, a PC at home, and an IP desk phone in the lab. A call for that user might need to ring all phones at once. Furthermore, the user might be mobile; one day at work, and the next day visiting a university. This dynamic location information needs to be taken into account in order to find the user.

Once the user to be called has been located, SIP can perform its second main function - delivering a description of the session that the user is being invited to. As mentioned, SIP itself does not know about the details of the session. What SIP does do is convey information about the protocol used to describe the session. SIP does this through the use of multipurpose internet mail extensions (MIME), widely used in web and e-mail services to describe content (HTML, audio, video, etc.). The most common protocol used to describe sessions is the session description protocol (SDP), described in RFC2327. SIP can also be used to negotiate a common format for describing sessions, so that other things besides SDP can be used.

Once the user has been located and the session description delivered, SIP is used to convey the response to the session initiation (accept, reject, etc.). If accepted, the session is now active. SIP can be used to modify the session as well. Doing so is easy - the originator simply re-initiates the session, sending the same message as the original, but with a new session description. For this reason, modification of sessions (which includes things like adding and removing audio streams, adding video, changing codecs, hold and mute) are easily supported with SIP, so long as the session description protocol can support them (SDP supports all of the above).

Finally, SIP can be used to terminate the session (i.e., hang up)

Type of Operation:

SIP is designed as part of the overall IETF multimedia data and control architecture.This multimedia data and control architecture is currently
incorporating protocols such as

- RTP the real-time transport protocol for transporting real-time data and providing QOS feedback,
- RTSP the real-time streaming protocol for controlling delivery of streaming media,
- SAP the session announcement protocol for advertising multimedia sessions via multicast, and
- SDP the session description protocol for describing multimedia sessions.

the functionality and operation of SIP does not depend on any of these protocols!!

SIP is based on the request-response paradigm. To initiate a session, the caller (known as the User Agent Client, or UAC) sends a request (called an INVITE), addressed to the person the caller wants to talk to. In SIP, addresses are URLs. SIP defines a URL format that is very similar to the popular mailto URL. If the user's e-mail address is jdrosen@dynamic-soft.com, their SIP URL would be sip:jdrosen@dynamicsoft.com. This message is not sent directly to the called party, but rather to an entity known as a proxy server. The proxy server is responsible for routing and delivering messages to the called party. The called party then sends a response, accepting or rejecting the invitation, which is forwarded back through the same set of proxies, in reverse order.

A proxy can receive a single INVITE request, and send out more than one INVITE request to different addresses. This feature, aptly called forking, allows a session initiation attempt to reach multiple locations, in the hopes of finding the desired user at one of them. A close analogy is the home phone line service, where all phones in the home ring at once.

More detailed description on request - response scheme

Performing Calls

This section explains the basic protocol functionality and operation. Callers and callees are identified by SIP addresses, described in Section SIP-URL. When making a SIP call, a caller first locates the appropriate server (Section SERVER) and then sends a SIP request (Section Type of Operation). The most common SIP operation is the invitation. Instead of directly reaching the intended callee, a SIP request may be redirected or may trigger a chain of new SIP requests by proxies. Users can register their location(s) with SIP servers.

Assuming the caller (jdrosen@dynamicsoft.com) wishes to place a call to joe@columbia.edu. Jdrosen sends his SIP INVITE message to the proxy for dynamicsoft.com (Step 1). This proxy then forwards the request out to Columbia, where it reaches the Columbia.edu server (Step 2).
This server is actually not a proxy, but a similar device called a redirect server. Instead of forwarding calls, a redirect server asks the requestor to contact the next server directly. The Columbia.edu server looks up Joe in its database, and determines that today, Joe is on sabbatical to foo.com. It therefore sends a special response, called a redirect, to the dynamicsoft.com proxy, instructing it to instead try joe@foo.com (Step 3).
The dynamicsoft proxy then acts on this response, which means it directly tries to contact joe@foo.com. So, its sends the INVITE to the foo.com server (Step 4). This server consults its database (Step 5), and learns (Step 6) that Joe is actually in sales. So, it constructs a new URL, joe@sales. foo.com, and sends the INVITE to the sales.foo.com proxy (Step 7).

The proxy for the sales department then needs to forward the INVITE to the PC where Jo is currently sitting. For getting out which PC Joe is currently using, SIP defines another request, called REGISTER, which is used to inform a proxy of an address binding. In this case, when Joe turned on his SIP client on his PC, the client would register the binding sip:joe@sales.engineering.com to sip:joe@mypc.sales.foo.com. This would allow the proxy to know that Joe is actually at mypc, a specific host on the network. The bindings registered through SIP are periodically refreshed, so that if the PC crashes, the binding is eventually removed.
The sales.foo.com proxy consults this registration database, and forwards the INVITE to joe@mypc.sales.foo.com (Step 8). This INVITE then reaches Joe at his PC. Joe can then respond to it (thus the request-response model). SIP provides many responses, and these include acceptance, rejection, redirection, busy, and so on. The response is forwarded back through the proxies to the original caller (Steps 9,10,11,12). An acknowledgement is sent (another type of request, called ACK) in Step 13, and the session is established. Media can then flow (Step 14).

SIP addressing:

A SIP URL follows the guidelines of RFC 2396 and has the syntax shown here. It is described using Augmented Backus-Naur Form. Note that reserved characters have to be escaped and that the "set of characters" reserved within any given URI component is defined by that component. In general, a character is reserved if the semantics of the URI changes if the character is replaced with its escaped US-ASCII encoding

SIP URLs are used within SIP messages to indicate the originator (From), current destination (Request-URI) and final recipient (To) of a SIP request, and to specify redirection addresses (Contact). A SIP URL can also be embedded in web pages or other hyperlinks to indicate that a particular user or service can be called via SIP. When used as a hyperlink, the SIP URL indicates the use of the INVITE method. The SIP URL scheme is defined to allow setting SIP request-header fields and the SIP message-body. This corresponds to the use of mailto: URLs. It makes it possible, for example, to specify the subject, urgency or media types of calls initiated through a web page or as part of an email message.

Some examples for use and default values of URL components for SIP headers:

