SIP 即时消息 RFC3428

目录

1、SIP 协议介绍

Internet 的许多应用都需要建立和管理一个会话，会话在这里的含义是在参与者之间的数据的交换。由于考虑到参与者的实际情况，这些应用的实现往往是很复杂的：参与者可能是在代理间移动，他们可能可以有多个名字，他们中间的通讯可能是基于不同的媒介（比如文本，多媒体，视频，音频等）－有时候是多种媒介一起交互。人们创造了无数种通讯协议应用于实时的多媒体会话数据比如声音，影像，或者文本。本 SIP（会话初始协议）和这些协议一样，同样允许使用 Internet 端点（用户代理）来寻找参与者并且允许建立一个可共享的会话描述。为了能够定位精确的会话参与者，并且也为了其他的目的，SIP 允许创建基础的 network hosts（叫做代理服务器），并且允许终端用户注册上去，发出会话邀请，或者发出其他请求。SIP 是一个轻形的，多用途的工具，可以用来创建，修改和终止会话，它独立运作于通讯协议之下，并且不依赖建立的会话类型。

2、SIP 协议功能概况

SIP 是一个应用层的控制协议，可以用来建立、修改、和终止多媒体会话（或者会议）例如 Internet 电话。SIP 也可以邀请参与者参加已经存在的会话，比如多方会议。媒体可以在一个已经存在的会话中方便的增加（或者删除）。SIP 透明地支持名字映射和重定向服务，这个用于支持个人移动业务－用户可以使用一个唯一的外部标志而不用关心他们的实际网络地点。SIP 在建立和维持终止多媒体会话协议上，支持 5 个方面：

用户定位： 检查终端用户的位置，用于通讯。

用户有效性：检查用户参与会话的意愿程度。

用户能力：检查媒体和媒体的参数。

建立会话：”ringing”,建立会话参数在呼叫方和被叫方。

会话管理：包括发送和终止会话，修改会话参数，激活服务等等。

SIP 不是一个垂直集成的通讯系统。SIP 可能叫做是一个部件更合适，它可以用作其他 IETF 协议的一个部分，用来构造完整的多媒体架构。比如，这些架构将会包含实时数据传输协议（RTP）（RFC 1889）用来传输实时的数据并且提供 QoS 反馈，实时流协议（RSTP）(RFC 2326)用于控制流媒体的的传输，媒体网关控制协议（MEGACO）(RFC 3015)用来控制到公共电话交换网（PSTN）的网关，还有会话描述协议（SDP）（RFC 2327）用于描述多媒体会话。因此，SIP 应该和其他的协议一起工作，才能提供完整的对终端用户的服务。虽然基本的 SIP 协议的功能组件并不依赖于这些协议。

SIP 本身并不提供服务。但是，SIP 提供了一个基础，可以用来实现不同的服务。比如，SIP 可以定位用户和传输一个封装好的对象到对方的当前位置。并且如果我们利用这点来通过 SDP 传输会话的描述，立刻，对方的用户代理可以得到这个会话的参数。如果我们用这个像传输会话描述（SESSION DESCRIPTION SD）一样呼叫方的照片，一个”呼叫 ID”服务很容易就建立了。这个简单的例子说明了，SIP 作为一个基础，可以在其上提供很多不同的服务。

SIP 并不提供会议控制服务（比如议席控制或者投票系统），并且并没有建议会议应该怎样管理。可以通过在 SIP 上建立其他的会议控制协议来发起一个会议。由于 SIP 可以管理参与会议的各方的会话，所以会议可以跨异构的网络，SIP 并不能也不打算提供任何形式的网络资源预留管理。

安全对于提供的服务来说特别重要。要达到理想的安全程度，SIP 提供了一套安全服务，包括防止拒绝服务，认证服务（用户到用户，代理到用户），完整性保证，加密和隐私服务。

SIP 可以基于 IPV4 也可以基于 IPV6

3、术语

在这个文档中，关键词”必须”,”不允许”,”要求”,”可以”,”不可以”,”应该”,”不应该”,”建议”,”不建议”,”可能”,”可选” 是根据 BCP14,RFC 2119[2]的规范描述 SIP 实现需要的不同层次

4、实施概览

这节通过简单的示例介绍了 SIP 的基本实现。本节是通过自然的而非正则的示例来介绍的。

第一个例子说明了 SIP 的基本功能：定位一个断点，发出通讯请求，通过协商会话参数建立会话，拆卸刚才建立的会话。

图一表示一个典型的 Alice 和 Bob 两个用户间的 SIP 消息交易交换例子.（每一个消息采用字母”F”和一个用来指向正文的一个数字做标记）在这个例子里，Alice 在她的 PC 上使用一个 SIP 的应用程序（比如说一个软的电话），呼叫 Bob 在 Internet 上的一个 SIP 电话。这个例子也掩饰了两个 SIP 代理之间，怎样为 Alice 和 Bob 建立会话连接。This typical arrangement is often referred to as the "SIP trapezoid" as shown by the geometric shape of the dotted lines in Figure 1.

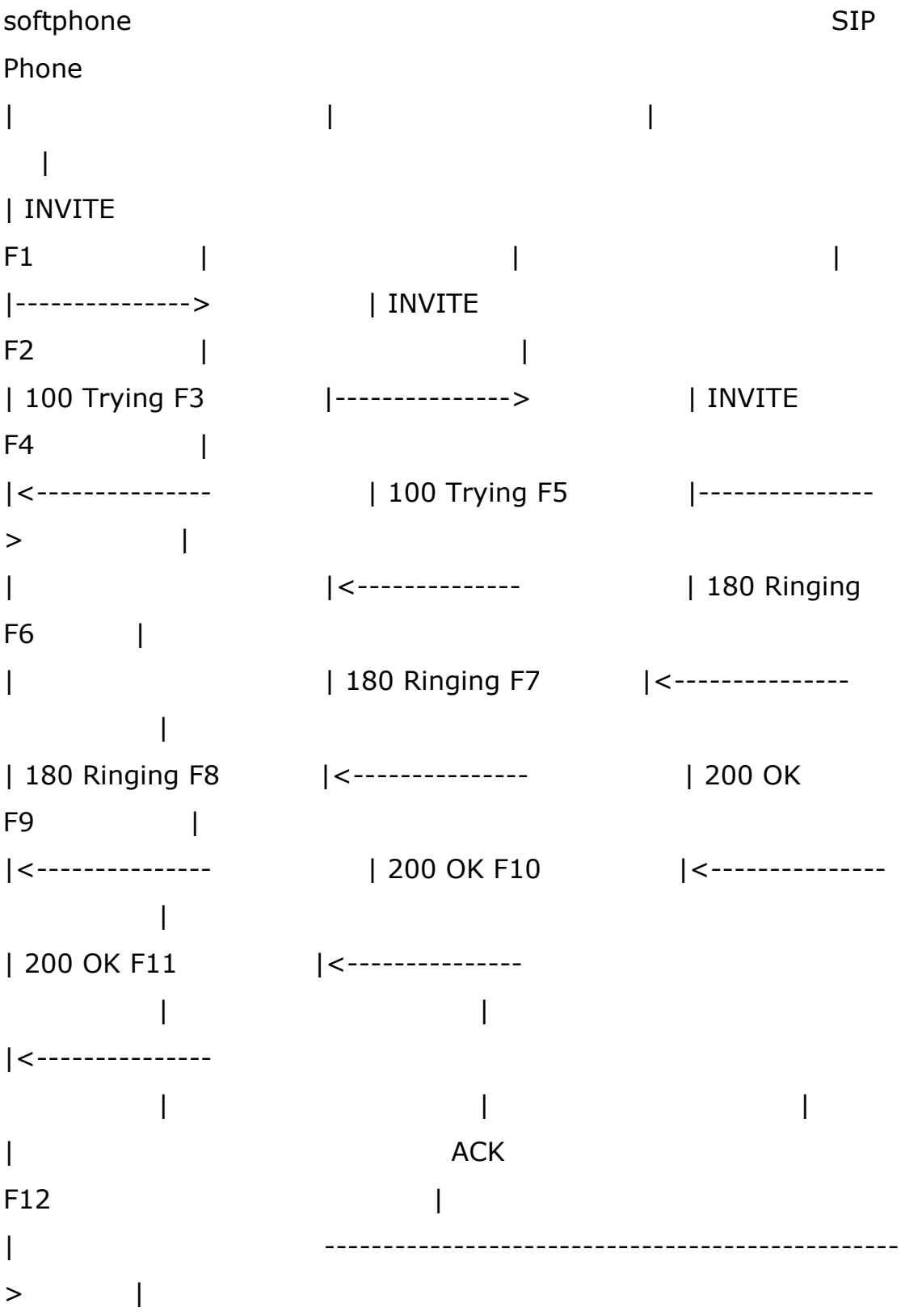
Alice 通过Bob的SIP标志 “呼叫” Bob，这个SIP标志是统一分配的资源（Uniform Resource Identifier URI）称作SIP URI。SIP URI在 19.1 节中定义。它很像一个email地址，典型的SIP URI包括一个用户名和一个主机名。在这个范例中，SIP URI是sip:bob@biloxi.com,biloxi.com是Bob的SIP服务提供商。Alice有一个SIP URI: sip:alice@atlanta.com。 Alice可以输入Bob的URI，也可以直接在地址本的一个超级链接上点击一下Bob的URI。SIP也提供保密URI，称作SIPS URI。例如：sips: bob@biloxi.com。一个基于SIPS URI的通话保证这个通话是安全的，并且对呼叫者和被叫的所有的SIP消息是加密传输的

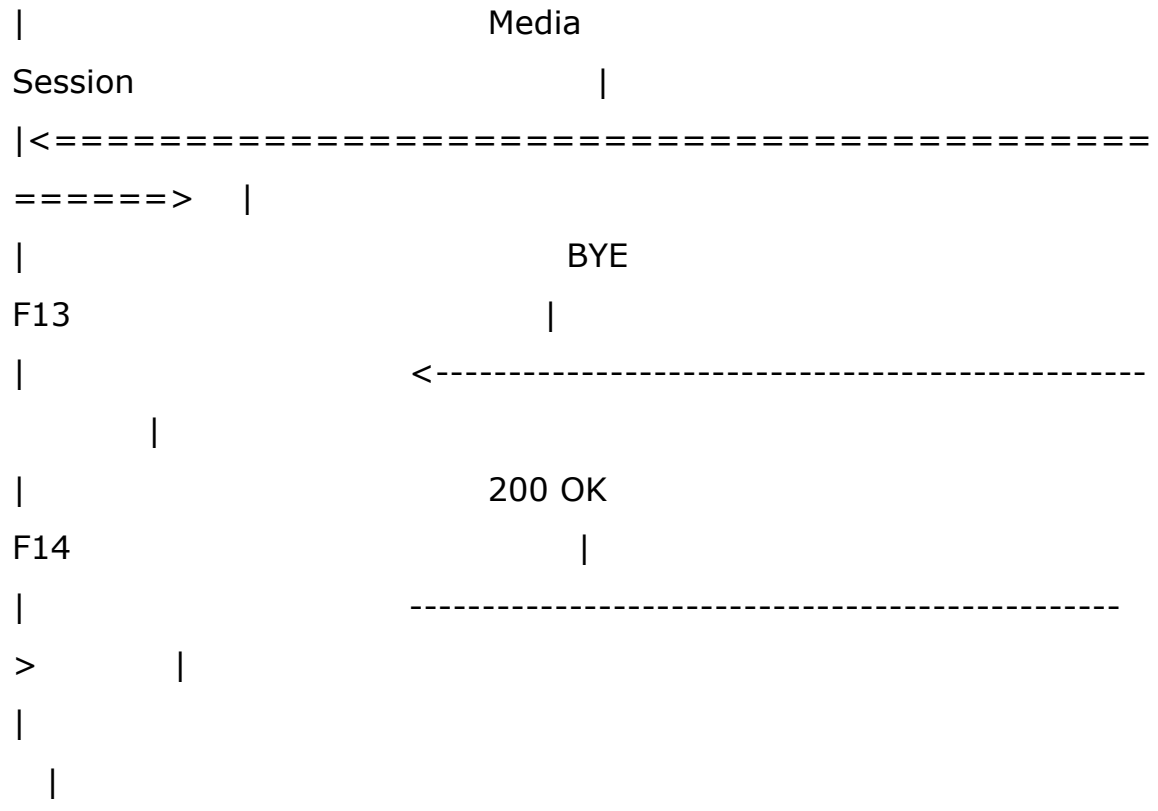
（叫做TLS）。在TLS中，请求是通过加密方式传输给被叫方，但是这个加密机制是基于被叫方宿主服务器的实现的。

SIP 是基于一个类似 HTTP 协议的请求应答的通讯模式。每一个通讯都包含对某个功能的请求，并且起码需要一个应答。在这个应答中，Alice 的软电话发送一个含有 Bob 的 SIP URI 抵制的 INVITE 通讯请求。INVITE 是一个 SIP 请求的例子，表示请求方（Alice）希望服务方（Bob）应答。INVTE 请求包含一系列的包头域（Header fields）。包头中包含很多属性并且包含了传输消息的附加信息。在 INVITE 中有如下的字段：呼叫的唯一标志，目的抵制，Alice 的地址，Alice 和 Bob 建立会话的类型。INVITE 请求（图 1 中的 F1）可能看起来像这样的：

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142
(Alice’s SDP not shown)

atlanta.com . . . biloxi.com
. proxy proxy .
. .
Alice’s . . . .
Bob’s


softphone
SIP Phone
INVITE F1
INVITE
100 Trying F3
INVITE F4
100 Trying F5
180 Ringing F6
180 Ringing F7
180 Ringing F8
200 OK F9
200 OK F10
200 OK F11
ACK F12



图一：SIP 矩形表达的 SIP 会话建立例子。

在文本消息的第一行，包含了请求的类型（INVITE）。在这行之后的是这个请求的头域。这个例子中包含了最少需要的头域集合。简单介绍一下：

VIA 域包含了 Alice 接收发送请求的服务器地址（pc33.atlanta.com）。同样这个包含了一个分支参数来标志 Alice 和这个服务器的会话事务。

TO 域包含了显示姓名（Bob）和一个 SIP 或者 SIPS URI（sip：bob@biloxi.com）请求将首先传输到这个 URI 中。显示姓名（Display names）在 RFC 2822 中描述。
From 域也同样包含一个显示姓名（Alice）和一个 SIP 或者 SIPS URI（sip:alice@atlanta.com）这个 URI 用来标志请求的原始发起者。

这个域也包含了一个 TAG 参数，这个 TAG 参数是一个随机字串（1928301774），是软电话（softphone）在 URI 上增加的一个随机串。用来做标志用途的。