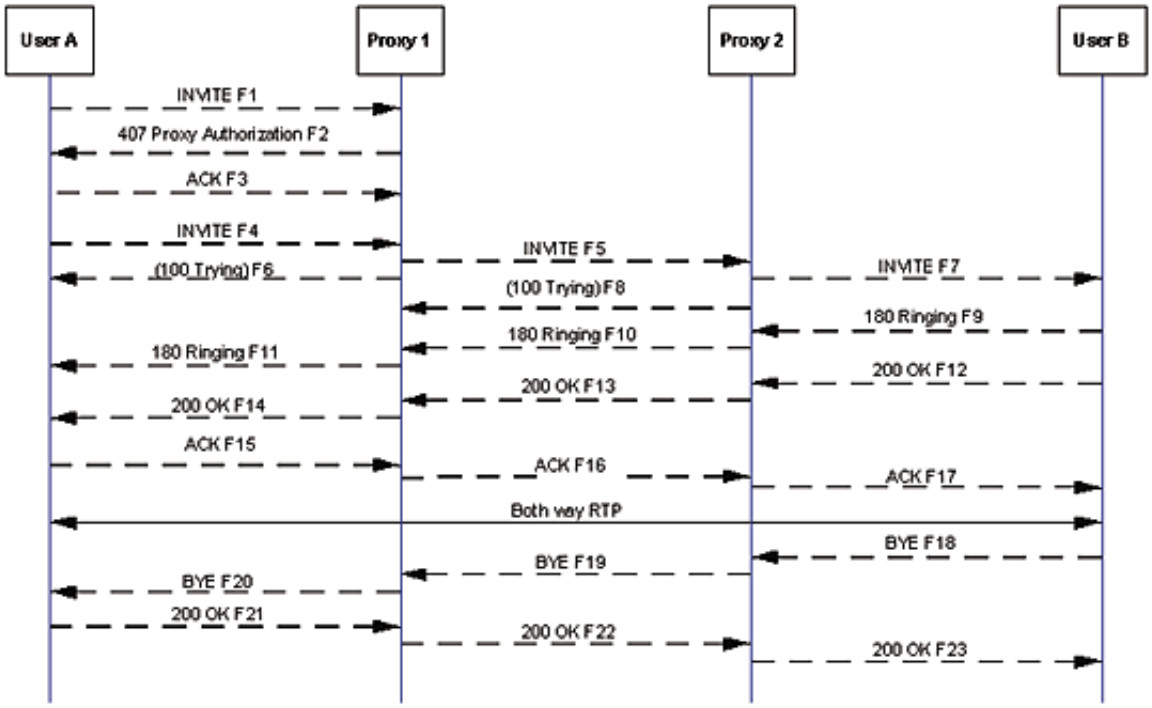
sip:j.doe@big.com
sip:j.doe:secret@big.com;transport=tcp
sip:j.doe@big.com?subject=project
sip:+1-212-555-1212:1234@gateway.com;user=phone
sip:1212@gateway.com
sip:alice@10.1.2.3
sip:alice@example.com
sip:alice%40example.com@gateway.com
sip:alice@registrar.com;method=REGISTER

SIP URLs are case-insensitive, so that for example the two URLs sip:j.doe@example.com and SIP:J.Doe@Example.com are equivalent. All URL parameters are included when comparing SIP URLs for equality. The Request-URI is a SIP URL or a general URI. It indicates the user or service to which this request is being addressed. Unlike the To field, the Request-URI MAY be re-written by proxies

---

Example: SIP Call Flow

Basic Call Flow



In Figure A, Caller A completes a call to User B using two proxies: Proxy 1 and Proxy 2. The initial INVITE (F1) does not contain the Authorization credentials that Proxy 1 requires, so an Authorization response is sent containing the challenge information.A new INVITE (F4) is then sent containing the correct credentials and the call proceeds. The call terminates when User B disconnects by initiating a BYE message.

F1 INVITE A -> Proxy 1

The call begins, as always, with an INVITE message that contains information on caller and called party as well as the session description request (2nd part).

```
INVITE sip:UserB@ss1.wcom.com SIP/2.0
Via: SIP/2.0/UDP here.com:5060
From: BigGuy
To: LittleGuy
Call-ID: 12345600@here.com
CSeq: 1 INVITE
Contact: BigGuy
Content-Type: application/sdp
Content-Length: 147

v=0
o=UserA 2890844526 2890844526 IN IP4 here.com
s=Session SDP
c=IN IP4 100.101.102.103
```

t=0 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000

F2 407 Proxy Authorization Required Proxy 1 -> User A

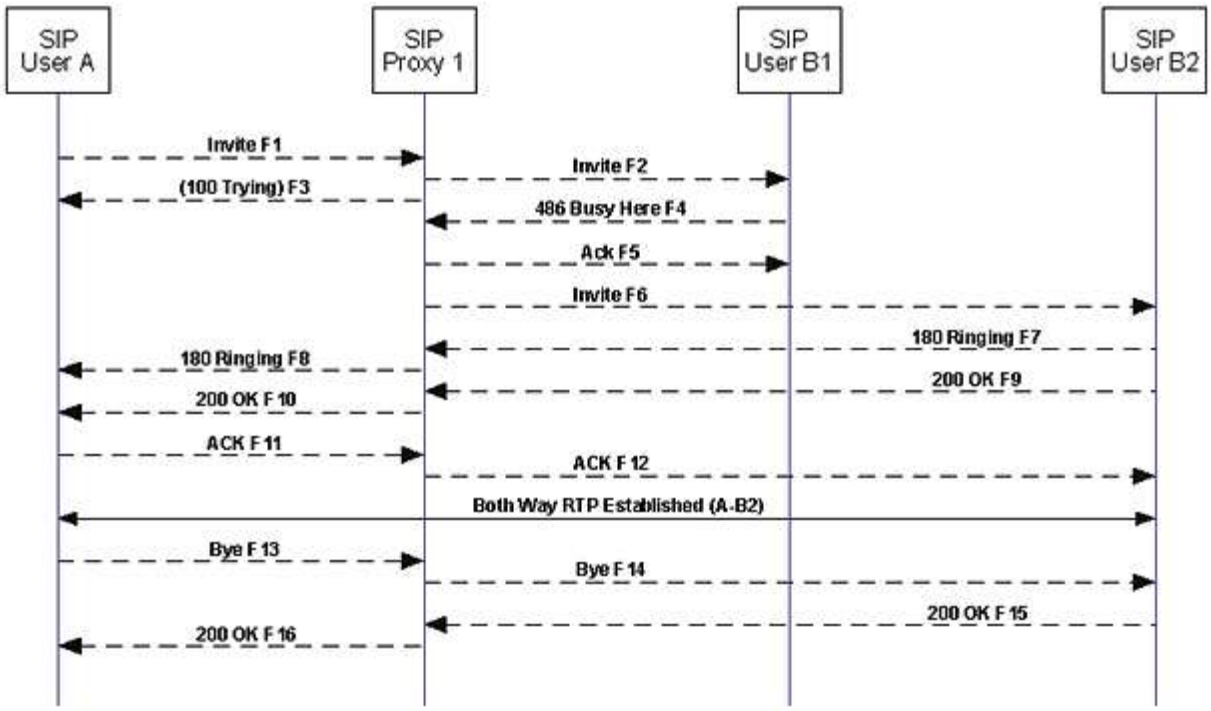
SIP always works in a request-response mode and in this example Proxy 1 challenges Caller A for authentication

SIP/2.0 407 Proxy Authorization Required
Via: SIP/2.0/UDP here.com:5060
From: BigGuy
To: LittleGuy
Call-ID: 12345600@here.com
CSeq: 1 INVITE
Proxy-Authenticate: Digest realm="MCI WorldCom SIP",
domain="wcom.com", nonce="wf84f1ceczx41ae6cbe5aea9c8e88d359",
opaque="", stale="FALSE", algorithm="MD5"
Content-Length: 0

As we move further down the call flow, the actual voice call begins, using Realtime Transport Protocol (RTP) to move the voice stream.

F17 ACK Proxy 2 -> B
ACK sip: UserB@there.com SIP/2.0
Via: SIP/2.0/UDP ss2.wcom.com:5060
Via: SIP/2.0/UDP ss1.wcom.com:5060
Via: SIP/2.0/UDP here.com:5060
From: BigGuy
To: LittleGuy ;tag=314159
Call-ID: 12345601@here.com
CSeq: 1 ACK
Content-Length: 0
Calls are then terminated with a BYE request to the caller.
F18 BYE User B -> Proxy 2
BYE sip: UserA@ss2.wcom.com SIP/2.0
Via: SIP/2.0/UDP there.com:5060
Route: ,
From: LittleGuy ;tag=314159
To: BigGuy
Call-ID: 12345601@here.com
CSeq: 1 BYE
Content-Length: 0

Call Forward On Busy



In this scenario User B wants calls forwarded to another destination if the original line is busy. It is assumed that the proxy knows where to forward the call.

F1 INVITE A -> Proxy
INVITE sip:UserB@ss1.wcom.com SIP/2.0
Via: SIP/2.0/UDP here.com:5060
From: BigGuy
To: LittleGuy
Call-ID: 12345600@here.com
CSeq: 1 INVITE
Contact: BigGuy
Proxy-Authorization: DIGEST username="UserA", realm="MCI
WorldCom
SIP", nonce="9137d175c20a0d6eadd7be1c863302ae", opaque="",
uri="sip:ss1.wcom.com",
response="cf25aad811c806bde46a369220158cec"
Content-Type: application/sdp
Content-Length: ...

v=0
o=UserA 2890844526 2890844526 IN IP4 client.here.com
s=Session SDP

c=IN IP4 100.101.102.103
t=0 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000

F4 486 Busy Here B1 -> Proxy

User B’s phone responds back with a busy message (486)

SIP/2.0 486 Busy Here
Via: SIP/2.0/UDP ss1.wcom.com:5060;branch=1
Via: SIP/2.0/UDP here.com:5060
From: BigGuy
To: LittleGuy ;tag=123456
Call-ID: 12345600@here.com
CSeq: 1 INVITE
Content-Length: 0

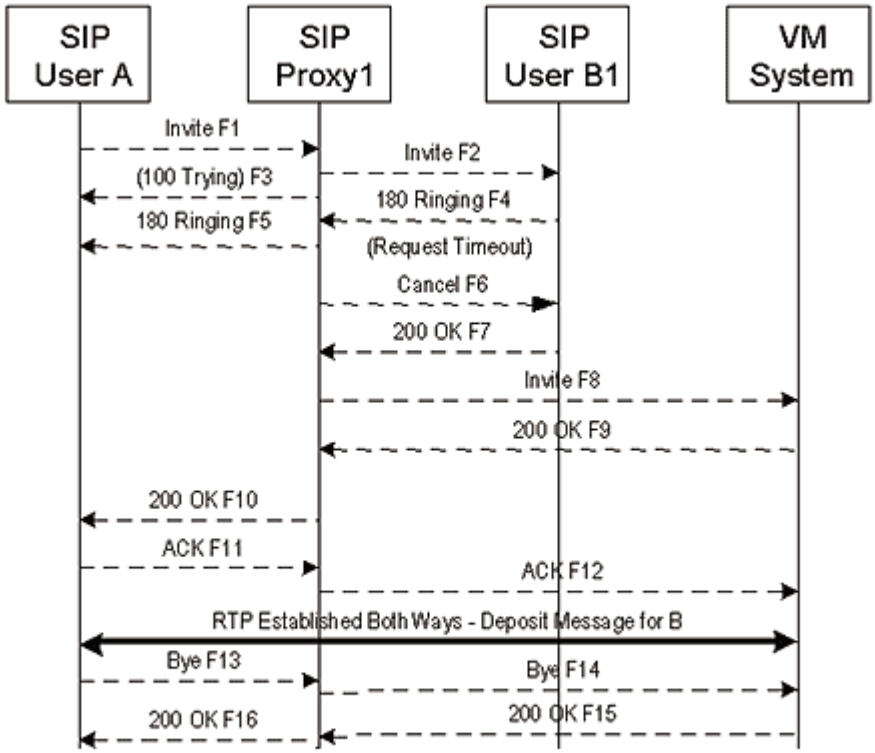
F6 INVITE Proxy -> B2

The call is then forwarded to a new location at "THERE.COM." Since the call is going to a new location, a new INVITE and session description is sent.

INVITE sip:UserB2@ there.com SIP/2.0
Via: SIP/2.0/UDP ss1.wcom.com:5060;branch=2
Via: SIP/2.0/UDP here.com:5060
Record-Route:
From: BigGuy
To: LittleGuy
Call-ID: 12345600@here.com
CSeq: 1 INVITE
Contact: BigGuy
Content-Type: application/sdp
Content-Length: ...

v=0
o=UserA 2890844526 2890844526 IN IP4 client.here.com
s=Session SDP
c=IN IP4 100.101.102.103
t=0 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000

Forwarded on No Answer



http://search.ietf.org/internet-drafts/draft-campbell-sip-service-control-00.txt

Here User A attempts to call User B, who does not answer. The call is forwarded to User B's mailbox, and the voicemail system plays a message for a ring-no-answer. The call flow then moves from User A to the voicemail system.

F1
INVITE sip:UserB@wcom.com SIP/2.0
Via: SIP/2.0/UDP here.com:5060
From: TheBigGuy
To: TheLittleGuy
Call-Id: 12345600@here.com
CSeq: 1 INVITE
Contact: TheBigGuy
Proxy-Authorization:Digest username="UserA",
realm="MCI WorldCom SIP",
nonce="ea9c8e88df84f1cec4341ae6cbe5a359", opaque="",
uri="sip:UserB@wcom.com", response= calculated hash goes here>
Content-Type: application/sdp
Content-Length:

v=0
o=UserA 2890844526 2890844526 IN IP4 client.here.com
s=Session SDP
c=IN IP4 100.101.102.103
t=0 0
m=audio 49170 RTP/AVP 0
a=rtpmap:0 PCMU/8000

Here B1 rings for, let’s say, nine seconds. (This duration is a configurable parameter in the Proxy Server.) The Proxy sends a Cancel option and proceeds down its internal list of options, eventually selecting a voicemail URL for "forward no answer."

SIP/2.0 180 Ringing
F5 Via: SIP/2.0/UDP here.com:5060From: TheBigGuy
To: TheLittleGuy ;tag=3145678
Call-Id: 12345600@here.com
CSeq: 1 INVITE
Content-Length: 0

Here the voicemail system is ready to set up a RTP session, and record the message by using the 200 OK signal. The ’contact’ field indicates where the file is being stored.

F9 SIP/2.0 200 OK
Via: SIP/2.0/UDP wcom.com:5060; branch=2
Via: SIP/2.0/UDP here.com:5060
Record-Route:
From: TheBigGuy
To: TheLittleGuy ;tag=123456
Call-Id: 12345600@here.com
CSeq: 1 INVITE
Contact: TheLittleGuyVoiceMail
Content-Type: application/sdp
Content-Length:

v=0
o=UserB 2890844527 2890844527 IN IP4 vm.wcom.com
s=Session SDP
c=IN IP4 110.111.112.114
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000

Finally User A hangs up on the voicemail system ...but the VM system could have disconnected the call on its own.