Call_ID 包含一个全局的唯一标志，用来唯一标志这个呼叫，通过随机字串和 softphone 的自己名字或者 IP 抵制混和产生的。通过 TO TAG, FROM TAG 和 CALL-ID 完整定义了 Alice 和 Bob 之间的端到端的 SIP 关系，并且表示这个是一个对话性质的关系。

CSEQ 或者 Command Sequence 包含了一个整数和一个请求名字。这个 Cseq 数字是顺序递增的。每当对话中发起一个新的请求都会引起这个数字的顺序递增。

Contact 域包含一个 SIP 或者 SIPS URI 用来表示访问 Alice 的直接方式，通常由用户名和一个主机的全名（Fully Qualified Domain Name FQDN）组成。当 FQDN 作为首选的时候，许多终端用户由于不会由名字登记（而导致不能访问 Alice 的主机），所以 IP 地址是可选的。

VIA 域告诉大家本请求发送到哪里并且应答到哪里，Contract 域告诉大家将来的请求将发送到哪里（奇怪…不是 Alice 发起的么，将来的请求应该是 Bob 才对啊）。

Max-Forwards:最大转发数量限制了通讯中转发的数量。它是由一个整数组成，每转发一次，整数减一。

Content—type 包含了消息正文的描述（消息正文在本范例中没有列出）

Content-length:包含消息正文的长度（字节数）

完整的 SIP 包头域的定义在 20 节。会话的细节，比如媒体的类型，codec，或者采样速率，没有通过 SIP 来描述。这个可以通过 SIP 的消息正文来描述，可以通过其他定义的协议在正文中进行描述。有一种是会话描述协议（Session Descripotion Protocol SDP）（RFC2327[1]）。这个 SDP 消息（没有在例子中列出）通过 SIP 的消息中传送，就像通过附件发送 EMAIL 一样，或者说通过 HTTP 传输的网页一样。

由于 softphone 并不知道 bob 或者 bob 的 sip 服务器 biloxi.com 在哪里，所以 softphone 发送 INVITE 请求到 Alice 的 sip 服务器,atlanta.com。这个 atlanta.com SIP 服务器应该已经在 Alice 的 softphone 中配置了，或者可以通过 DHCP 获得。atlanta.com SIP 服务器是一台代理服务器。代理服务器接收 SIP 请求并且根据请求转发。在这个例子中，代理服务器接收到 INVITE 请求，并且回送一个 100（Trying）应答给 Alice 的 softphone。100（Trying）应答表示 INVITE 请求已经收到，并且代理服务器正在转发 INVITE 请求。SIP 的应答是通过一个三位数的数字表示的。SIP 应答同样包含 TO、FROM、Call-ID，CSEQ 和在 VIA 中的分支参数，这个参数使得 Alice 的 softphone 可以把请求和应答关联起来。atlanta.com 代理服务器收到 INVITE 请求之后，就去找 biloxi.com 可能通过 DNS 服务来找提供这个 biloxi.com 的 SIP 服务器。这在[4]中有描述。最后，转发 INVITE 请求到 biloxi.com 或者能到达 biloxi.com 的代理服务器。在转发请求之前，atlanta.com 代理服务器会在 via 头上增加一个一段包含自己抵制的值（INVITE 已经包含了 Alice 的的地址 VIA 域）。biloxi.com 代理服务器收到这个 INVITE 请求并且返回一个 100（Trying）应答给 atlanta.com 代理服务器标志这它已经收到这个请求并且正在处理这个请求。这个代理服务器通过查询数据库，通常叫做地址服务，这个服务中包含了 bob 的当前 ip 地址。（我们在下一节可以看到这个数据库是怎么回事）biloxi.com 代理服务增加另一段包含自己地址的 VIA 头域并且发送它到 bob 的 sip 电话。

Bob 的 SIP 电话接收到 INVITE 请求并且提醒 bob 有一个从 Alice 的呼入，这样 bob 可以决定是否响应这个呼入。这个意思就是：bob 的电话响了。bob 的 sip 电话发送一个 180（Ringing）回应，这个回应将通过两个代理服务器原路返回给 Alice。每一个代理服务器通过 via 头域决定该把这个应答发送给哪里，并且在发送之前把自己的地址从头上拿走。虽然 DNS 和定位服务在路由最初的 INVITE 请求，180（ringing）响应可以简单返回给发起者而不需要查找发起者在哪里，并且不需要在代理服务器保留状态，同时，每一个转发 INVITE 的代理也可以得到 INVITE 的每一个应答，这样的特性也非常有用。

当 Alice 的 softphone 收到 180（Ringing）应答的时候，它提示 Alice,可能是通过一个回铃音，或者屏幕上的一个消息提示。

在这个例子中，Bob 决定响应这个呼叫。当他拿起电话，他的 SIP 电话发送 200（OK）回应给发送者，表示这个电话已经接起来了。这个 200（OK）包含了一个消息体，这个消息体包含 SDP 媒体描述，这个媒体描述包含 Bob 希望和 Alice 建立何种媒体连接。同样，SDP 消息也是两段交换：Alice 发送一个给 Bob，Bob 发送一个回给 Alice。这个两段的交换提供基本的兼容性协商，并且基于简单的 SDP 提出/应答交换模型。如果 Bob 不想响应这个呼叫或者正在响应别的呼叫，一个错误的响应会代替正常的 200（OK）回送出去，这样，就不会有连接建立。SIP 完整的返回代码在 21 节有介绍。Bob 发出的 200（OK）（图一的 F9 消息）可能长得像这样的：

SIP/2.0 200 OK
Via: SIP/2.0/UDP server10.biloxi.com
;branch=z9hG4bKnashds8;received=192.0.2.3
Via: SIP/2.0/UDP bigbox3.site3.atlanta.com
;branch=z9hG4bK77ef4c2312983.1;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com
;branch=z9hG4bK776asdhds ;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710@pc33.atlanta.com
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131
(Bob’s SDP not shown)

应答的第一行包含了应答代码（200）和原因（ok）。剩下的行包含了包头域。VIA,TO,FROM,CALL-ID, Cseq 包头域是从 INVITE 请求包中直接拷贝过来的。（有三个 VIA 域值－一个是 Alice SIP 电话增加的，一个是 atlanta.com 代理加的，一个是 biloxi.com 代理加的）。Bob 的 SIP 电话增加了一个 TAG 参数。这个 TAG 参数会被参与对话的各方所使用，并且在以后的对话中被使用。Contract 域包含了一个能直接联系到 Bob 的 URI。Content－type 和 Content_Length 域包含了消息体（没有在例子中体现），这个消息体里边是 Bob 的 SDP 媒体信息。

除了 DNS 和位置服务之外，代理服务器可以自主决定路由，也就是说自己决定应该向哪里转发请求。比如，如果 Bob 的 SIP 电话返回一个 486（电话正忙）信号，biloxi.com 这个代理服务器可以转发这个 INVITE 请求到 Bob 的语音邮箱服务器。一个代理服务器可以同时向 N 个地方发送 INVITE 请求。这种并发寻找就是传说中的分流（forking）。

在这个例子中，200（OK）应答通过两个代理并且发送到 Alice 的 softphone 上，Alice 的 softphone 收到这个应答，停止振铃，并且标志电话 Bob 已经接听。最后，Alice 的电话发送一个确认消息，ACK，到 Bob 的 SIP 电话来确认接收到了这个最后的 200（ok）应答。在这个例子中，ACK 信号是直接由 Alice 的 softphone 发送到 Bob 的 SIP phone 上，跨过了两个代理服务器。这是因为两个端点（Alice 和 Bob）通过 INVITE/200(OK)的请求应答包中的 Contact 包头域都知道互相之间的地址了，这个地址是最开始发起 INVITE 请求的时候所不知道的。所以，不需要两个代理服务器再查找对方的地址了，所以代理服务器不参与接下来的通话流了。这就完成了一个完整的使用 INVITE/200/ACK 三方握手来建立 SIP 会话的过程。会话建立过程中的细节描述再 13 节由描述。

现在，Alice 和 Bob 的媒体会话开始了，他们通过发送刚才建立会话所交换的 SDP 包中约定的互相明白的媒体包来进行会话。一般情况下，端到端的媒体包和 SIP 信号控制包通过不同的通讯路径来发送。

在会话中，Alice 或者 Bob 都可以改变他们自己的媒体会话属性。这个可以通过发送一个包含新媒体属性描述的 re-INVITE 请求来完成。这个 re-INVITE 是捆绑在一个现有的会话的，这样参与会话的对方可以明白这是要改变现有的会话属性而不是新建立一个会话。对方收到这个 re-INVITE 请求后，会发送一个 200（OK）应答表示接受这个改变。请求方通过一个 ACK 来表示接受了对方的这个 200(OK)应答。如果对方不同意这个媒体属性变化，他会发送一个错误的应答比如 488（暂时不能进行），这个也会收到发起者的一个 ACK 响应。不管怎样，就是是 re-INVITE 的失败也不会影响到现有的会话－原有的会话还可以用上次的媒体会话属性继续。可以在 14 节找到会话属性更改的细节说明。

在通话结束的时候，Bob 首先断开（挂机 hangs up），并且发送一个 BYE 的消息。这个 BYE 的消息将直接送到 Alice 的 softphone，同样是跳过代理的。Alice 通过发送 200（OK）应答来确认收到了这个 BYE 消息，这个消息终止了会话并且应答了 BYE 的请求。ACK 在这里不需要发送－一个 ACK 信号只在响应一个 INVITE 的响应的时候被发送。我们稍晚一点会讨论这个 INVITE 的特别处理，但是基于 SIP 的可靠性的机制，一个通话的时间可以认为包含电话振铃和挂机的时间（but relate to the reliability mechanisms in SIP, the length of time it can take for a ringing phone to be answered, and forking.）基于这样的原因，SIP 请求的处理通常根据是否 INVITE 请求进行分类，INVITE 类和非 INVITE 类请求分开处理。结束会话的细节可以在 15 节查到。

24.2 节描述了图 1 中使用的全部消息详细解释。在某些情况下，所有会话中的包都继续通过代理转发会很有用。比如，如果 biloxi.com 代理服务器希望在 INVITE 之后继续保持 SIP 消息流，他会在 INVITE 中增加一个头域（Record-

Route）包含一个 URI 指向这个代理服务器的 hostname 或者 IP 地址。这个消息会被 Bob 的 SIP 电话和 Alice 的 softphone 所接到（因为 Record-Route 头域将在 200(OK)应答中被送回），并且在会话中一直保存。那么 biloxi.com 代理服务器就可以继续接收和转发 ACK,BYE,给 BYE 的 200（OK）应答。每一个代理都可以单独决定是否接收 INVITE 以后的后续消息，并且这些后续消息都可以被发送到那些决定接收后续消息的代理服务器。这种情况通常发生在提供 mid-call 业务的代理服务器上。

登记服务是另一个常用的 SIP 操作。登记服务是 biloxi.com 代理服务器知道 Bob 当前地址的一个方法。在初始化的时候，或者每隔一段时间，Bob 的 SIP 电话发送 REGISTER 消息给 biloxi.com 的一个注册服务器。REGISTER 消息包含了 Bob 当前登陆服务器的 SIP 或者 SIPS 的 URI（sip:bob@biloxi.com）（转换成为 Contact 域中的 SIP 或者 SIPS URI）。登记服务器登记这个映射，这个叫做绑定（binding），写到一个数据库里边，叫做定位服务（location service），这个数据库可以被 biloxi.com 的代理服务器使用。通常登记服务器和代理服务器是做在一起的。一个很重要的概念就是 SIP 服务器的差别在逻辑上，并非在物理上的差别。

Bob 并没有限定非得在一个单个设备上发起注册。比如，他家里的 SIP 电话和公司的 SIP 电话都可以注册。这些消息在定位服务（location service）中保存，并且允许代理服务器通过不同的手段查找 Bob。同样的，不同的用户也可以在同一个设备上同时注册。

定位服务（location service）是一个逻辑概念。他是让代理服务通过输入一个 URI 来查询到底应该向哪里转发请求。可以简单通过用户注册来建立这个定位服务所需要的资料，也可以通过其他方法。可以通过其他任意的地址映射方式来实现定位服务。

最后在 SIP 中需要注意的是，注册服务只是用来提供路由收到的 SIP 请求的，它并不做请求的身份认证的判定。在 SIP 中授权和认证可以通过建立在基于请求/应答的模式上的上下文相关的请求来实现，也可以使用更底层的方式来实现（具体在 26 节有描述）。

完整的注册 SIP 消息描述例子在 24.1 节。

其他 SIP 的操作，比如检查 SIP 服务器的负载，或者使用客户端使用可选项（OPTIONS），或者用 CANCEL 取消一个未决的请求，在后续的章节中会介绍。

5、协议的结构

SIP 是一个分层的协议，意思是说 SIP 协议由一组相对无关的处理层次组成，这些层次之间只有松散的关系。协议分为不同层次来描述是为了能够更清晰的表达，在同一个小节里有功能的公共要素的交叉描述。本协议并没有规定一个具体的实现。当我们说一个要素”包含”某一个层，我们的意思是这个要素符合这个层定义的规则。

不是 SIP 每一个要素都一定包含每一个层。此外，SIP 定义的要素是逻辑上的要素，不是物理要素。一个物理的实现可以实现不同的逻辑要素，或许甚至是基于串行事务处理原理。SIP 最底层的是它的语法和编码层。编码方式是采用扩展的 Backus-Naur Form grammar(BNF 范式)。完整的 BNF 描述在 25 节；第 7 节有简要的 SIP 消息结构描述。

第二层是传输层。它定义了一个客户端如何发送请求和接收应答，以及一个服务器如何接收请求和发送应答。所有的 SIP 要素都包含一个通讯层。第 18 节有通讯层的描述。

第三层是事务层。事务是 SIP 的基本组成部分。一个事务是客户发送的一个请求事务（通过通讯层）发送到一个服务器事务，连同服务器事务的所有的该请求的应答发送回客户端事务。事务层处理应用服务层的重发，匹配请求的应答，以及应用服务层的超时。任何一个用户代理客户端（user agent client UAC）完成的事情都是由一组事务构成的。有关事务的讨论在第 17 节有描述。用户代理包含一个事务层，来实现有状态的代理服务器。无状态的代理服务器并不包含事务层。事务层包含一个客户元素（可以认为是一个客户事务）和一个服务器元素（可以认为是一个服务器事务），他们都可以用一个有限状态机来处理特定的请求。

在事务层之上是事务用户（TU）。每一个 SIP 实体，除了无状态代理，都是一个事务用户。当一个 TU 发出一个请求，它首先创建一个客户事务实例（client transaction instance）并且和请求一起发送，这包括了目标 IP 地址、端口号、以及发送请求的设备。TU 可以创建客户事务，也可以取消客户事务。当客户取消一个事务，它请求服务器终止正在处理的事务，并且回归状态到该事务开始前的状态，并且产生指定的该事务的错误报告。这是由 CANCEL 请求完成的，这个请求有自己的事务，并且包含一个被取消的事务（第 9 节）。

SIP 要素，包含用户代理客户端和服务器，无状态和有状态代理服务器和注册服务器，包含一个可以互相区别的核心（Cores）。Cores，除了无状态代理服务器，都是事务用户。UAC(用户代理客户端)和 UAS(用户代理服务端)的 cores 的行为依赖于实现，对所有的实现来说，有几个公共的原则（第 8 节）。对 UAC 来说，这些规则约束请求的建立；对 UAS 来说，这些规则约束请求的处理和应答。由于注册服务在 SIP 中是一个重要的角色，所以 UAS 处理 REGISTER 请求有一个特别的名字：登记员（registrar，登记服务器）。第 10 节描述了 UAC 和 UAS 的对 REGISTER 实现的 core（核心）行为。第 11 节描述了 OPTIONS 的 UAC 和 UAS 的 core 实现，这个 OPTIONS 用来检测 UA 的处理能力的（UA-user agent）。

在对话中，有其他的相关会被发送。一个对话是一个持续一定时间的两个用户之间的端到端的 SIP 关系。对话过程要求两个用户代理之间的信息是有序的而且请求被正确路由传输的。在这个规范中，只有 INVITE 请求可以用来建立会话。当一个 UAC 在一个对话中发出请求的时候，它不仅遵循第 8 节描述的一般 UAC 规则而且也遵循对话中的请求规则。第 12 节讲述了对话并且讨论了对话的创建和维持，以及在对话中创建一个请求。

SIP 中最重要的方法就是 INVITE 方法，它用来在不同的参与者中创建会话使用。一个会话由一组参与者之间，用于交流的媒体流组成。第 13 节讲述了这些会话的创建初始化过程，以及如何创建一个或一组对话。第 14 节讲述了在对话中使用 INVITE 请求来改变会话的属性。最后，第 15 节，讲述了如何终止会话。

第 8、10、11、12、13、14、15 节讲述了完整的 UA 核心（第 9 节描述了取消，在 UA 核心和代理核心中使用）。第 16 节讲数了代理服务器，代理服务器用于在两个 UA 之间做消息路由使用。

6、协议的定义

以下讲述的名词对 SIP 有着额外的意义：

Address-of-Record: 记录地址。一个 address-of-record(AOR)是一个 SIP 或者 SIPS URI 它指向了一个具有定位服务的主机，这个主机可以把 URI 映射成为用户真正物理位置的 URI。通常情况下，定位服务器是通过登记服务来建立的。一个 AOR 经常被认为是一个用户的”公共地址”

Back-to-Back UserAgent: 背对背的用户代理（B2BUA）是一个逻辑实体，它就像用户代理服务器（UAS）一样接收和处理请求。为了决定该如何应答一个请求，B2BUA 就像 UAC 一样工作，并且发出请求。但是它不像代理服务器

（proxy），它需要维持对话状态，并且参与已经建立的对话中的每一个请求。由于它是直接的 UAC 和 UAS 的串连，所以，不需要对他有额外的定义。

Call：呼叫，一个呼叫是一个非正式的术语，它是指在端点之间一个一些通讯行为，通常用于建立多媒体对话。
Call Leg: 对话的别名；在本规范中没有使用。

Call Stateful: 如果一个代理服务器（proxy）保存一个对话的状态（从最开始的 INVITE 到对话终结的 BYE），那么这个代理服务器就是请求有状态的。一个请求有状态（call stateful）的代理服务器也一定是事务有状态的，但是事务有状态的不一定是请求有状态的。

Client：客户端。一个客户端是一个任意的网络元素，它发出 SIP 请求和接收 SIP 应答。客户端可能会也可能不会和人交互。用户代理客户端（UAC）和代理服务器(UAS)都是客户端。

Conference: 一个包含多个参与方的多媒体会话（见后）。

Core：核心。核心定义了 SIP 实体的特定类别。比如定义了一个有状态和无状态的代理服务器，一个用户代理或者注册服务器（registrar）。所有的核心，除了无状态代理服务器，都是事务用户。

Dialog: 对话，一个对话是持续一段时间的两个 UA 之间的端到端的 SIP 关系。一个对话由 SIP 消息建立，就像用 2xx 响应 INVITE 请求。我们用 Call identifier，local tag（本地 tag），remote tag（对方 tag）来标志一个对话，一个对话在 RFC 2543 中被正式叫做 CALL LEG.

Downstream: 它是事务中的消息传递方向。它特指从 UAC 到 UAS 的请求流的方向，

Final Response：终结响应。一个响应终端 SIP 事务的应答，和事务中间的临时响应相反。所有的 2xx，3xx，4xx，5xx，6xx 响应都是终结响应。

Header：头。头域是在 SIP 消息头部用来描述这个 SIP 消息信息的部分。它由一堆头域字段组成。

Header Field:头域字段。头域字段是在 SIP 消息头域的字段。一个头域字段可以由多个头域字段行组成。一个头域字段由一个头域名和（零个或多个）头域值组成。多个头域值用’,’分割。某些头域字段只能有单个值，比如结果域（result）就只能有一个值。

Header Field Value:头域值。一个头域值是一个单个的值，一个头域字段可以有 0 个或者多个头域值。

Home Domain：宿主机。一个提供 SIP 服务的主机。一般指的是在登记服务中指明的记录地址中的 URI 的主机。

Informational Response: 提示应答。和临时应答一样。
Initiator, Calling Party, Caller: 用 INVITE 初始一个会话（和对话）的那方。一个 caller 从发出 INVITE 请求建立对话开始，到对话终止都一直是这个角色。

Invitation: 一个 INVITE 请求。

Invitee, Invited User, Called Party, Callee:被叫方。收到 INVITE 请求并且建立会话的那方。一个被叫方从收到 INVITE 请求起，到终止 INVITE 建立的对话结束，都称作被叫方。

Location Service: 定位服务。定位服务是用来给 SIP 转发或者代理服务器确定被叫方可能的位置使用的。它包含一张绑定了 address-of-record 的表，被叫方可能有 0 到多个记录。绑定的记录可以通过多种渠道添加和删除；本规范定义了 REGISTER 方法来更新绑定表。

Loop：环路。当请求抵达一个代理服务器，代理服务器转发这个请求，当这个请求再次来到同一个代理服务器，就称之为环路。当第二次抵达的时候，Request－URI 中包含了上次抵达的资料，并且由于并没有什么东西可以改变转发的策略，这样就导致这个请求还会再次被转发回来。环路请求是错误的，所以，处理程序需要检测和防止协议中出现的环路请求。

Loose Routing: 丢失路由。代理服务器在下述情况下会丢失路由。
A proxy is said to be loose routing if it follows the procedures defined in this specification for processing of the Route header field. These procedures separate the destination of the request (present in the Request-URI) from the set of proxies that need to be visited along the way (present in the Route header field). A proxy compliant to these mechanisms is also known as a loose router.

Message: 消息。SIP 元素之间传送的协议数据就是消息。SIP 消息既可以是请求也可以是应答。

Method: 方法。方法是在服务器请求处理的主要功能。方法是请求消息自身携带的。典型的方法就是 INVITE 和 BYE。

Outbound Proxy: 对外代理服务器。一个代理服务器接收到客户的请求，即使它不是由 Request_URI 所决定的服务器。通常一个 UA 会手工配置一个对外的代理服务器，或者可以通过一个自动配置的协议自动配置一个。

Parallel Search: 并行搜索。并行搜索情况下，代理服务器会向多个用户可能存在的地方发起请求，并且等待应答。同串行搜索不同的地方是，并行搜索不会等待上一个请求应答回来之后再发起下一个搜索，而是一个接一个的发起搜索请求。

Provisional Response: 临时应答。服务器用来标志自己正在处理的应答，但是本应答并不结束一个 SIP 事务。1xx 应答就是临时的，其他应答标志着事务的结束。

Proxy, Proxy Server:代理、代理服务器。一个中间的实体。它本身即作为客户端也作为服务端，为其他客户端提供请求的转发服务。一个代理服务器首先提供的是路由服务，也就是说保证请求被发到更加”靠近”目标用户的地方。代理服务器对某些强制政策有用（比如，确认一个用户是否允许建立一个呼叫等）。一个代理服务器翻译，并且如果有需要的话，再转发前会重写请求消息。

Recursion：回路、递归。一个客户端，在响应请求的时候产生新的到 Contract 包头域的 URI 请求的时候，会在 3xx 响应中陷入递归。A client recurs on a 3xx response when it generates a new request to one or more of the URIs in the Contact header field in the response.

Redirect Server: 重定向服务器。一个重定向服务器是一个产生 3xx 应答的 UAS 服务器，指示客户端连接别的 URI。

Registrar: 登记员。一个登记员（登记服务器）是一个接收 REGISTER 请求得服务器。他把请求得信息放到定位服务器中，这样可以让定位服务器很方便得查找位置信息。

Regular Transaction: 常规事务。凡不包含 INVITE,ACK,或者 CANCEL 方法得事务就是常规事务。

Request: 请求。 一个由客户端发到服务端得 SIP 信息，用于执行特定得功能。

Response：应答。一个由服务端发到客户端得 SIP 信息。用来标志从客户端发往服务端得请求处理得情况得。

Ringback: 回铃音。回铃音是一个信号音。是给呼叫方得一个信号表示被叫方正在振铃（Ringing）。

Route Set: 路由集。路由集合是一个顺序得 SIP 或者 SIPS URI。这些 URI 描述了传递一个请求所必须经历得代理列表。一个路由集可以是自适应得，因为包头中包含了 Record-Route(记录路由)，也可以是依赖配置得到得。

Server：服务器。一个 server 是一个网络元素接收请求并且处理请求并且发送回应给请求方。典型的服务器就是代理服务器（proxies），用户代理服务器（user agent servers），重定向服务器，登记服务器。

Sequential Search：顺序查找。在顺序查找中，代理服务器顺序尝试联系地址，在处理下一个之前必须等待上一个请求已经有一个结束应答。一个 2xx 或者 6xx 系列得最终应答总是结束一个顺序查找。

Session：会话。根据 SDP 得描述：”一个多媒体会话是一个由多媒体发送方和接受方组成得集合，并且包括在发送方和接受方之间得数据流。一个多媒体会议是一个典型得多媒体会话。”(RFC 2327[1])(一个 session 在 SDP 订一下可以是一个或者多个 RTP session)。在定义中，一个被叫方可以被多次邀请，被不同的呼叫方邀请，到同一个会话。在 SDP 中，一个会话可以被 SDP 用户名，session id，网络类型，地址类型，地址元素得一个集合串所规定。

SIP 事务：一个 SIP 事务是在客户端和服务端得事件，包括了从第一个由客户端发送到服务端得请求，到最后一个（非 1xx）服务端向客户端发出得终结应答。如果请求是一个 INVITE 请求，并且终结应答是一个非 2xx 得应答，那么事务还包括一个 ACK 给服务器做应答。给 INVITE 请求的 2xx 应答的 ACK 回应，是一个独立的事务。

Spiral：回溯。一个回溯是指一个SIP请求，路由给一个proxy，并且转发，但是又被路由回这个proxy，但是不同于回路（递归）的是，这次路由回来的请求包的包头中，包含了不同于原请求的请求包部分，使得本次proxy决定的路由转发与上次不同。通常，这是说，请求的Request-URI不同于上次的Request_URI。一个回溯不是一个错误，不同于回路（环路loop）。通常导致这样的现象是呼叫转发（call forwarding）。一个用户呼叫joe@example.com。example.com代理服务器转发请求到Joe的PC,并且Joe的pc呼叫转移到bob@example.com。这个请求被转发回example.com代理服务器。可是这个并不是一个环路（loop）。因为请求的目的地址变成了另一个用户，这就是回溯，是一个合法的情况。

Stateful Proxy:有状态的代理服务器。在逻辑上，有状态的代理服务器就是处理一个请求的过程中，维持的一个本规范所定义的客户端和服务端的事务状态机。也是一个事务又状态代理服务器(transaction stateful proxy)。具体的 stateful proxy 在第 16 节定义。一个（事务）有状态代理服务器和一个 call stateful proxy 不是一回事。

Stateless Proxy：无状态的代理服务器。在逻辑上，无状态代理服务器在处理请求中，并不维持客户和服务端的事务状态机。一个无状态的代理服务器直接转发每一个接收到的请求和每一个接收到的响应。

Strict Routing: 严格路由。路由处理规则如果符合 RFC2543 协议（and many prior work in progress versions of this RFC） 就是一个严格路由。在这个规则下，如果在包头中包含 Route 域，那么代理服务器就会删除 Request_URI 域内容。本文档并不要求一定要有严格路由，本文档只要求松散路由就可以了。支持严格路由的代理服务器也叫严格路由器。

Target Refresh Request: 目标刷新请求。一个 Target Refresh Request 是一个在对话中发出的请求，用来更改对话目标的请求。

Transaction User(TU):事务用户。在 transaction 层之上的协议层。TU 包括了 UAC 核心，UAS core,和 proxy core。

Upstream: 上行流。一个在事务中的消息流向方向。它是指由用户代理服务器（UAS）发出应答到用户代理客户端（UAC）的消息流向方向。

URL-encoded: 一串根据 RFC2396－2.4 节编码的字符。

User Agent Client(UAC):用户代理客户端。用户代理客户端是一个逻辑的概念，他创建一个新请求，并且用客户事务状态机发送这个请求。UAC 角色只在事务中存在。换句话说，UAC 就是一小段代码初始化一个请求，并且在事务中遵循 UAC 的规则。如果它接下来收到一个请求，那么在那个事务中，它就是作为 UAS 来处理请求。

UAC Core: UAC 核心。在 transaction 和 transport 层之上得 UAC 实现的功能集合。

User Agent Server(UAS): 用户代理服务器.UAS 是一个逻辑的实体，对 SIP 请求做响应的。应答接受、拒绝、或者转发对应的请求。UAS 角色在事务中存在。换句话说，是响应请求的一小段软件，在事务中作为 UAS 存在。如果他发出请求，那么他就在事务中作为 UAC 存在。