F13 BYE sip:UserB@wcom.com SIP/2.0
Via: SIP/2.0/UDP here.com:5060
Route:
From: TheBigGuy
To: TheLittleGuy ;tag=123456
Call-Id: 12345600@here.com
CSeq: 2 BYE
Content-Length: 0

Characteristics:

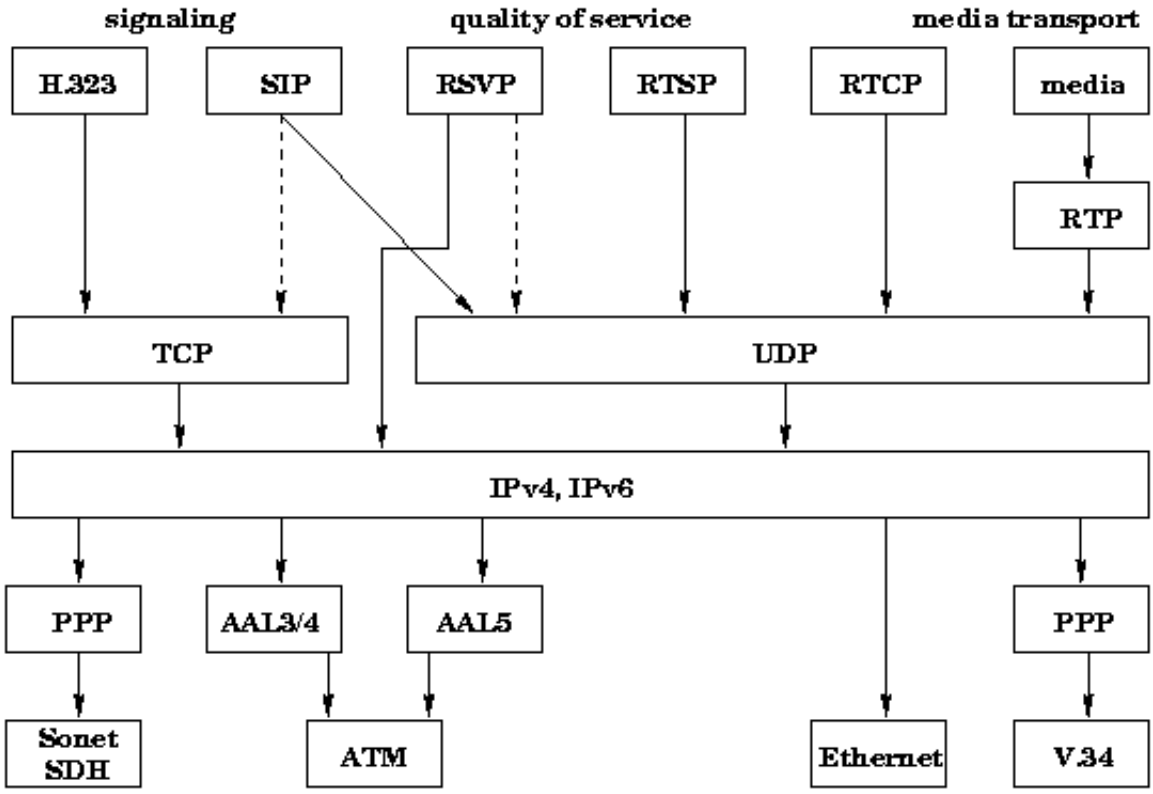
layers
transparently supports name mapping
redirection
transport mode (=text based)
using ABNF

Layers

SIP makes minimal assumptions about the underlying transport and network-layer protocols. The lower-layer can provide either a packet or a byte stream service, with reliable or unreliable service. In an Internet context, SIP is able to utilize both and TCP as transport protocols, among others. UDP allows the application to more carefully control the timing of messages and their retransmission, to perform parallel searches without requiring TCP connection state for each outstanding request, and to use multicast. Routers can more readily snoop SIP UDP packets. TCP allows easier passage through existing firewalls.

possibly lower layers:

UDP
TCP
ATM AAL5
IPX
frame relay
X.25



Messaging:

SIP is patterned after HTTP in many ways. HTTP is also request-response. SIP borrows much of the syntax and semantics from HTTP. The textual message formatting, usage of headers, MIME support, and many headers are identical. An http expert looking at a SIP message would have difficulty distinguishing them.

The 1500 bytes accommodates encapsulation within the "typical" ethernet MTU without IP fragmentation. The next lower common MTU values are 1006 bytes for SLIP and 296 for low-delay PPP (RFC 1191). Thus, another reasonable value would be a message size of 950 bytes, to accommodate packet headers within the SLIP MTU without fragmentation.

Text based

if you are missing explanations on setting Bits in specific Bytes to gain a special function you can seek for a very long time! SIP is not using binary mode messaging for its work.
SIP is text-based, using ISO 10646 in UTF-8 encoding throughout. This allows easy implementation in languages such as Java, Tcl and Perl, allows easy debugging, and most importantly, makes SIP flexible and extensible. As SIP is used for initiating multimedia conferences rather than delivering media data, it is believed that the additional overhead of using a text-based protocol is not significant. Except for the above difference in character sets, much of the message syntax is and header fields are identical to HTTP/1.1, but cannot be seen as an extension to HTTP!

ABNF

For infrormation on this topic you can click here.

Comparing SIP to H.323

There are numerous differences between SIP and H.323. The first is scope;
H.323 specifies a complete, vertically integrated system. Not much room is left for flexibility or different architectures. SIP, on the other hand, is a single component. It works with RTP, for example, but does not mandate it. SIP systems can be composed into a variety of architectures, and numerous protocols and additional systems can be plugged in at the discretion of the service provider. SIP can be considered a building block, whereas H.323 is a specific system.

H.323	ITU developed H.323. Version 1 standardized in 1996. Focus was multimedia communications services for LANs without QoS. H.323 v.1 not targeted for IP specifically, but any type of packet LAN. Version 2, released in 1998. Version 3 has been recently completed, and Version 4 is now under development.
SIP	IETF, origins in late 1996 as a component of the Mbone set of utilities and protocols. Focus on distribution of multimedia content, including talks and seminars, broadcasts of space shuttle launches, and IETF meetings. mechanism for inviting users to listen in on an ongoing or
H.323	complete, vertically integrated suite of protocols architecture for delivering multimedia conferencing applications. includes signaling, registration, admission control, security, interworking requirements with H.320, H.321, and other ITU conferencing systems, inter-domain data exchange, transport, and codecs. defines several entities, including terminals (end systems, like PCs), gateways, multipoint conferencing units, and something called a gatekeeper. A gatekeeper is similar to a SIP proxy, in that it plays the role of a signaling relay.
SIP	single component, works with e.g. RTP but does not mandate it SIP systems can be composed into a variety of architectures, and numerous protocols and additional systems can be plugged in at the discretion of the service provide
H.323	LAN protocol; numerous enhancements (such as FastStart) added to gain behaviour as a wide-area protocol
SIP	designed as a wide-area protocol, no enhancements needed.

H.323	borrows its call-signaling component from existing work done in ITU, namely the Q.931 protocol, used for user-to-network signaling in ISDN => telephony-centric flavor
SIP	borrows much of its concepts from HTTP => web flavor allows to integrate with web, e-mail, and other existing IP applications. KISS (Keep It Simple Stupid) principle => easier to implement and interoperate
H.323	extendable by add non-standard elements identified by vendor ID and version change => backward compatible, takes up more room than its predecessor
SIP	extended in numerous ways: including adding headers, new methods, new bodies, and parameters to existing headers
H.323	H.245 contains powerful mechanisms for conference control for distributed multiparty conferences. (deny - grant speaking privileges)
SIP	kind of control possible within SIP-established conference, but not addressed by SIP itself, currently no standalone standard protocols that can do this

---

Connectivity:

SIP transparently supports name mapping and redirection services, allowing the implementation of ISDN and Intelligent Network telephony subscriber services. The phone identifier is to be used when connecting to a telephony gateway. Even without this parameter, recipients of SIP URLs MAY interpret the pre-@ part as a phone number if local restrictions on the name space for user name allow it.