UAS Core：UAS 核心。在 transaction 和 transport 层智商的 UAS 实现的功能集合。

User Agent（UA）。一个逻辑实体的概念，包含 UAC 和 UAS。
UAC 和 UAS，就像代理服务器和转发服务器，是在事务 by 事务的原理（串行事务处理）上定义的。例如，当发出一个初始化 INVITE 请求的时候，UA 作为 UAC 初始化一个呼叫动作，当从被叫方接收到一个 BYE 请求的时候，UA 作为 UAS 响应。类似的，同样的代码可以对一个请求做为 proxy 服务器处理，对另一个请求作为重定向服务器。

proxy，location，registrar 服务器都是逻辑实体，在它们的实现中，可能是作为单个应用实现的。

7、SIP 消息：

SIP 协议是一个基于文本的协议，使用 UTF-8 字符集（RFC2279[7]）。
一个 SIP 消息既可以是一个从客户端到服务器端的请求，也可以是一个从服务器端到客户端的一个应答。

即使在字符集上和语法细节上有所不同，请求（7.1）还是应答（7.2）消息都基于 RFC2822 格式的。（SIP 允许包头域不是标准的 RFC2822 包头域）。这两种消息类型都由一个起始行，一个或者多个包头域，一个可选的消息中文组成。

一般消息＝　　起始行
　　*消息包头
　　CRLF
　　[消息正文]
起始行＝　　请求行/状态行

起始行、每一个包头行，空行、都必须由回车换行组成（CRLF）。即使消息正文没有，也必须有一个空行跟随。
除了在字符集上的区别以外，很多 SIP 的消息和包头域的格式都同 HTTP/1.1 一样。我们在这里就不重复它的语法和语义了，我们用[HX.Y]来标志 HTTP/1.1 规范（RFC2616[8]）的 X.Y 节的描述。
SIP 并非一个 HTTP 的超集或者扩展。

7．1 请求

SIP 请求是根据起始行中的 Request-Line 来区分的。一个 Request_line 包含方法名字，Request－URI，用单个空格（SP）间隔开的协议版本。
Request-Line 由 CRLF 结束。除了用作行结束标志以外，不允许 CR 或者 LF 出现在其他地方。在其他域中，不允许出现线形的空白（liner whitespace LWS）

Request-Line =　　Method SP Request-URI SP SIP-VERSION CRLF
Method: 这个规范规定了 6 中方法：REGISTER 用于登记联系信息，INVITE，ACK, CANCEL 用于建立会话，BYE 用于结束会话，OPTIONS 用于查询服务器负载。SIP 扩展、标准 RFC 追加可能包含扩展的方法。

Request-URI: Request-URI 是一个 SIP 或者 SIPS URI，他们在 19.1 节由描述。也可以是一个通用的 URI(RFC 2396[5])。它标志了这个请求所用到的用户或者服务的地址。Request-URI 禁止包含空白字符或者控制字符，并且禁止用”<>”括上。

SIP 元素可以支持除了 SIP 或者 SIPS 之外所规定的 Request-URIs。比如”tel” URI 模式（RFC 2806[9]）。SIP 元素可以用他们自己的机制来转换 non-SIP URIs 到 SIP URI，SIPS URI 或者其他什么格式的 URI。

SIP-Version：请求和应答消息都包含当前使用的 SIP 版本，这个遵循[H3.1](类似 HTTP 用 SIP 替代，用 SIP/2.0 替代 HTTP/1.1)中关于版本的规定，版本依赖，升级版本号。一个应用，发出的 SIP 消息一定包含了 SIP-Version “SIP/2.0”。这个 SIP 版本串是大小写不铭感的，但是在实现中必须发送大写。不像 HTTP/1.1,SIP 把版本号当作一个文本串处理。在实现上，这个应该不会有区别。

7．2 应答

SIP 应答和 SIP 请求的区别在于在 START-LINE 中是否包含一个 STATUS-LINE。一个 status-line 在由数字表达的 status－code 之前，是一个协议的版本串，每一个元素之间用一个单个 SP（空格）分开。

除了最后用作结束标志以外，CR/LF 不允许出现在其他地方。

status-line = SIP-VERSION SP STATUS-CODE SP Reasong-Phrase CRLF

Status-Code 是一个 3 位的数字 result code，用来标志处理请求的一个结果。Reason-Phrase 是一个简短的 Status-Code 的说明。Status-Code 是为了能自动处理使用的，但是 Reason-Phrase 是用来给用户看得。一个客户端并不要求一定要显示或者解释这个 Reason-Phrase。本文档建议输出这个 reason-phrase, 实现中可以选择其他文本，比如用请求包头中指定的合适语言来显示。

status－code 的第一个数字表示了应答的类型。接下来两个数字并不作分类使用。基于这个原因，任何 status code 在 100 到 199 的可以统称位”1xx 应

答”，类似的，在 200 到 299 的可以统称位”2xx 应答”，依此类推。SIP/2.0 允许 6 类应答：

1xx：临时应答－请求已经接收，正在处理这个请求。

2xx：成功处理－请求已经成功接收，并且正确处理了这个请求。

3xx：重定向－还需要附加的操作才能完成这个请求，本请求转发到其他的服务器上处理。

4xx：客户端错误－－请求包含错误的格式或者不能在这个服务器上完成。

5xx：服务器错误－服务器不能正确的处理这个显然合法的请求。

6xx：全局错误－请求不能被任何服务器处理。

21 节定义了详细的代码说明。

7.3 头域

SIP 头域和 HTTP 头域在语法和语义上都比较类似。特别的，SIP 头域遵循[H4.2]关于消息头的语法的定义，并且遵循多行的扩展头域的规则。(后者 is specified in HTTP with implicit whitespace and folding)。这个规范遵循 RFC2234[10]，并且把清晰的空白和封装作为内在的语法规则。

[H4.2]也定义了具有相同域名的多个头域，他们的值是用逗号分开的列表，可以合并到一个头域中。这个也适用于 SIP，但是细节上略有不同，因为语法不同。实际上，任何 SIP 的包头语法都是基于如下范式的：

header ＝ ‟ header-name” HCOLON header-value *(COMMA header-value)

这个允许合并在具有同一个域名的多个头域，到一个用逗号分割的单个头域中。Contact 头域除了当域值是”*”之外，都允许用逗号分割的列表。

7.3.1 头域格式。

头域遵循在 RFC2822 的 2.2 节定义的通用头域格式。每一个头域都由一个域名加上冒号（”:”）和域值组成。

field-name:field-value

消息头的正则语法在 25 节中有介绍介绍。

在消息头中，允许在冒号的左右有任意个数的空白；但是，在实现中，建议避免域名和冒号中间有空格，并且建议在冒号和值之间使用单个空格（SP）。

Subject: lunch

Subject : lunch

Subject :lunch

Subject: lunch

所以，上面的都是合法的，也是相等的，但是最后一种是我们所推荐的。

头域可以扩展成为多行的，只要在每一个附加行前边用至少一个 SP 或者水平 TAB(HT)打头就可以了。这种多行的头域在行结尾并且在下一行之前的空白 SP（或者 HT）将被认为是一个单个的 SP 字符。所以，下边的例子是相等的：

Subject: I know you’re there, pick up the phone and talk to me!

Subject: I know you’re there,

pick up the phone,

and talk to me!

头域中的不同域名的相关顺序并没有什么意义。虽然如此，我们还是强烈建议与路由相关的域（VIA,ROUTE,Record-Route,Proxy-Require,Max-Forwards,Proxy-Authorization 等等）放在消息头的最前边，这样可以提高处理的速度。相同域名的域之间的顺序非常重要。只有当单个头域的域值是可以用逗号分割的列表的时候，才可以表达成为同一个域名的多个头域（这就是说，如果遵循 7.3 定义的语法）。也就是意味着必须可以将同一个域名的多个头域在不改变消息语义的前提下，改换表达成为一对”域名: 域值”；这个转换是通过顺序增加每一个域的域值，域值之间用逗号分割。这个规则有几个例外，就是 WWW-Authenticate, Authorization, Proxy-Authenticate,和 Proxy-Authorization

头域。多个头域行可以在消息头中出现，但是由于他们的语法并不遵循 7.3 中定义的通用格式，所以，他们并不能合并成为单个头域行。

在实现上，必须能够既能够处理多个头域行的情况，也必须能够处理用逗号分割的合并的单个头域行的情况。

下边的几组头域是相等的：

Route: <sip:alice@atlanta.com>
Subject: Lunch
Route: <sip:bob@biloxi.com>
Route: <sip:carol@chicago.com>

Route: <sip:alice@atlanta.com>, <sip:bob@biloxi.com>
Route: <sip:carol@chicago.com>
Subject: Lunch

Subject: Lunch
Route: <sip:alice@atlanta.com>, <sip:bob@biloxi.com>
<sip:carol@chicago.com>

下边各组是合法的，但是并不相等。

Route: <sip:alice@atlanta.com>
Route: <sip:bob@biloxi.com>
Route: <sip:carol@chicago.com>

Route: <sip:bob@biloxi.com>
Route: <sip:alice@atlanta.com>
Route: <sip:carol@chicago.com>

Route:
<sip:alice@atlanta.com>,<sip:carol@chicago.com>,<sip:bob@biloxi.com>

每一个头域值的格式是依赖于它的头域名的。他可以是任意顺序的 TEXT-UTF8 字符，也可以是一个空格，标记，分隔符，引号括起来的字串的组合。很多头域都回附带一个通用的域值格式。这个域值格式是由分号分开的参数名和参数值的组合：

field-name: field-value *(;parameter-name=parameter-value)

虽然在域值里边可以有任意数量的 parameter-name/parameter-value 对，但是不能允许有相同的 parameter-name 存在（唯一性）。除了特别指出的头域之外，头域中的域名、域值、parameter name parameter-value 都是大小写不敏感的。标记词始终是大小写不敏感的。除非有特别的指定，引号串的字符串是大小写敏感的。例如：

Contact: <sip:alice@atlanta.com>;expires=3600

和

CONTACT: <sip:alice@atlanta.com>; ExPiReS=3600

相同。

Content-Disposition: session;handling=optional

和

content-disposition: Session;HANDLING=OPTIONAL

相同。

下边的两个头域不相同：

Warning: 370 devnull “Choose a bigger pipe”

Warning: 370 devnull “CHOOSE A BIGGER PIPE”

7.3.2 头域分类。

有一些头域是仅仅在请求（或者应答）中有效的。这些头域叫做请求头域或者应答头域。如果消息中的头域与这个消息的类型不匹配（比如在应答消息中出现的请求头域），这个头域必须被忽略。20 节定义了每一个头域的分类。

7.3.3 缩写格式

SIP 提供了一个用缩写格式来表达通用头域名字的机制。这个有助于避免消息过大而导致通讯层无法传输（比如在 UDP 传输的时候超过了最大传输单元(MTU)）。这个缩写格式在 20 节定义。

缩写格式的消息头域名字可以在不改变消息语义的情况下替代较大的消息头域名字。在单个消息中，头域名字既可以用长的格式，也可以用缩写格式。在实现中，必须同时支持对长名字和缩写名字的处理。

7.4 包体

请求信息，包括这个规范以后的扩展的新请求，都可以包含一个消息正文体。对消息正文体的解释依赖域请求的方法（请求类型）。对于应答消息来说，请求方法和应答状态（response status code）决定了消息正文体的格式。所有的应答消息都可以有一个消息正文体（body）。

7.4.1 消息正文类型(MessageBodyType)

消息中的 internet 媒体类别必须在 Content-Type 头域中指明。如果消息正文（body）通过某种形式的编码（encoding），比如压缩等等，都必须在 Content-Encoding 头域中指明，否则 Content-Encoding 域必须忽略。如果可行，消息体的字符集作为 Content-type 头域的值的一部分表达。

在RFC2046[11]中定义的多部分”multipart” MIME类型可以在消息体中应用。在由多部分组成的消息体发送的时候，如果接受方的实现中，包头域的Accept域

中，不包含多部分的标记，那么发送方必须发送一个非多部分的session description。[1]

SIP 消息可以包含二进制的包体或者部分包体。如果发送方没有其他显示的字符集参数指出，媒体的文本”text”子类型会是缺省的字符集”UTF-8”。

7.4.2 消息体长度

在 Content-Length 头域中存放了包体的字节长度。第 20.14 节讲述了本域的详细解释。

HTTP/1.1 的“chunked”传输编码方式并不适用于 SIP。（备注：chuncked 编码传输方式是通过把消息正文体分为一系列的块来传输的，每一块有它自己的大小标记）

7.5 分帧的SIP 消息（Framing SIP Messages）

不同于 HTTP 的是，SIP 实现可以使用 UDP 或者其他非可靠传输协议。每一帧包括一个请求或者应答。第 18 节讲述了非可靠传输的应用。

在处理以面向流的通讯为基础的 SIP 消息的时候，必须忽略在开始行之前的 CRLF[H4.1]。

Content-Length 头域用来确定每一个 SIP 消息在通讯流中的结束位置的。在基于面向流通讯基础上的 SIP 消息一定要使用这个头域。

8 一般用户代理行为

一个用户代理代表了一个终端系统。它包含一个用户代理客户端（UAC）,用来产生请求的，它包含一个用户代理服务端（UAS），用来响应请求的。UAC 可以由一些外部的东西来发出请求和处理应答（比如用户按了一个按钮，或者按下了一个

电话键产生了一个音频信号等等）。UAS 是一个能够接收请求，并且产生应答的东西，它可以根据用户输入，外部输入，程序执行结果或者其他什么机制来产生应答。

当一个 UAC 发送一个请求，这些请求可能通过一些 PROXY（代理服务器）传递到 UAS 上。当 UAS 产生一个应答，那么这个应答就会同样的被传送到 UAC。UAC 和 UAS 的处理由两个特点。第一，基于请求或者应答是否在一个对话里，第二，基于请求的方法（method）。会话的彻底描述在第 12 节；哪里描述了点对点的用户代理之间的关系，并且通过一些 SIP 方法建立了会话，比如 INVITE 方法等。

在本节，我们将讨论在处理对话外的请求时，UAC 和 UAS 的方法无关的规则。这些当然也包括用于建立会话的请求。在 26 节讲述了对在对话外的请求和应答的安全处理。特别时，UAS 和 UAC 之间的互相认证的机制。通过用 S/MIME 加密的消息体可以提供有限的隐私保证。

8.1 UAC 特性

本节讲述 UAC 在会话外的特性。

8.1.1 产生一个请求

一个合法的 SIP 请求必须至少包含如下头域：TO, FROM, Cseq, Call-ID, Max-Forwards, Via;这些字段在所有 SIP 请求中必须包含。这 6 个字段是 SIP 消息的基本组成部分，他们提供了用于路由用的核心信息，包含了消息的地址，响应的路由，消息传递次数，详细的顺序，事务的唯一标志。
这些头域字段是必须包含在请求行之后的，请求行包含了请求的方法，Request-URI,SIP 的版本号码。

有两个在对话外的发送请求的示例（通过 INVITE 请求建立连接，第 13 节），（通过 OPTIONS 请求查询负载，第 11 节）。

8.1.1.1 Request-URI

最开始的 Request-URI 头域应该是 TO 头域的的值。但是在 REGISTER 方法中，有一个值得注意的不同；REGISTER 方法的 Request-URI 头域在第 10 节中指出。出于隐私的原因而把这些字段的值设置成为同一个值并不太合适（尤其是如果初始的 UA 期望 Request-URI 会在传输中改变的话）。

在一些特定的情况下，预先设置的路由表（route-set）会影响消息中的 Request-URI。一个预置路由表是由一串 server 的 URI 组成，这些服务器是 UAC 往外发送会话外请求所需要经过的。通常，他们是由用户或者服务提供商手工在 UA 上设置的，或者通过一些非 SIP 的方法自动设置。当一个提供商希望配置一个出口 proxy 给一个 UA，我们强烈建议通过一个预置一个单个 URI 路由表的方式来实现，这个单个路由就是出口 proxy。

当要使用预置路由表(route set)，必须提供 Request-URI 和 Route 头域（在 12.2.1.1 节中有详细描述）（甚至在没有对话存在的时候也必须提供），并且把 Request-URI 当作远端目标 URI。

8.1.1.2 TO

To 头域是第一个并且也是最先指定请求的”逻辑”接收地，或者是这个请求的用户或者资源的 address-of-record。这个域内的地址可以是也可以不是请求的最终接收者。TO 头域可以用 SIP 或者 SIPS URI,也可以用其他方式的 URI(比如电话 URL (RFC2806[9]))。所有的 SIP 实现必须支持 SIP URI 的实现。任何支持 TLS 的实现必须支持 SIPS URI 的实现。

To 头域允许有一个显示用的姓名。

UAC可以通过无数的方法来知道在一个给定请求的时候该如何填写TO头域。通常用户会建议采用人工界面中输入的To头域，可能手工输入这个URI或者从地址本

中选择（就好像outlook邮件中的to一样）。用户通常不会输入完整的URI,可能只是一个简单的字串（比如”bob”）。这就要求UA能够判断用户输入的这个到底是那个URI。一般使用用户输入的字串加上”@”标志和主机的名字组合成为SIP比如sip:bob@example.com)。如果希望通讯在保密机制下进行，那么就用用户输入的字串组成SIPS URI的部分，用户输入的将加上”@”和主机的名字作为整个SIPS URI。这个主机的名字通常是请求方的主机名字，这个主机允许处理外发请求。这个很像”缩位拨号”的机制，这个机制要求请求者自身的主机能够解释这个缩位拨号一样。 URI(

如果 UA 不希望指定主机，那么就需要将用户输入的电话号码解释成为一个电话的 URL。相当于，每一个请求经过的主机都会有机会来处理这个请求。比如，一个用户在机场可能登陆机场的代理服务器，并且通过机场的代理服务器发出一个请求。如果他输入”411”(美国本地电话本查询服务号码)，这个就需要机场的外发的代理服务器进行解释和处理，而不是解释成有主机的用户。在这里，tel:411 是一个正确的解释。

在会话外的请求中，不能包含 To tag 字段，在 to 头域中的 tag 是用来在对话中做标志的。既然对话还没有建立，那么 tag 就不能存在。

20.39 节有进一步的描述。

下边这个例子是一个 To 头域的例子：

To: Carol <sip:carol@chicago.com>

8.1.1.3 From

From 头域包含了请求发起者的逻辑标志，可能是用户的 address-of-record。就像 To 头域一样，From 头域也包含一个 URI 并且可以包含一个显示的姓名。SIP 可以用这个头域来实现对请求的检查和选择一个规则进行对请求的处理（比如，自动的呼叫拒绝，凡是 x 人发过来的东西，一律无视）。同样的，因为 From 头域包含的是逻辑名字，所以 From URI 也可以不包含 IP 地址或者 UA 对应的主机名字 FQDN。

From 头域可以包含一个显示姓名。在客户身份隐藏的情况下，一个 UAC 应该使用显示名字”Anonymous”，连通一个语法正确，但是没有意义的 URI(比如：

sip:thisis@anonymous.invalid)。通常，用户或者用户的本地主机的管理人员会事先规定请求头域中的 From 头域的值。如果给定的 UA 是多个用户共同使用的，那么必须有一个 URI 对应身份的 profile，这样才能够切用户的 profile。收到请求的服务方可以根据这个用于分辩身份的 URI 来区分同一个 UA 上的不同的用户，并且根据他们的 From 头域来判定他们的身份。（22 节有更多的验证说明）。

From 域必须包含一个由 UAC 产生的新的"tag"参数。19.3 节有 tag 的详细描述。20.20 节有更深入的资料。

例子：

From: "Bob" <sips:bob@biloxi.com> ; tag=a48s

From: sip:+12125551212@phone2net.com;tag=887s

From: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8

8.1.1.4 Call-ID

Call-ID 是一个在一系列消息中，区分一组消息的唯一标志。在对话中的任一 UA 的所有请求和所有应答的 Call-ID 必须一致。在 UA 的每次注册中，都应该是一样的。在会话外的时候，UAC 发起一个新的请求，这个 Call-ID 头域必须由 UAC 产生一个全局（在时间和空间上都是）唯一的 Call-ID, 除非是请求头的方法（method）指明了别的产生方式。所有的 SIP UA 都必须保证自己产生的 Call-ID 不会和其他 UA 产生的 Call-ID 重复。注意，如果是请求的重新尝试，则重新尝试的请求不被当作一个新的请求，所以不需要新的 Call-ID（重新尝试的请求例如：认证冲突等等）。（见 8.1.3.5）

我们强烈建议用密码乱序随机串（RFC 1750[12]）来产生Call-ID。实现中，可以用类似"localid@host"这样的格式产生。Call-ID是大小写敏感的，并且通过简单字节/字节的来进行比较。

采用密码乱序随机串可以降低会话被窃听的机会，并且降低 Call-ID 重复的冲突。

不规定或者要求使用用户界面来选择输入 Call-ID 头域的值。参见 20.8 节。

例子：

Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@foo.bar.com

8.1.1.5 Cseq

Cseq 头域是用来区分和做位事务的顺序使用的。他由一个方法（method）和一系列的顺序号码组成。方法（method）必须和请求的方法一致。对于对话外的非 REGISTER 请求来说，顺序号码可以是任意的。这个顺序号码必须可以由 32 位的无符号整数表达，必须小于 2^31。只要遵循了上述指导方针，客户端可以用任意的方法来产生这个 Cseq 头域。12.2.1.1 节讲述了在对话中如何创建 Cseq

例子：

Cseq: 4711 INVITE

8.1.1.6 Max-Forwards

Max-Forwards 头域用来限制请求到他的目的地中间的跳转。它包含一个每隔一个跳转就自动减一的数字。如果 Max-Forwards 在到达目的之前就减到 0，他会报告一个 483(太多的路由)错误回应。

一个 UAC 必须为每一个请求填写一个 Max-Forwards 头域，这个字段的缺省值应该是 70。这个数字是保证了请求在没有环路的 SIP 网络中都能够送达，也保证了在有环路的时候，尽量少消耗 proxy 的资源。如果这个数字要变小，则必须保证 UA 知道整个网络的拓扑结构。

8.1.1.7 Via

Via 头域是标志了用于事务传输的传输设备，并且也标志了应答送回的地址。只有当需要通过选择传输设备到达下一个节点(hop)的时候，才需要在头域中包含 Via 域。当 UAC 创建一个请求，它必须在头域中添加一个 Via 域。protocol 名字和 protocol 版本必须分别是 SIP 和 2.0。Via 头域必须包含一个分支(branch)参

数。这个参数用于区分请求创建的事务。这个参数客户端和服务器都会使用。除了CANCEL 和给非 2xx 应答的 ACK 以外，branch 参数对 UA 发出的所有的请求来说，在时间和空间上必须唯一。在下边的讲解中，CANCEL 请求的 branch 参数必须和他所取消的请求的 branch 参数一致。在 17.1.1.3 节中讲述了给非 2xxx（non－2xx）应答的 ACK 必须和其对应的的 INVITE 请求有相同的 branch ID。

利用 branch ID 参数的唯一性来作为事务的 ID（transaction ID）,并非 RFC 2543 的一部分。根据本标准产生的 branch ID 必须用”z9h64bK”开头。这 7 个字母是一个乱数 cookie（定义成为 7 位的是为了保证旧版本的 RFC2543 实现不会产生这样的值），这样服务器收到请求之后，可以很方便的知道这个 branch ID 是否由本规范所产生的（就是说，全局唯一的）。在这样的要求下，精确的 branch ID 的格式必须事先有实现的定义。

Via 头域中，maddr,ttl,和 sent-by 字段会在 transport 层处理请求的时候设置（18 节）。Via 在 proxy 的处理在 16.6 节 8 段和 16.7 节 3 段描述。

8.1.1.8 Contact

Contact 头域提供了访问后续请求的特定 UA 实例的联系方法：SIP 或者 SIPS URI。在任何会建立一个对话的请求中，Contact 头域必须提供和包含一个 SIP 或者 SIPS URI。在这个规范中定义的方法中，只有 INVITE 请求会建立一个会话。对这些请求来说，Contact 的作用域是全局性的。这就是说，Contact 头域中包含的 URI 是 UA 能够接收请求的，这个 URI 必须是有效的，甚至在对话外的请求中的 Contact 头域中的 URI 也必须是有效的。

如果 Request-URI 或者头上的 Route 头域包含了 SIPS URI,Contact 头域也必须是一个 SIPS URI。在 20.10 节有更进一步的说明。

8.1.1.9 Supported 和 Require

如果 UAC 支持服务端响应请求的 SIP 扩展，UAC 应该在请求的时候包含一个 Supported 头域说明 options tags（19.2 节）描述那些 SIP 扩展。option tags 中出现的扩展说明必须是遵循 RFCs 的标准扩展说明。这样可以防止服务端支持非标准的客户端扩展实现。在 Support 头域中的对于 SIP 扩展定义中，严格遵守不支持试验性质的或者说明性质的 RFCs 扩展，这个也是由于这些扩展是描述提供商定义的扩展说明。如果 UAC 要求 UAS 能够支持扩展，以便 UAS 能够处理 UAC 的特定请求，那么它必须在请求头中增加一个 Require 头域来说明处理本特定请求需要什么样的一个扩展 option tags。如果 UAC 需要请求经过的所有 proxy 都支持它发出的某个请求的扩展部分，它必须增加一个 Proxy-Require 头域来说明需要 Proxy 支持何种 option tag 扩展。

如同在 Supported 头域指出的，Require 和 Proxy-Require 头域中的 option 字段必须限定于 RFCs 的标准扩展。

8.1.1.10 附加信息部分

在一个新请求创建以后，以上的头域都被正确初始化了以后，就可以位这个请求增加它所需要的附加头域了。SIP 请求允许包含一个 MIME-encoded 消息正文。无论请求包含哪种消息正文，都必须引入头域来指出这个正文的类型，以及这个正文的一些其他说明。关于这些头域的详细说明，请参见 20.11 节到 20.15 节。

8.1.2 发送一个请求

于是，我们就开始查找请求发送的目标。除非有其他的特定说明，目标必须是通过 DNS 来查找的（参见[4]说明）。如果路由表(route set)中的第一个元素表明这是一个严格路由（strict router,在 12.2.1.1 节中讲述），那么这些过程必须在请求的 Request-URI 中说明。否则，这些过程在请求中被应用于第一个 Route 头域中（如果存在），或者在请求的 Request-URI 中（如果 Route 头域不存

在）。这样一些过程产生了一系列的地址，端口，和用于传输的传输器。无论那个 URI 用在这个[4]中描述的过程的输入，如果 Request-URI 指明了 SIPS，那么 UAC 必须按照[4]中描述的说明来认为输入的 URI 是 SIPS 的 URI。

本地策略可以指定一套额外的目的地用于发送。如果 Request-URI 包含一个 SIPS URI,任何额外的目的地都必须用 TLS 来表达。除此之外，如果请求没有包含 Route 头域，那么就没有对额外的目的地有什么其他的限制了。这个就提供了一个简单的外发（outbound）proxy 的事前路由的选择。但是，用这样的方法配置一个外发 proxy 是不推荐的；应该由单个 UPI 规定的预先设定的路由集来指定外发 proxy。如果请求包含了 Route 头域，请求应该发送到 Route 头域最上边的一个位置，但是请求也可能被发给由本文档约定的 Route 或者 Request-URI 所指定的服务器(同 RFC2543 定义的相反)。特别的，一个配置了外发 proxy 的 UAC 应该首先尝试把请求发送给由第一个 Route 头域值指定的位置，而不是采用把所有消息都发给外发 proxy 的策略。这就保证了外发的 proxy 通过不增加 Record-Route 头域而不参与后续请求的路径。这个也允许让不能分析第一个 Route URI 的终端，把请求交给外发 proxy 来发送。UAC 应该遵循[4]中定义的过程来实现有状态的元素，尝试每一个地址直到连接到一个服务器。每一个尝试都是一个事务，因此，每一个都有一个不同的 Via 头域值和一个新的 branch 参数值。

此外，在 Via 头域中的 transport 的值被设置成为要到目标服务器所必须的 transport。

8.1.3 处理应答

应答首先是被 transport 层（传输层）处理，并且被 transport 层发送给上一层 transaction 层（事务层）处理。transaction 层处理完成之后将应答发送给更上一层 TU 处理。在 TU 层进行的对应答的主要处理是方法相关的。但是也有集中通用的处理原则：

8.1.3.1: transaction 层的错误

在某些情况下，从 transaction 层返回的应答不一定是一个 SIP 消息，而是一个 transaction 层的错误。当从 transaction 层收到一个 timeout 错误的时候，必须将这个 timeout 错误当作是收到了一个状态码是 408（请求 timeout）的应答。如果 transport 层报告了一个严重错误（通常取决于 UDP 传输中的严重的 ICMP 错误，或者是 TCP 连接中的错误），必须把这个错误当作是状态码 503（服务未提供）的错误。