Main Advantages:

Services:

SIP transparently supports name mapping and redirection services, allowing the implementation of ISDN and Intelligent Network telephony subscriber services. These facilities also enable personal mobility. In the parlance of telecommunications intelligent network services, this is defined as: "Personal mobility is the ability of end users to originate and receive calls and access subscribed telecommunication

Internet telephony began on the premise that it was cheaper than normal phone calling. Users were willing to tolerate degraded quality or reduced function for lower cost. However, the cost differentials are rapidly disappearing. To continue to exist, Internet telephony must find another reason to be. The answer is services.
Some of the most exciting applications have already found killer status on the Internet, though not (yet) in

the form of multimedia services. Now think of integrating multimedia communications, such as voice, with web, e-mail, buddy lists, instant messaging, and online games. Whole new sets of features, services, and applications become conceivable.

SIP is ideally suited here. Its use of URLs, its support for MIME and carriage of arbitrary content (SIP can carry images, MP3s, even Java applets), and its usage of e-mail routing mechanisms, means that it can integrate well with these other applications. For example, it is just as easy to redirect a user to another phone as it is to redirect a user to a web page.

Scalability:
SIP uses the Internet model for scalability - fast and simple in the core, smarter with less volume in the periphery. To accomplish this, SIP defines several types of proxy servers. Call-stateful proxies generally live at the edge of the network. These proxies track call state, and can provide rich sets of services based on this knowledge. Closer to the core, transaction-stateful (also known as just stateful) proxies track requests and responses, but have no knowledge of session or call state. Once a session invitation is accepted, the proxy forgets about it. When the session termination arrives, the proxy forwards it without needing to know about the session.
Finally, stateless proxies exist in the core. These proxies receive requests, like INVITE, forward them, and immediately forget. The SIP protocol provides facilities to ensure that the response can be correctly routed back to the caller. Stateless proxies are very fast, but can provide few services. Call-stateful proxies are not as fast, but they live at the periphery, where call volumes are lower.

Extensibility:
History has taught Internet engineers that protocols get extended and used in ways they never intended (e-mail and web are both excellent examples of this). So, they've learned to design in support for extensibility from the outset. SIP has numerous mechanisms to support extensions. It does not require everyone to implement the extensions. Facilities are provided that allow two parties to determine the common set of capabilities, so that a session initiation can always be completed, no matter what.

Flexibility:
SIP is not a complete system for Internet telephony. It does not dictate architecture, usage patterns, or deployment scenario. It does not mandate how many servers there are, how they are connected, or where they reside. This leaves operators tremendous flexibility in how the protocol is used and deployed. One way to think of it is that SIP is a LEGO block; operators can piece together a complete solution by obtaining other LEGO blocks, and putting them together in the way that they see fit.

Multimedia:
Besides the traditional call-forwarding, follow-me, and do-not-disturb, SIP has the potential for enabling a whole new class of services that integrate multimedia with web, e-mail, instant messaging, and presence (meant here as, are you currently online?). The value that the Internet brings to Internet telephony is the suite of existing applications that can be merged with voice and video communications. As an example, at the end of a call, a user can transfer the other party to a web page instead of another phone. This transfer would end the call, and cause the other party's web browser to jump to the new page. In essence, the value of VoIP and SIP comes not from integration at the network layer (i.e., run your voice services on top of your data network), but at the services layer (i.e., combine your voice services with your data services)

Main Drawbacks:

Emerging issues in the Internet could ruin the promise of SIP (as well as H.323) over the long term. The problem is the increasing shortage of IP v4 numbers and the growing use of network address translators (NATs). There are similar issues when running SIP and H.323 through firewalls.
NATs break many protocols that act as establishment mechanisms for other protocols, such as SIP. NATs provide a boundary between the private IP addressing of a network and the public Internet. They aremost often used if an enterprise is unable to secure access to a sufficient block of IP numbers from their ISP, or if the enterprise wants the presumed luxury of being able to switch ISPs without having to renumber their network.
SIP, fundamentally, is a control channel for establishing other sessions (namely, the media sessions). These kinds of protocols (of which FTP and H.323 are other examples) cause problems for NATs, since the addresses for the established sessions are in the body of the application layer messages, as we see in the session description protocol examples shown in the sidebar, SIP Call Flow Examples.

When used with SDP, SIP messages carry the IP addresses and ports to be used for the media sessions. There may be multiple media sessions within a particular SIP call. Since SDP carries IP addresses and not host names, the external caller user agent will send media to an IP address that is not globally routable. It is only a valid IP address within the private network.

A nearly identical problem exists for firewalls. When a user inside the firewall sends media to an address outside the firewall, it will be dropped by the firewall unless a rule is established to allow it to pass. Since the media is sent on dynamic ports to dynamic addresses, these rules must be dynamically installed through application-aware devices, such as proxies.

Interfacing:

Developing services, of course, requires APIs. What kind of APIs are used to program services delivered by SIP? There has been significant activity in this area, resulting in numerous new interfaces, each with its own distinct set of strengths and weaknesses.

The first API that surfaced is the call processing language (CPL). CPL is not actually an API, but rather an XML-based scripting language for describing call services. It is not a complete programming language, either. It has primitives for making decisions based on call properties, such as time-of-day, caller, called party, and priority, and then taking actions, such as forwarding calls, rejecting calls, redirecting calls, and sending e-mail. CPL is engineered for end-user service creation.
A server can easily parse and validate a CPL, guarding against malicious behavior. The running time and resource requirements of a CPL can also be computed automatically from the CPL. An interpreter for CPL is very lightweight, allowing CPL services to execute very quickly. For these reasons, it is possible for an end user to write a CPL (typically with some kind of GUI tool), upload it to the network, and have it instantly verified and instantiated in real time.

At the opposite end of the spectrum in SIP is CGI (the common gateway interface). Many web designers are familiar with HTTP CGI; it's an interface that allows people to generate dynamic web content using Perl, Tcl, or any other programming language of choice. Since HTTP and SIP are so similar, it was recognized that an almost identical interface could be used for SIP. The result is SIP CGI, which is roughly 90% equivalent to HTTP CGI. Like HTTP CGI, SIP CGI passes message parameters through environment variables to a script that runs in a separate process. The process sends instructions back to the server through its standard output file descriptor. The benefit of SIP CGI is that it makes development of SIP services work much like the creation of dynamic web content. In fact, for SIP services that contain substantial web components, development will closely mirror web-only services. The importance of leveraging web tools for voice service creation is that a much larger class of developers becomes available.