8.1.3.2 未知的应答

UAC 必须把自己不认识的所有最终应答当作是 x00 的那类应答，当然 UAC 也必须能够处理所有的类别应答的 x00 的应答。比如，如果 UAC 收到了不认识的应答代码 431，他可以很安全的假设在他发出的请求中有什么地方弄错了，并且可以很简单的把这个应答错误当作是接收到了一个应答代码是 400（非法请求）的错误应答。并且，UAC 必须能够处理所有的不认识的非终结应答响应当作是 183(session progress)。一个 UAC 必须能够处理 100 和 183 应答。

8.1.3.3 Vias

如果在应答中，有不只一个 Via 头域值存在，那么 UAC 应该丢弃这个消息。包含超过一个 Via 头域值的消息是因为被错误的路由或者消息被破坏。

8.1.3.4 处理 3xx 应答

由于接收到一个重定向的应答（比如，状态码是 301 的应答），客户端应该用在 Contact 头域中的 URI(s)来组织一个或者多个基于重定向以后的新请求，这个处理过程同 proxy 处理一个 3xx 类别的应答很类似，相关资料在 16.5 节和 16.6 节中有描述。客户发起请求的时候只有一个目标 URI，就是原始请求中的

Request-URI。如果客户端想在这个请求基础上重构一个基于 3xx 类别应答的新请求，那么就需要把这个需要尝试的 URIs 放到目标集合中去。基于本规范的规定，一个客户端可以选择放置那个 Contact URIs 到目标集合中（target set）。同 proxy 会递归一样，客户端处理 3xx 应答的时候必须不能重复添加任何 URI 到 target set。如果原始请求的 Request-URI 头域中包含了一个 SIPS URI，那么客户端可以选择改乘一个非 SIPS URI，但是需要知会客户转发到一个非安全的 URI 去。任何新的请求都可能导致接收到这些请求的 3xx 应答，并且在 Contact 中包含原始的 URI。可以通过对两个位置的配置来形成互相重定向。只要保证在 target set 中任何 URI 都只出现 1 次就能避免无穷的重定向循环。当 target set 增长了，客户端可以对这个 URIs，用任意顺序来产生新的请求。常见机制是通过在 Contact 头域的值中设置”q”参数来指定这个顺序。对这些 URIs 的请求可以是并行产生的也可以是串行产生的。有一个实现是按照 q 参数值递减的方法顺序处理 URIs，并且对相同 q 参数值的 URIs 进行并行处理。还有一种就是直接按照顺序的方法处理 URIs，对于 q 参数值不同的按照递减的顺序处理，对于 q 参数值相同的按照随机顺序处理。

如果发送给连接表上的地址失败了，（在下一段我们有定义），那么就选择列表中的下一个地址进行发送，直到列表全部遍历一遍。如果列表遍历完了还没有，那么请求就失败了。

通过响应码（大于 399）我们可以知道请求的失败；对于网络错误来说，客户 transaction 会给 transaciton user 报告 transport 层的通讯错误错误。注意有一部分响应码（8.1.3.5）表示请求可以被重试；这个时候，请求可以重发而不是简单的当作一个错误。如果某个 contact 地址发送失败，那么 client 应该尝试下一个 contact 地址。这个会导致创建一个新的客户事务来处理这个新的请求。

为了在处理 3xx 应答中创建一个基于一个 contact 地址的请求，UAC 必须首先从 target set 中拷贝除了”method-param”和”header”URI 参数之外的整个 URI 到 Request-URI（见参数的定义见 19.1.1）。通过使用”header”参数来创建新请求的头域值，按照 19.1.5 节的指引，根据重定向的请求来重写头域的值。

注意在某些情况下，在 contact 地址中进行通讯所需要的头域，可能代替添加到现有的在原始转发的请求的请求头域中。作为通用的规则，如果头域可以接受用逗号分割的值列表，那么新的头域值可以添加到原始转发的请求的任何值后边。如果请求头域不接收多值列表，那么原始转发请求中的头域值可以由 contact 地址中进行通讯所需要的头域的值所替换。比如，如果 contract 地址返回了如下的值：

sip:user@host?Subject=foo&Call-Info=Http://www.foo.com

那么在原始转发请求中的任何 Subject 头域都被重写，但是 HTTP URL 仅仅添加到现存的 Call-info 头域值中。

我们推荐 UAC 重用与原始转发请求相同的 To,From,和 Call-ID 域值，但是也允许 UAC 为新的请求改变 Call-ID 头域。

最后，当一个新的请求构造好以后，这个请求将通过一个新的客户 transaction 发送，应此在最开始的 Via 头域中必须有一个新的 branch ID （8.1.1.7 节说明）

在其他的方面，转发的请求应该重用原始请求的头域和消息体。

在某些情况下，Contact 头域的值可能在 UAC 中暂时或者永久保存，这个依赖于接收到的请求码和过期的时间；参见 21.3.2 和 21.3.3 节。

8.1.3.5 处理 4xx 应答

某个 4xx 应答码要求特定的 UA 处理，和请求的方法无关。

当接收到 401（未授权）或者 407（Proxy 认证需要）应答的时候，UAC 应该遵循在 22.2 和 22.3 中规定的认证步骤，重新发送带认证信息的请求。

当收到 413（请求过大）应答的时候（21.4.11 节），这说明请求包含了一个 UAS 所不能接收长度的消息体。如果可能，UAC 应该尝试重新 retry 这个请求，或者去掉包体或者换一个小一点的长度。

如果收到了一个 415（不支持的媒体类型）应答（21.4.13 节），那么请求中包含的媒体类别是 UAS 所不支持的。UAC 应该重发这个请求，并且这次发出的请求

只包含应答中的 Accept 头域所指明的媒体类别，并且采用 Accept-Encoding 头域中指明的 encoding 方法，还有 Accept-Language 指明的语言。
如果收到了一个 416（不支持的 URI Scheme）应答（21.4.14 节），Request-URI 使用的 URI Scheme（方案）是服务端所不支持的。客户端应该重新尝试这个请求，并且换用 SIP URI。
如果收到了一个 420（非法扩展）应答（21.4.15 节），请求的 Require 或者 Proxy-Require 头域包含的 option-tag 中包含了 UAS 或者 proxy 不支持的特性。UAC 应该尝试去掉应答中的 Unsupported 头域中列出的扩展以后然后再尝试。
在上述所有的情况中，所有需要重试的请求，都需要经过适当修正成为一个新的请求。这个新请求采用一个新的 transaction 并且应该有和上次请求相同的 Call-ID,TO,From 头域，Cseq 应该有一个新的顺序号码（比原有顺序号码更大）。
其他的 4xx 应答，包括尚未制定的，是否允许请求重试，依赖于具体的方法和应用。

8.2 UAS 特性

UAS 在处理对话外的请求的时候，有一组规则需要遵守，这组规则与方法无关。
12 节指明了一个方法来判定一个请求是否在一个对话里。
注意，请求的处理是原子级别的。如果请求被处理，那么这个请求的相关状态一定是一起更新的。如果它被拒绝了，那么这个请求的所有相关状态一定是没有改变的。
UASs 应当遵循本节所规定的顺序来处理请求。（就是说，首先是身份认证，然后是方法判定，然后是头域，然后按照本文规定处理剩余部分）

8.2.1 方法判定

当请求被认证（或者身份认证被忽略），UAS 必须首先判定这个请求的方法。如果 UAS 发现自己不能处理这个请求的方法的时候，它必须给出一个 405（方法不支持）的应答。产生应答的步骤在 8.2.6 节规定，并且 UAS 必须在给出的 405（方法不支持）应答中增加一个 Allow 头域。这个 Allow 头域必须列明哪些方法 UAS 支持。Allow 头域的说明在 20.5 节。

如果请求中的方法是服务器所支持的，那么处理将继续。

8.2.2 包头判断

如果 UAS 不认识请求中的包头域（就是说，包头域不在本规范中定义或者不在任何扩展中定义），那么服务器必须忽略掉这个包头域并且继续处理本请求。UAS 必须忽略任何处理本请求所不需要的长得畸形的包头域。

8.2.2.1 TO 和 Request-URI

To 头域包含了由 From 域描述的发送者发出的请求的原始接受者。原始接受者可能是也可能不是正在处理这个请求的 UAS,取决于呼叫转移或者其他的 proxy 操作。当 TO 域值和自身不相符的情况下，UAS 可以自行决定是否接收这个请求。但是，我们依旧是建议 UAS 处理这个请求，甚至 TO 这个头域是以他们不认识的 URI 方案表达的（比如一个 tel:URI），或者 To 头域并非指向这个自身处理的 UAS。当然，另外一方面来说，如果 UAS 决定拒绝这个请求，它应该产生一个 403（禁止访问）的状态码，并且交给服务器的 transaction 层来发送。

但是，Request-URI 确定 UAS 来处理这个请求。如果 Request-URI 使用了一个 UAS 所不支持的方案（比如 tel:URI），那么 UAS 应当拒绝这个请求，并且给出拒绝代码 416（不支持的 URI 方案）。如果 Request-URI 并没有指明本 UAS 来处理这个请求，那么 UAS 应当给出一个 404（未找到）的应答。比如，一个 UA 使用 REGISTER 方法来绑定它的 address-of-record 到一个特定的联系地址，将会收到 Request-URI 等于那个特定联系地址的请求。

其他潜在的 Request-URI 资源包括建立和刷新对话的 UA 发出的请求和应答的 Contact 头域。

8.2.2.2 合并的请求

如果请求的 To 头域中没有 tag 标志，UAS 的处理核心必须检查基于正在进行的 transactions 上的请求。如果接收到的请求和正在处理的 transaction 的请求中的头域 From tag,Call-ID,CSeq 精确匹配了，但是请求并不匹配那个事务（基于事务匹配机制 17.2.3 节），UAS 核心应该产生一个 482（检测到循环）应答并且给服务器的 transaction 层发送。

这是由于相同的请求通过不同的路径到达 UAS，很多情况下是由于分支的原因。UAS 处理了第一个请求并且给其他所有这个请求以 482（检测到循环）应答。

8.2.2.3 Require

如果请求的各项要素通过了 UAS 的判定，那么如果存在 Require 头域，接下来就是检查 Require 头域。Require 头域是 UAC 用来通知 UAS 应该用什么样的 SIP 扩展来处理本请求的。Require 的格式在 20.32 节中有介绍。如果 UAS 不支持请求的 Require 头域中的 option-tag 列表，那就必须产生一个 420 应答（错误的扩展）。并且 UAS 必须添上 Unsupported 头域，里边填上刚才接收到的请求的 Require 头域中，哪些 options 是自己所不支持的。

需要注意的是，Require 和 Proxy-Require 禁止出现在 SIP CANCEL 请求中，或者回应给非 2xx 应答的 ACK 请求中。就算出现了在处理的时候也必须被忽略。并且回应给 2xx 应答的 ACK 请求必须只能包含在初始请求（在这个 ACK 请求之前的请求）中包含的 Require 和 Proxy-Require 所规定 options，这样才能保证服务端能够正确处理。

例子：

UAC->UAS: INVITE sip:watson@bell-telephone.com SIP/2.0
Require: 100rel
UAS->UAC: SIP/2.0 420 Bad Extension
Unsupported: 100rel

这个特性（Unsupported）是为了保证客户－服务端都能够无阻碍的交互，除非是 options 对方不支持（就像上边的例子说明的一样）。对于相互匹配的客户－服务端（相互匹配的意思就是客户端 Require 的正好是服务端支持的），那么这些请求、应答将会处理的非常迅速，减少了一个请求的往返协商的浪费。另外，这个也避免了客户端不知道服务端到底不支持那些特性扩展。
某些特性扩展只对终端（endsystem）有效例如呼叫处理域等等。

8.2.3 内容处理

当 UAS 支持客户端请求中要求的扩展支持后，UAS 要检查消息头域描述的消息体部分。如果 UAS 并不支持消息体部分的类型（Content-Type 指明），语言（Content-Language 指明），编码（Content-Encoding 指明），并且这个消息体部分并非可选的消息体（Content-Disposition 头域指明）,UAS 必须回应一个 415 错误应答（不支持的媒体类型）应答。并且如果不支持请求中包含的消息体的正文类型，那么在应答中必须包含 UAS 所支持的消息体的类型列表（在 Accept 头域中指明）。如果不支持请求包含的消息体的 encoding 方式，那么应答中必须包含 Accept-Encoding 头域列明服务端支持的 encoding 方式。如果请求中的语言部分不支持，那么就必须在应答中包含 Accept-Language 头域列明支持的语言。在这些检查之外，消息体正文的处理依赖于方法和类型（method and type）。关于处理内容相关的头域的进一步的资料在 7.4、20.11 到 20.15 节。

8.2.4 应用扩展

如果 UAS 希望应用一部分 SIP 扩展，那么不可以在产生应答的时候做 SIP 扩展，除非这个扩展是在请求中的 Supported 头域中指明了的。如果这个扩展并没有在本请求的 Supported 头域中指明，那么服务端必须基于基准 SIP 给出应答，或者给出已知客户端支持的扩展应答。在极少数情况下，如果服务端不用扩展就无

法处理请求，那么服务端应该发送 421（需要扩展支持）应答。这个应答说明如果没有适当的扩展就无法给出正确的应答。在应答中需要的扩展必须在应答中的 Require 头域中指出。我们并不推荐这个方法，因为它会降低协同工作的能力。除了 421 应答之外的其他应答中，如果需要应用扩展，那么这些扩展需要在 421 的应答中的 Require 头域中列明。当然，服务端不允许使用没有在请求中的 Supported 头域中列明的扩展。因此，应答中的 Require 头域只会包含标准的 RFCs 的扩展 option tags。

8.2.5 处理请求

当所有的检查都通过了以后，UAS 根据方法决定如何处理请求。第 10 节讲述了 REGISTER 请求的处理，11 节讲述了 OPTIONS 请求的处理，13 节讲述了 INVITE 的处理，15 节讲述了 BYE 的处理。

8.2.6 产生应答

UAS 产生应答，需要遵守接下来的几个小节中讲述的步骤产生一个应答。
在本节没有描述的应答代码的其他的行为，也可能会跟据应答代码的不同而要求。
当创建应答的步骤完成之后，UAS 将应答交给收到这个请求的服务端的 transaction 去处理。

8.2.6.1 发送一个临时应答

很多情况下，在与方法无关的应答规范中，在非 INVITE 请求的情况下，我们都要求 UAS 不应该发送临时应答给请求者。在这种情况下，UAS 应该尽快发送一个终结应答给非 INVITE 请求。
当需要产生一个 100（Trying）应答的时候，所有对应请求中包头的 Timestamp 域必须也拷贝到这个应答包头（就是说如果请求中有 Timestamp，应答中也必须有 timestamp）。而且如果应答有延时，那么 UAS 应该在这个

Timestamp 上增加延时的值。这个数值必须包含了接收请求和发送应答的时间，用秒来计数。

8.2.6.2 包头和 Tags

应答中的 From 头域必须和请求中的 From 头域相等。应答中的 Call-ID 头域必须和请求中的 Call-ID 头域相等。应答中的 Cseq 头域必须和请求中的 Cseq 头域相等。应答中的 Via 头域必须和请求中的 Via 头域相等，而且顺序也必须相等。如果请求中包含了 To tag，那么应答中的 To 头域必须和请求中的 To 头域相等。如果请求中的 To 头域并不包含 Tag, 那么应答中的 To 头域的 URI 必须和请求中的 TO 头域的 URI 相等；此外，UAS 还必须增加一个 Tag 到 To 头域上（100（trying）应答是一个例外，在 100 中可能已经存在了一个 tag）。这就提供了一个 UAS 正在应答的标志，也许就是对话 ID 的一部分。对同一个请求来说，它的应答必须有相同的 tag 标志，包括终结应答和临时应答（同样 100（trying）除外）。生成 tag 的步骤在 19.3 节。

8.2.7 无状态 UAS 行为

无状态 UAS 就是说 UAS 本身不保持事务的状态。但是它在发送对应请求的应答之后并不保存请求的状态。如果无状态 UAS 接收到了一个重新发送的请求，它会重新产生一个应答并且重新发送应答，就像它接收到的是一个第一个请求一样（就像是新请求而不是重发的请求一样）。只有当相同请求的处理会导致相同的应答的时候，这个 UAS 才会是无状态的。例如：这条规则排除了无状态的登记服务。无状态的 UAS 并不包含一个 transaction 层；他们直接从通讯层接收请求和发送应答。

无状态的 UAS 通常用于处理不需要身份认证的请求，而且这些请求的应答是可以预期的。如果当不需要身份认证的请求被作为有状态的 UAS 来处理，那么大量的不需要身份认证的请求会造成服务器大量的事务状态，可能会导致 UAS 的处理非

常慢，甚至中断处理，这样就导致了 DoS（denial of service）状态。在 26.1.5 节中有进一步的描述。

无状态的 UAS 有下列重要的特性：

o 无状态的 UAS 不许发送临时应答（1xx）

o 无状态的 UAS 必须忽略 ACK 请求

o 无状态的 UAS 必须忽略 CANCEL 请求

o To 头域必须依据无状态的规则来产生－就是说对于相同的请求，产生相同的 tag 标记。19.3 节有讲 tag 标记。

对于其他情况而言，无状态的 UAS 遵循有状态的 UAS 的规则。对于一个新请求来说，同一个 UAS 可以是有状态的，也可以是无状态的。

8.3 重定向服务器

在某些架构下，可以透过重定向机制，来降低 proxy 服务器上的路由请求的压力，从而提高消息转发的效率。重定向允许服务器在一个请求的应答中，推送路由信息到客户端，这样就可以做到把自己从后续的消息流中脱离出来，但是同时又能提供准确的请求定位服务。当请求的发起者接收到转发的应答之后，他会重新产生一个基于接收到的 URI（s）的请求。从 network 的中央转发到最边缘，转发机制可以适用于跨度很大的网络。

转发服务器在逻辑上是通过一个服务器的 transaction 层建立的，他作为 transaction user 可以访问某些类型的位置服务（参见第 10 节有关注册服务器和位置服务的说明）。这个位置服务可以很方便的用数据库里边的一个 URI 对应多个地址 URI（可能在这个地址找到对应的客户）来实现。

一个重定向服务器并不发出任何指向它自己的请求。接收到任何一个非 CANCEL 的请求之后，服务器要么拒绝这个请求，要么从位置服务器上找到这个请求应该去的其他位置然后返回一个 3xx 的终结应答。对于合法的 CANCEL 请求，它应该返回一个 2xx 的应答。这个应答将会结束掉 SIP 事务。重定向服务器在整个 SIP 事务中位置事务的状态。在重定向服务器之间检测循环死锁应该是客户端的责任。

当重定向服务器给请求方返回一个 3xx 应答的时候，它会在 Contact 头域填写一组（一个或者多个）其他位置信息。如果需要指明这个 Contact 数据的生存周期，可以在 Contact 头域添加一个”expires”参数。Contact 头域包含指向新位置的 URI 或者可以试试看的用户名，或者就是简单的附加通讯参数等等。301（永久去除）或者 302（临时去除）应答同样可以指定和原始请求一样的目的位置和用户名，但是会附加像不同的服务器或者多点传送地址、或者 SIP 由 UDP 到 TCP 传输等等这样的通讯参数。
重定向服务器必须不能重定向一个请求到它自己的 Request-URI 列表中的地址；但是倘若那个 URI 并没有指向自己，重定向服务器可以路由这个请求到目的 URI，或者可以返回一个 404 拒绝。

如果一个客户端正在使用外出的 proxy，并且这个 proxy 实际上是重定向请求，那么就可能出现无限重定向循环。

注意，Contact 头域的值可能指向一个与原始呼叫者无关的资源。比如，一个 SIP 对 PSTN（本地电话网）网关的呼叫可能被转递给一个特别的说明，比如”你所呼叫的电话号码已经改为….”。Contact 应答头域可以包含任何合适的能处理这个呼叫对方的 URI，并不限于 SIP URI。比如，它可以是电话，传真，或者 IRC（如果有支持和定义的话）或者一个 mailto：（RFC 2368[32]） URL。在 26.4.4 节讲述了对 SIPS URI 到非 SIPS URI 的实现和限制。

在 Contact 头域中的”expires”参数指明了这个 URI 的生存周期。这个参数的值是以秒作为单位的。如果这个参数没有提供，那么这个”expires”的缺省是有 Expires 头域所指定的。这个参数中如果设置了非法的值，那么都应改更改成为缺省的 3600。这提供了对 RFC2543 的向后兼容，RFC2543 允许在这个头域中填写绝对时间。在本协议中，如果这个 expires 填写了绝对时间，那么就会当作是非法的值，就会被当作是 3600。

重定向服务器必须忽略自己所不能理解的特性（包括不认识的头域，不认识的 Require 中的 option tag，甚至是不认识的方法名），并且带着问题处理这个请求的重定向。

9 取消一个请求(Cancel)

前边几节讲述了对所有方法的处理请求和处理应答的 UA 的通用处理过程。在本节中，我们讨论一个通用的方法，CANCEL。CANCEL 请求，就像名字所说的，是用来取消客户端发起的上一个请求的。特别是，它请求 UAS 去终止上一个请求并且对上一个请求产生一个错误的应答。CANCEL 对 UAS 已经给出终结应答的请求无效。所以，CANCEL 请求的最大用处是取消需要服务器长时间处理的请求。也就是说，CANCEL 最常用来处理取消 INVITE 请求，因为 INVITE 通常需要花费很长时间来产生一个终结应答。在这种使用中，UAS 接收到对一个 INVITE 请求的 CANCEL 请求，当这个 INVITE 还没有得到终结应答的时候，UAS 会”停止振铃”，并且给 INVITE 请求一个错误的应答（487）。

CANCEL 可以由 proxy 或者 UAC 发起。15 节讲述了在何种情况下，UAC 会 CANCEL 一个 INVITE 请求，在 16.10 节讲述了 proxy 对 CANCEL 的使用。

一个有状态的 proxy 需要对 CANCEL 进行响应，而不是简单的转发从下行流中接收到的一个应答。基于这个原因，CANCEL 是一个”点对点”（hop-by-hop）的请求，也就是说，CANCEL 需要每一个有状态的 proxy 节点进行处理和应答。

9.1 客户行为(Client Behavior)

CANCEL 请求不应该取消除了 INVITE 之外的请求。因为除了 INVITE 之外的请求的响应都是立即响应的，所以，发送 CANCEL 来取消一个非 INVITE 请求总是形成一种赛跑的局面（就是说，cancel 先到还是被取消的请求先到）。

下列步骤用于创建一个 CANCEL 请求。在 CANCEL 请求中的 Request-URI，Call-ID，To，Cseq 的数字部分，From 这些头域都必须和被取消的请求头域一样，包含这些头域的 tags. 客户端创建的 CANCEL 必须只有一个 Via 头域值，这个头域值和被取消的请求的最上一个 Via 头域值相同。这些头域的值和请求的值相同可以让 CANCEL 请求和被取消的请求相匹配（9.2 节描述了如何匹配）。在 Cseq 请求头域的 method 部分必须是一个 CANCEL 方法。这个让这个 CANCEL 请求被当作自己的事务而被正确的鉴别和处理（参见 17 节）。

如果被取消的请求包含一个 Route 头域，CANCEL 请求也必须包含这个 Route 头域的值。这个是让无状态的 proxy 能够正确路由 CANCEL 请求。

CANCEL 头域必须不能包含任何 Require 或者 Proxy-Require 头域。

一旦 CANCEL 请求被创建了，客户端应当检查是否收到了这个 CANCEL 请求取消的原始请求的任何应答（临时的或者终结的应答）。如果没有任何临时应答收到，这个 CANCEL 请求一定不能发送，直到客户端等到了第一个临时应答。如果原始请求已经收到一个终结应答，这个 CANCEL 也不应当发送，因为 CANCEL 请求对已经产生了终结应答的请求没有任何作用。当客户端决定发送一个 CANCEL,它会为这个 CANCEL 创建一个客户 transaction，并且通过目的地址、端口、传输层来发送 CANCEL 请求。这个 CANCEL 中的目标地址、端口和传输层必须和原始请求一样。

如果允许在接收应答之前发送 CANCEL 请求，那么服务端必须支持在接收原始请求之前接收到 CANCEL 请求。

注意，原始请求的事务和 CANCEL 请求的事务都是互相独立的。也就是说，UAC 判定一个请求的取消不能依赖原始请求的一个 487（请求终止）应答，遵循 RFC2543 协议，UAS 不会产生这样一个应答。如果原始请求经过了 64*T1（T1 在 17.1.1.1 节定义）秒还没有应答，客户端应当认为原始请求已经取消，并且应当销毁对应原始请求的客户端事务。

9.2 服务端行为(Server Behavior)

CANCEL 请求要求服务端的 TU 取消相关的事务（transaction）。TU 根据接收到的 CANCEL 请求，并且假定请求的方法不是 CANCEL 或者 ACK，并且用 17.2.3 节描述的事务匹配方法来匹配事务，这样 TU 就可以决定那个事务需要被取消了，被匹配的事务就是需要被取消的事务。

服务端对 CANCEL 请求的处理依赖于服务器的类型。一个无状态的 proxy 会转发这个请求，一个有状态的 proxy 可能会响应这个请求，并且自己再产生一些 CANCEL 请求，UAS 会响应这个 CANCEL 请求。16.10 节讲述了 proxy 怎样处理 CANCEL 请求。

当 UAS 收到 CANCEL 请求，首先按照 8.2 节的 UAS 通用处理方法进行处理。不过，既然 CANCEL 请求是基于”点对点”（hop－by－hop）的，也是不能再提交的，他们不能由服务器为了获得 Authorization 头域中正确的认证而反复尝试。注意，因此 CANCEL 请求也不能包含 Require 头域。

如果根据上边的步骤，UAS 不能找到与 CANCEL 请求相匹配的事务，它应该给 CANCEL 一个 481 应答（调用的 Leg/Transaction 不存在 会话/事务不存在）。如果对应原始请求的事务存在，那么 UAS 在接收到 CANCEL 请求的处理就依赖于是否已经给这个原始请求发出了终结应答。如果已经发出了，不会对 CANCEL 请求对应的原始请求做任何处理，不会更改任何会话状态，不会对原始请求的应答做任何处理。如果 UAS 没有发出对原始请求的终结应答，它会依赖于 CANCEL 所取消的原始请求方法。如果原始请求方法是 INVITE，UAS 应当立刻响应 INVITE 一个 487（请求终止）。本协议中，对 CANCEL 取消的其他本协议中定义的方法没有约定。

不管原先请求的方法是什么，只要 CANCEL 匹配一个事务，UAS 就响应 CANCEL 请求一个 200（OK）应答。这个应答根据 8.2.6 节约定构造。注意，给 CANCEL 应答的 To tag 和给原始请求的应答的 To tag 应该是一样的。对 CANCEL 的应答会通过服务端 transaction 来传送。

10 注册(Registrations)

10.1 概览

SIP 提供了一个搜索机制，如果一个用户希望建立和其他用户的会话，SIP 必须查找能够找到对方用户正在使用的当前主机(hosts)。这个搜索机制经常被 SIP 网络基本元素使用，比如 proxy 服务器，重定向服务器等等。他们在接收、以及响应一个请求的时候，会基于这个用户的位置信息来判定这个消息应该发送到哪里。要实现这个，SIP 网络部件考虑了一个抽象的服务：位置服务；位置服务是通过对特定地区提供地址绑定来实现的。这些地址绑定转换输入的 SIP 或者 SIPS URI，比如 sip:bob@biloxi.com，转换到一个或者一组更加”接近”目标用户的 URI，比如 sip:bob@engineering.biloxi.com。基本上，一个 proxy 会从把输入的 URI 转换到用户实际位置的位置服务中得到最终用户的位置。

注册服务为特定地区的位置服务创建绑定关系，这个绑定关系是用来建立包含一个或者多个联系地址的 address-of-record URI。因而，当那个地区的 proxy 接收到一个请求，这个请求的 Request-URI 和 address-of-record 的记录匹配，那么这个 proxy 会转发请求到这个 address-of-record 中登记的联系地址中去。通常，只有当对那个 address-of-record 的请求会被路由到这个区域的时候，登记这个 address-of-record 到这个这个区域的位置服务才是有意义的。在大多数情况下，这个就要求登记服务所覆盖的区域和 URI 中的 address-of-record 所覆盖的区域相同。有很多种方法来建立位置服务。一个方法是 administratively(管理)。在上述的例子种，Bob 我们通过查询公司数据库知道他是一个工程部职员。不过，SIP 提供了一个让 UA 能够创建精确绑定的机制。这个机制就是登记服务。登记服务需要向一个特殊的 UAS 服务器（注册服务器 registrar）发出 REGISTER 请求。注册服务器(registrar)为一个区域的位置服务作为前端接入，根据 REGISTER 请求的内容读写位置对照表。这个位置服务通常为处理这个区域的 proxy 服务器提供位置服务。

总的登记服务处理流程在图 2 中说明。需要注意的是，登记服务器(registrar)和 proxy 服务器都是逻辑上的角色，可以在网络中用一个设备来部属；在例子图中是为了能够清楚的表示所以分开描述。还需要注意的是如果他们（登记服务器和 proxy）本身是分开的，那么 UA 可以通过 proxy 服务器发送注册请求。

SIP 本身对实现位置服务器（location service）没有特别的要求。唯一的要求是某些区域的注册服务器（registrar）必须能够对位置服务的数据进行读写，并且这个区域的 proxy 或者重定向服务器必须能够兼容读取相同的数据。注册服务器（registrar）可能和某一个区域的 proxy 服务器部署在一起。

10.2 构造一个 REGISTER 请求

REGISTER 请求用来增加、删除、查询绑定资料。一个 REGISTER 请求可以增加一个 address-of-record 和一个或者多个联系地址之间的绑定。在合适的第三方认证的情况下，可以做 address-of-record 的登记。客户端同样可以删除前边绑定的内容也可以查询 address-of-record 的当前绑定地址。除了特别说明以外，REGISTER 请求的构造以及客户端如何发送 REGISTER 请求和通常的 8.1 节描述的和 17.1 节描述的 UAC 发出请求是一致的。

一个 REGISTER 请求并不建立一个对话。当基于事先给定路由集（8.1 节）的情况下，一个 UAC 可以在 REGISTER 请求中包含一个 Route 头域。在 REGISTER 请求和应答包中，Record-Route 头域并没有任何意义，如果这个头域存在，必须被忽略。另外，UAC 一定不能基于 REGISTER 请求的应答包中的任何 Record-Route 头域来创建新的路由集合。

下面这些头域，除了 Contact,必须在 REGISTER 头域中包含。Contact 头域可选。

Request-URI:　　这个头域指明了登记服务所指明的位置服务所在的区域（比如 sip:chicago.com）。”userinfo”和”@”元素在 SIP URI 中不能出现。

To：　　这个头域包含了被查询、增加、修改的 address-of-record。to 头域和 Request-URI 头域通常是不同的，因为这个由用户名组成。这个 address-of-record 必须是一个 SIP URI 或者 SIPS URI.