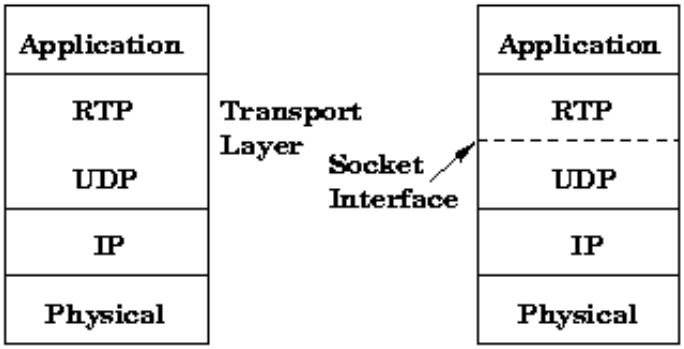
CGI has substantially more flexibility than CPL (CGI doesn't even mandate a particular programming language), but is much more risky to execute. Furthermore, because of its usage of separate processes, SIP CGI doesn't scale as well as CPL. Somewhere in the middle are SIP Servlets. HTTP Servlets are in wide use for developing dynamic web content. Servlets are very similar to the CGI concept. However, instead of using a separate process, messages are passed to a class that runs within a JVM (Java Virtual Machine) inside of the server. As a result, Servlets are restricted to Java, but suffer less overhead than SIP CGI. Use of a JVM for executing servlets means that the Java sandbox concept can be applied to protect the server from the script. Like SIP CGI, SIP Servlets closely mirror the operation of HTTP Servlets; they simply enhance the interface to support the wider array of functions a proxy can execute, as compared to an HTTP origin server

---

Real Time Transport Protocol

IETF audio-video transport group started to develop RTP in 1993. The aim of the protocol was to provide services required by interactive multimedia conferences, such as play-out synchronization, demultiplexing, media identification and active party identification. However, not only multimedia conferencing applications can benefit from RTP, but also storage of continuous data, interactive media distribution, distributed simulation, and control applications can utilize RTP

RTP consists of a data and a control part. The latter is called RTCP. Implementation will often be integrated into application rather than being implemented as a separate protocol layer. In applications RTP is typically run on top of UDP to make use of its port numbers and checksums. The RTP framework is relatively "loose" allowing modifications and tailoring depending on application. Additionally, a complete specification for a particular application will require a payload format and profile specification. The payload format defines how a particular payload is to be carried in RTP. A payload specification defines how a set of payload type codecs are mapped into payload formats.



RTP session setup consists of defining a pair of destination transport addresses one IP address and UDP port pair, one for RTP and another for RTCP. In the case of multicast conference the IP address is a class D multicast address. In multimedia session each medium is carried in a separate RTP session with its own RTCP packets reporting only the quality of that session. Usually additional media are allocated in additional port pairs and only one multicast address is used for the conference.

RTP has important properties of a transport protocol: it runs on end systems, it provides demultiplexing. It differs from transport protocols like TCP in that it (currently) does not offer any form of reliability or a protocol-defined flow/congestion control. However, it provides the necessary hooks for adding reliability, where appropriate, and flow/congestion control. (application-level framing). As lower layers are required to transfer data RTP is not really real time, but provides functionality suited for carrying real-time content, e.g., a timestamp and control mechanisms for synchronizing different streams with timing properties.

QoS:

Perhaps the most vexing problem in voice-over-IP, in general, has been the issue of quality of service. The delay in conversations that many VoIP users encounter is caused by the jitter and latency of packet delivery within the Internet itself. It’s useful to review some of the basic principles of the Internet to understand what can be done about the problem, what the IETF’s response has been, and how it impacts SIP.

Currently, the Internet offers a single service, traditionally referred to as best effort. In other words, all packets are created equal. There is no difference to the Internet whether a packet is e-mail, FTP, or the download of a web page. If the Internet gets very busy, packets get dropped or delayed. Unfortunately, the human ear is extremely sensitive to latency in the delivery of sound. The human ear can detect delays of 200 milliseconds or greater in voice conversations.

SIP itself does not get involved in reservation of network resources or admission control. This is because SIP messages may not even run over the same networks that the voice packets traverse. The complete independence of the SIP path and the voice path enables ASPs to provide voice services without providing network connectivity. This is an extremely important advantage of the SIP architecture. Given this, SIP

relies on other protocols and techniques in order to provide quality of service.

Encryption:

SIP requests and responses can contain sensitive information about the communication patterns and communication content of individuals. The SIP message body MAY also contain encryption keys for the session itself. SIP supports three complementary forms of encryption to protect privacy:

End-to-end encryption of the SIP message body and certain sensitive header fields;
hop-by-hop encryption to prevent eavesdropping that tracks who is calling whom;
hop-by-hop encryption of Via fields to hide the route a request has taken.

Not all of the SIP request or response can be encrypted end-to-end because header fields such as To and Via need to be visible to proxies so that the SIP request can be routed correctly. Hop-by-hop encryption encrypts the entire SIP request or response on the wire so that packet sniffers or other eavesdroppers cannot see who is calling whom. Hop-by-hop encryption can also encrypt requests and responses that have been end-to-end encrypted. Note that proxies can still see who is calling whom, and this information is also deducible by performing a network traffic analysis, so this provides a very limited but still worthwhile degree of protection.

SIP Via fields are used to route a response back along the path taken by the request and to prevent infinite request loops. However, the information given by them can also provide useful information to an attacker. End-to-end encryption relies on keys shared by the two user agents involved in the request. Typically, the message is sent encrypted with the public key of the recipient, so that only that recipient can read the message. All implementations should support PGP-based encryption and may implement other schemes.

A SIP request (or response) is end-to-end encrypted by splitting the message to be sent into a part to be encrypted and a short header that will remain in the clear. Some parts of the SIP message, namely the request line, the response line and certain header fields need to be read and returned by proxies and thus MUST NOT be encrypted end-to-end. Possibly sensitive information that needs to be made available as plaintext include destination address (To) and the forwarding path (Via) of the call. The Authorization header field must remain in the clear if it contains a digital signature as the signature is generated after encryption, but MAY be encrypted if it contains "basic" or "digest" authentication. The From header field should normally remain in the clear, but MAY be encrypted if required, in which case some proxies MAY return a 401 (Unauthorized) status if they require a From field.

Privacy of SIP Responses

SIP requests can be sent securely using end-to-end encryption and authentication to a called user agent that sends an insecure response. This is allowed by the SIP security model, but is not a good idea. However, unless the correct behavior is explicit, it would not always be possible for the called user agent to infer what a reasonable behavior was. Thus when end-to-end encryption is used by the request originator, the encryption key to be used for the response should be specified in the request. If this were not done, it might be possible for the called user agent to incorrectly infer an appropriate key to use in the

response. Thus, to prevent key-guessing becoming an acceptable strategy, we specify that a called user agent receiving a request that does not specify a key to be used for the response should send that response unencrypted.

Any SIP header fields that were encrypted in a request should also be encrypted in an encrypted response. Contact response fields MAY be encrypted if the information they contain is sensitive, or MAY be left in the clear to permit proxies more scope for localized searches.

Known Security Problems

With either TCP or UDP, a denial of service attack exists by a rogue proxy sending 6xx responses. Although a client SHOULD choose to ignore such responses if it requested authentication, a proxy cannot do so. It is obliged to forward the 6xx response back to the client. The client can then ignore the response, but if it repeats the request it will probably reach the same rogue proxy again, and the process will repeat.

The example shows a message encrypted with ASCII-armored PGP that was generated by applying "pgp -ea" to the payload to be encrypted.

```
INVITE sip:watson@boston.bell-telephone.com SIP/2.0
  Via: SIP/2.0/UDP 169.130.12.5
  From: <sip:a.g.bell@bell-telephone.com>
  To: T. A. Watson <sip:watson@bell-telephone.com>
  Call-ID: 187602141351@worcester.bell-telephone.com
  Content-Length: 885
  Encryption: PGP version=2.6.2,encoding=ascii

hQEMAxkp5GPd+j5xAQf/ZDIfGD/PDOM1wayvwdQAKgGgjmZWe+MTy9NEX8O25Redh0/pyrd/+DV5C2BYs7yzSOSXaj1
C/tTK/4do6rtjhP8QA3vbDdVdaFciwEVAcuXsODxlNAVqyDi1RqFC28BJIvQ5KfEkPuACKTK7WlRSBc7vNPEA3nyqZGBT
whxRSbIRRuFEsHSVojdCam4htcqxGnFwD9sksqs6LIyCFaiTAhWtwcCaN437G7mUYzy2KLcAzPVGq1VQg83b99zPzIxRdlZ
+K7+bAnu8Rtu+ohOCMLV3TPXbyp+err1YiThCZHIuX9dOVj3CMjCP66RSHa/ea0wYTRRNYA/G+kdP8DSUcqYAAAE/hZ
PX6nFIqk7AVnf6IpWHUPTelNUJpzUp5Ou+q/5P7ZAsn+cSAuF2YWtVjCf+SQmBR13p2EYYWHoxlA2/GgKADYe4M3JSw
OtqwU8zUJF3FIfk7vsxmSqtUQrRQaiIhqNyG7KxJt4YjWnEjF5EWUIPhvyGFMJaeQXIyGRYZAYvKKklyAJcm29zLACxU5al
X4M25lHQd9FR9Zmq6JedwbWvia6cAIfsvlZ9JGocmQYF7pcuz5pnczqP+/yvRqFJtDGD/v3s++G2R+ViVYJOz/lxGUZaM4IWB
Cf+4DUjNanZM0oxAE28NjaIZ0rrldDQmO8V9FtPKdHxkqA5iJP+6vGOFti1Ak4kmEz0vM/Nsv7kkubTFhRl05OiJIGr9S1Uhen
lZv9l6RuXsOY/EwH2z8X9N4MhMyXEVuC9rt8/AUhmVQ===bOW+
```

Kinds of Servers:

Redirect Server
User Agent Server
Proxy Server
Proxying Requests

Proxying Responses
Stateless Proxy: Proxying Responses
Stateful Proxy: Receiving Requests
Stateful Proxy: Receiving ACKs
Stateful Proxy: Receiving Responses
Stateless, Non-Forking Proxy
Forking Proxy

Internet Resources:

http://www.cs.columbia.edu/sip/
http://www.sipforum.org/

Additional Facts:

UTF-8:

The UTF-8 encoding allows Unicode to be used in a convenient and backwards compatible way in environments that, like Unix, were designed entirely around ASCII. UTF-8 is the way in which Unicode is going to be used under Unix, Linux, and similar systems. -> ISO 10646
http://www.cl.cam.ac.uk/~mgk25/unicode.html

ISO 10646:

The international standard ISO 10646 defines the Universal Character Set (UCS). UCS is a superset of all other character set standards. It guarantees round-trip compatibility to other character sets. If you convert any text string to UCS and then back to the original encoding, then no information will be lost. UCS contains the characters required to represent practically all known languages. This includes not only the Latin, Greek, Cyrillic, Hebrew, Arabic, Armenian, and Georgian scripts, but also also Chinese, Japanese and Korean Han ideographs as well as scripts such as Hiragana, Katakana, Hangul, Devangari, Bengali, Gurmukhi, Gujarati, Oriya, Tamil, Telugu, Kannada, Malayalam, Thai, Lao, Khmer, Bopomofo, Tibetian, Runic, Ethiopic, Canadian Syllabics, Cherokee, Mongolian, Ogham, Myanmar, Sinhala, Thaana, Yi, and others. For scripts not yet covered, research on how to best encode them for computer usage is still going on and they will be added eventually. This includes not only Hieroglyphs and various Indo-European languages, but even some selected artistic scripts such as Tengwar, Cirth, and Klingon. UCS also covers a large number of graphical, typographical, mathematical and scientific symbols, including those provided by TeX, Postscript, APL, MS-DOS, MS-Windows, Macintosh, OCR fonts, as well as many word processing and publishing systems, and more are being added.

ISO 10646 defines formally a 31-bit character set. However, of this huge code space, so far characters have been assigned only to the first 65534 positions (0x0000 to 0xFFFD). This 16-bit subset of UCS is called the Basic Multilingual Plane (BMP) or Plane 0. The characters that are expected to be encoded outside the 16-bit BMP belong all to rather exotic scripts (e.g., Hieroglyphics) that are only used by specialists for historic and scientific purposes. Current plans suggest that there will never be characters assigned outside the 21-bit code space from 0x000000 to 0x10FFFF, which covers a bit over one million potential future characters. The ISO 10646-1 standard was first published in 1993 and defines the architecture of the character set and the content of the BMP. A second part ISO 10646-2 which defines characters encoded outside the BMP is under preparation, but it might take a few years until it is finished. New characters are still being added to the BMP on a continuous basis, but the existing characters will not be changed any more and are stable.

UDP:

User Datagram Protocol RFC 768:
User Datagram Protocol is a Connectionless protocol. It uses IP to send datagrams in a similar way to TCP, except that like IP, and unlike TCP, UDP does not care if the packets reach their destination. UDP is used in applications where it is not essential for 100% of the packets to arrive. This may sound strange, but often you dont need all the packets. You wouldnt use UDP to transmit a program, because if one single bit was wrong (let alone loosing a whole packet) the file would be absolutely useless. It is up to program designers to choose what method is most suitable. While TCP is safer, UDP is becoming more common. It especially favored for Streaming or Real-time applications. More recently, internet applications have used both UDP and TCP. TCP is used for the essential or Control data, while UDP is used for data for which losses are acceptable.

ABNF:

A SIP message is either a request from a client to a server, or a response from a server to a client. SIP header fields are similar to HTTP header fields in both syntax and semantics. In particular, SIP header fields follow the syntax for message-header as described below.

SIP-message = Request | Response

Request:

Request = Request-Line *(general-header | request-header | entity-header) CRLF [message-body]

Request-Line = Method SP Request-URI SP SIP-Version CRLF

Method = "INVITE" | "ACK" | "OPTIONS" | "BYE" | "CANCEL" | "REGISTER"

INVITE

The INVITE method indicates that the user or service is being invited to participate in a session. The message body contains a description of the session to which the callee is being invited. For two-party calls, the caller indicates the type of media it is able to receive and possibly the media it is willing to send as well as their parameters such as network destination. A success response must indicate in its message body which media the callee wishes to receive and may indicate the media the callee is going to send.

ACK

The ACK request confirms that the client has received a final response to an INVITE request. (ACK is used only with INVITE requests.) 2xx responses are acknowledged by client user agents, all other final responses by the first proxy or client user agent to receive the response. The Via is always initialized to the host that originates the ACK request, i.e., the client user agent after a 2xx response or the first proxy to receive a non-2xx final response. The ACK request is forwarded as the corresponding INVITE request, based on its Request-URI.
The ACK request MAY contain a message body with the final session description to be used by the callee. If the ACK message body is empty, the callee uses the session description in the INVITE request.

OPTIONS

The server is being queried as to its capabilities. A server that believes it can contact the user, such as a user agent where the user is logged in and has been recently active, may respond to this request with a capability set. A called user agent MAY return a status reflecting how it would have responded to an invitation, e.g., 600 (Busy). Such a server SHOULD return an Allow header field indicating the methods that it supports. Proxy and redirect servers simply forward the request without indicating their capabilities.

BYE

The user agent client uses BYE to indicate to the server that it wishes to release the call. A BYE request is forwarded like an INVITE request and may be issued by either caller or callee. A party to a call should issue a BYE request before releasing a call ("hanging up"). A party receiving a BYE request must cease transmitting media streams specifically directed at the party issuing the BYE request.

CANCEL

The CANCEL request cancels a pending request with the same Call-ID, To, From and CSeq (sequence number only) header field values, but does not affect a completed request. (A request is considered completed if the server has returned a final status response.)
A user agent client or proxy client may issue a CANCEL request at any time. A proxy, in particular, may choose to send a CANCEL to destinations that have not yet returned a final response after it has

received a 2xx or 6xx response for one or more of the parallel-search requests. A proxy that receives a CANCEL request forwards the request to all destinations with pending requests.
The Call-ID, To, the numeric part of CSeq and From headers in the CANCEL request are identical to those in the original request. This allows a CANCEL request to be matched with the request it cancels. However, to allow the client to distinguish responses to the CANCEL from those to the original request, the CSeq Method component is set to CANCEL. The Via header field is initialized to the proxy issuing the CANCEL request. (Thus, responses to this CANCEL request only reach the issuing proxy.)
Once a user agent server has received a CANCEL, it must not issue a 2xx response for the cancelled original request.

REGISTER

A client uses the REGISTER method to register the address listed in the To header field with a SIP server.
A user agent MAY register with a local server on startup by sending a REGISTER request to the well-known "all SIP servers" multicast address "sip.mcast.net" (224.0.1.75). This request SHOULD be scoped to ensure it is not forwarded beyond the boundaries of the administrative system. This MAY be done with either TTL or administrative scopes [25], depending on what is implemented in the network. SIP user agents MAY listen to that address and use it to become aware of the location of other local users [20]; however, they do not respond to the request. A user agent MAY also be configured with the address of a registrar server to which it sends a REGISTER request upon startup. Requests are processed in the order received. Clients SHOULD avoid sending a new registration (as opposed to a retransmission) until they have received the response from the server for the previous one.
The meaning of the REGISTER request-header fields is defined as follows. We define "address-of-record" as the SIP address that the registry knows the registrand, typically of the form "user@domain" rather than "user@host". In third-party registration, the entity issuing the request is different from the entity being registered.

To: The To header field contains the address-of-record whose registration is to be created or updated.
From: The From header field contains the address-of-record of the person responsible for the registration. For first-party registration, it is identical to the To header field value.

Response:

After receiving and interpreting a request message, the recipient responds with a SIP response message. The response message format is shown below:

Response = Status-Line *(general-header | response-header | entity-header) CRLF [message-body]

Status-Line = SIP-version SP Status-Code SP Reason-Phrase CRLF

Status-Code = Informational | Success | Redirection | Client-Error | Server-Error | Global-Failure | extension-code

extension-code = 3DIGIT
Reason-Phrase = *<TEXT-UTF8, excluding CR, LF>

1xx:

Informational -- request received, continuing to process the request;

2xx:

Success -- the action was successfully received, understood, and accepted;

3xx:

Redirection -- further action needs to be taken in order to complete the request;

4xx:

Client Error -- the request contains bad syntax or cannot be fulfilled at this server;

5xx:

Server Error -- the server failed to fulfill an apparently valid request;

6xx:

Global Failure -- the request cannot be fulfilled at any server.

SIP response codes are extensible. SIP applications are not required to understand the meaning of all registered response codes, though such understanding is obviously desirable. However, applications must understand the class of any response code, as indicated by the first digit, and treat any unrecognized response as being equivalent to the x00 response code of that class, with the exception that an unrecognized response must not be cached.

SIP-MESSAGE

Both Request and Response messages use the generic-message format of RFC 822 for transferring entities (the body of the message). Both types of messages consist of a start-line, one or more header fields (also known as "headers"), an empty line (i.e., a line with nothing preceding the carriage-return line-feed (CRLF)) indicating the end of the header fields, and an optional message-body.

generic-message	= start-line *message-header CRLF [message-body]
start-line	= Request-Line | Status-Line
message-header	= (general-header | request-header | response-header | entity-header)

general-header	= Accept | Accept-Encoding | Accept-Language | Call-ID | Contact | CSeq | Date | Encryption | Expires | From | Record-Route | Timestamp | To | Via
entity-header	= Content-Encoding | Content-Length | Content-Type
request-header	= Authorisation | Contact | Hide | Max-Forwards | Organization | Priority | Proxy-Authorisation | Proxy-Require | Route | Require | Response-Key | Subject | User-Agent
response-header	= Allow | Proxy-Authenticate | Retry-After | Server | Unsupported | Warning | WWW-Authenticate

SIP-URL

A SIP URL follows the guidelines of RFC 2396 and has the syntax shown below.It is described using Augmented Backus-Naur Form. Note that reserved characters have to be escaped and that the "set of characters reserved within any given URI component is defined by that component. In general, a character is reserved if the semantics of the URI changes if the character is replaced with its escaped US-ASCII encoding".

SIP-URL	= "sip:" [userinfo "@"] hostport url-parameters [headers]
userinfo	= user [":" password]
user	= *(unreserved | escaped | "&" | "=" | "+" | "$" | ",")
password	= *(unreserved | escaped | "&" | "=" | "+" | "$" | ",")
hostport	= host [":" port]
host	= hostname | IPv4address
hostname	= *(domainlabel ".") toplabel ["."]
domainlabel	= alphanum | alphanum *(alphanum | "-") alphanum
toplabel	= alpha | alpha *(alphanum | "-") alphanum
IPv4address	= 1*digit "." 1*digit "." 1*digit "." 1*digit
port	= *digit
url-parameters	= *(";" url-parameter)
url-parameter	= transport-param | user-param | method-param | ttl-param | maddr-param | other-param
ttl-param	= "ttl=" ttl

ttl	= 1*3DIGIT ; 0 to 255
transport-param	= "transport=" ("udp" | "tcp")
maddr-param	= "maddr=" host
user-param	= "user=" ("phone" | "ip")
method-param	= "method=" Method
tag-param	= "tag=" UUID
UUID	= 1*(hex | "-")
other-param	= (token | (token "=" (token | quoted-string)))
headers	= "?" header *("&" header)
header	= hname "=" hvalue
hname	= 1*uric
hvalue	= *uric
uric	= reserved | unreserved | escaped
reserved	= ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" | ","
digits	= 1*DIGIT
telephone-subscriber	= global-phone-number | local-phone-number
global-phone-number	= "+" 1*phonedigit [isdn-subaddress] [post-dial]
local-phone-number	= 1*(phonedigit | dtmf-digit | pause-character) [isdn-subaddress] [post-dial]
isdn-subaddress	= ";isub=" 1*phonedigit
post-dial	= ";postd=" 1*(phonedigit | dtmf-digit | pause-character)
phonedigit	= DIGIT | visual-separator
visual-separator	= "-" | "."
pause-character	= one-second-pause | wait-for-dial-tone
one-second-pause	= "p"
wait-for-dial-tone	= "w"
dtmf-digit	= "*" | "#" | "A" | "B" | "C" | "D"

used keywords