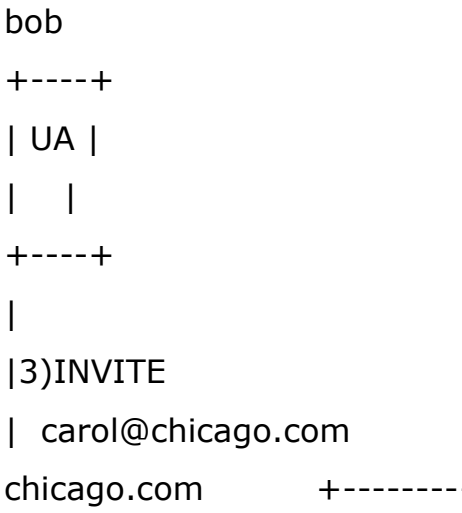
From:　　这个头域包含了提交这个注册信息的用户的 address-of-record 资料。这个值和 To 头域的值相同，除非这个请求是第三方发起的注册请求。

Call-ID:　　UAC 发出的给某个注册服务器（registrar）的所有注册请求都应该有相同的 Call-ID 头域值。如果相同的客户端用了不同的 Call-ID 值，注册服务器(registrar)就不能检测是否一个 REGISTER 请求由于延时的关系导致了故障。

Cseq:　　Cseq 值保证了 REGISTER 请求的正确顺序。一个 UA 为每一个具备相同的 Call-ID 的 REGISTER 请求顺序递增这个 Cseq 字段。

Contact:　　REGISTER 请求可以有一个 Contact 头域。这个头域可以有 0 个或者多个包含绑定地址信息的值。

UA 在没有收到上一个注册请求的应答或者上一个 REGISTER 请求超时之前，禁止发送新的注册请求（就是说，包含一个新的 Contact 头域值，而不是重发）。



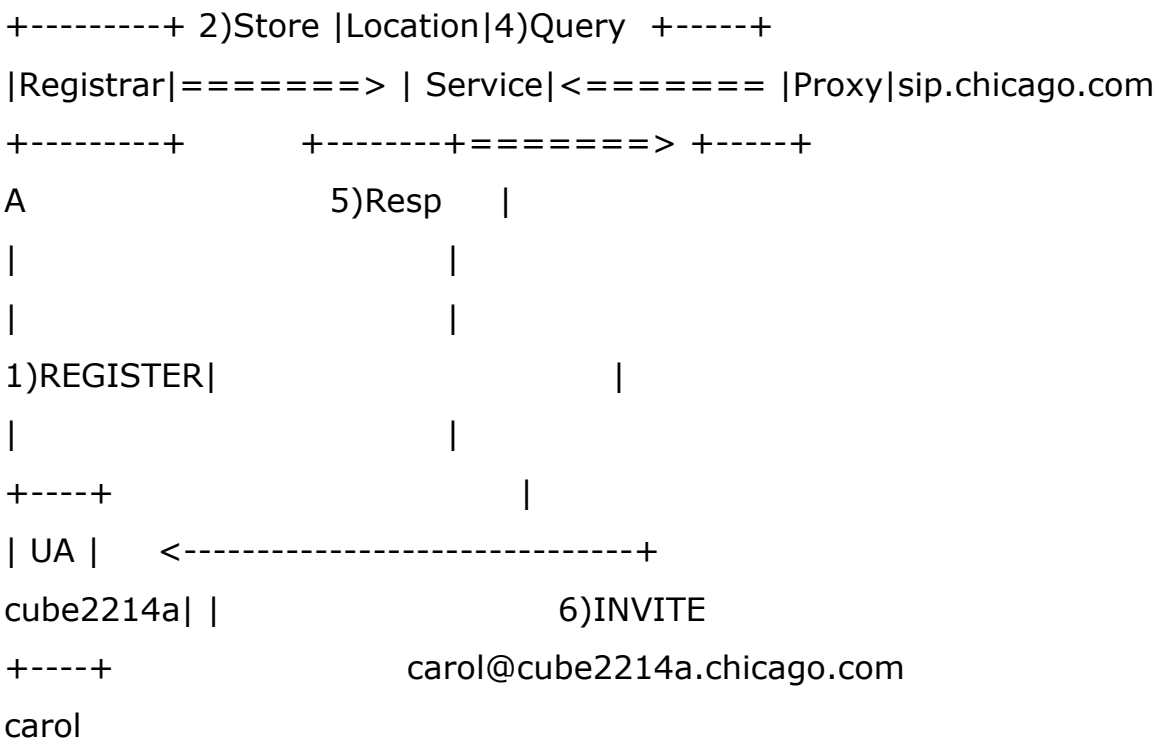


图 2：REGISTER 例子

下边的 Contact 头域参数在 REGISTER 请求中有特别的意义：

action: 在 RFC2543 中的“action”参数已经废弃了，UAC 不能使用”action”参数。

expires：“expires”参数表明 UA 的绑定的有效时间。以秒为单位的整数。如果本参数没有制定，那么这个参数的值就是 Expires 头域的值。实现中，可以把超过 2**32-1 的值（4294967295 秒或者 136 年）认为是 2**32-1。非法的值应当视同 3600。

10.2.1 增加绑定

REGISTER 请求是向注册服务器（registrar）发送一个包含对某一个 address-of-record 的地址的 SIP 请求应当发送的实际联系地址。address-of-record 包含在 REGISTER 请求的 To 头域中。

请求中的 Contact 头域通常包含了 SIP 或者 SIPS 的 URI，这些 URI 表明了特定的 SIP 端点（比如 sip:carol@cube2214a.chicago.com）,他们也可以使用其

他的 URI 表示方法。一个 SIP UA 可以选择注册一个电话号码（比如使用 tel URL, RFC 2806[9]）或者一个 email 地址（比如用 mailto URL, RFC2368[32]）来作为 address-of-record 的联系地址 Contact 域。

例如，Carol,有一个 address-of-record”sip:carol@chicago.com”，将会在区域 chicago.com 的注册服务器上注册。她的注册服务信息将会被 chicago.com 区域的 proxy 服务器使用，用来路由和转发到 Carol 的 address-of-record 请求到她的 SIP 终端。

当客户端在注册服务器（registrar）上建立好了绑定以后，它可以根据需要发送后续的注册请求，包含新的绑定信息或者修改以前的绑定信息。给 REGISTER 请求的 2xx 应答中，在 Contact 头域中是在这个注册服务器（registrar）上登记的完整的这个 address-of-record 的绑定列表。

如果 REGISTER 请求中的 To 头域中的 address-of-record 是一个 SIPS URI, 那么任何在 REGISTER 请求中的 Contact 头域都应当是 SIPS URI。客户端只有在有其他手段保证非 SIPS URI 的安全性的情况下，才能在 SIPS 的 address-of-record 的地址上注册非 SIPS URI。这个也可以适用域使用非 SIP 协议的 URI，或者用非 TLS 来加密的 SIP 设备。

注册并不需要更新所有的绑定。一般情况下，UA 只更新它现在的联系地址。

10.2.1.1 设置 Contact 地址的过期参数

当一个客户端发出一个 REGISTER 请求，它可能包含一个过期参数用来表示这个注册的地址的有效期。（就像在 10.3 节讲述的那样，注册服务器（registrar）根据自己的策略选取实际的时间间隔来计算有效期）。

客户端设置有效期的方法有两种：一个是通过设置 Expires 头域，一个是通过设置”expires”contact 头域的参数来设置。 后一种允许针对同一个 REGISTER 请求中的多个绑定联系地址中的每一个联系地址单独设定有效期，然后所有没有带”expires”参数的 Contact 头域的值都使用 Expires 的设置。

如果 REGISTER 中没有两种有效期都没有设置，这就表明这个有效期由服务器来决定。

10.2.1.2 Contact Adress 的参数选择

如果在一个 REGISTER 请求中包含多个 Contact，着说明 UA 想要把这些 Contact 头域的内容都和 To 头域中制定的 address-of-record 地址绑定起来。这个列表可以用”q”参数来区分 Contact 头域的优先级。”q”参数用来标志特定 Contact 头域值和其他绑定的 address-of-record 的联系地址之间的优先级。16.6 节讲述了一个 proxy 服务器如何使用优先级。

10.2.2 删除绑定

注册信息是一个纯粹软件的状态，并且如果不刷新会过期。如果需要，也可以被删除。一个客户端可以设置注册服务器(registar)的有效期（10.2.1）。一个 UA 可以通过发出有效期为”0”的 REGISTER 请求，使某一个联系地址立刻失效。UAS 都需要实现这个机制使得在联系地址过期前能够被删除。

REGISTER 规范中的 Contact 头域如果设置成为”*”则表示需要操作所有的注册项。但是也只能在具有一个 Expires 头域并且这个头域为’0’的情况下能够使用。（这就是说，只能够在要删除所有的注册项的时候使用”*”）。

用”*”来删除所有的注册项有一个好处，就是使得 UA 不需要知道每一个注册项的精确值。

10.2.3 访问绑定

无论请求是否包含了 Contact 头域，给任何 REGISTER 请求的成功应答都包含了一个完整的绑定列表。如果 REGISTER 请求头域中不包含 Contact 头域，那么注册服务器的绑定列表将不会改变。

10.2.4 刷新绑定

每一个 UA 都对先前它建立的绑定信息由刷新的义务。禁止对其他 UA 建立的绑定信息进行刷新。在注册服务器（registrar）给出的 200（OK）应答中，包含了的 Contact 头域中列明了所有的当前绑定信息。UA 需要挨个比较这些联系地址，看看是否这个地址是可以建立联系的，这个比较是通过 19.1.4 节中的比较规则来进行的。如果是，它通过更新 expires 参数来更新过期时间（或者 Expires 头域）。于是在这些绑定信息过期前，UA 为每个绑定发出 REGISTER 请求来刷新绑定。也可以通过一个 REGISTER 请求来刷新数个绑定请求。

UA 在一个刷新周期中，应该使用相同的 Call-ID 来进行注册调用。刷新的注册信息应该是和原来登记的信息一致。

10.2.5 设置内部时钟

如果 REGISTER 请求的应答中包含一个 Date 头域，客户端可以用这个头域来校正当前内部的时钟。

10.2.6 寻找注册服务器

UA 有 3 种方法来决定向哪里发出注册请求：通过配置，使用 address-of-record,广播方式。一个 UA 可以用非本文档规定的方式，配置一个注册服务器的地址。如果 UA 没有配置任何注册服务器的地址，UA 应该用请求的 Request-URI 部分种的 address-of-record 的服务器部分（host part），用普通的 SIP 服务器定位机制[4]。比如，用户”sip:carol@chicago.com”地址的注册服务应该是”sip:chicago.com”。

最后，UA 可以通过监听广播的形式来获得注册服务器地址。监听广播的注册服务器是通过监听著名的”全部 SIP 服务器”广播地址”sip.mcast.net”(224.0.1.74)。没有 Ipv6 的广播地址。SIP 的 UA 可以监听这个地址，并且用这个来知道其他本地用户的地址（见附件[33]）；不过他们并

不对请求做响应。通过监听广播的登记服务可能在某些环境下不能用，比如，基于同一个本地网络的多个商业应用的情况。

10.2.7 传送一个请求

当 REGISTER 请求被构造好以后，并且也有了登记服务起地址，UAC 遵循 8.1.2 节的说明来提交 transaction 层来发送 REGISTER 请求。如果 transaction 层返回一个由于 REGISTER 请求没有应答的超时错误，UAC 不应该立刻重新尝试对同一个注册服务器的注册请求。

立刻重新尝试很有可能导致再次超时。等待一个合理的时间在尝试可以降低网络的负载，在这里并没有一个约定的等待时间间隔。

10.2.8 错误响应

如果 UA 接收到一个 423（间隔太简略）应答，它可能需要更改 REGISTER 请求中的所有有效期，使得这些有效期必须大于等于 423 应答头中的 Min-Expires 头域中的有效期，并且重新尝试发送这个 REGISTER 请求。

10.3 处理 REGISTER 请求

一个注册服务器(registrar)就是一个 UAS，这个 UAS 用来响应 REGISTER 的请求，并且维持一个绑定表，这个绑定表用来提供给它所管理的区域中的 proxy 服务器和重定向服务器的。一个注册服务器根据 8.2 和 17.2 中的规定来处理请求，但是它仅仅处理 REGISTER 请求。一个注册服务器禁止产生 6xx 应答。一个注册服务器可以适当的转发 REGISTER 请求。通常用于一个注册服务器（registar）监听一个多点广播，并且通过 302 应答（临时转移）转发这个多点广播的 REGISTER 请求到它正确的处理位置。

注册服务器必须忽略在 REGISTER 请求中的 Record-Route 头域，并且也不能在 REGISTER 请求的应答中包含任何 Record-Route 头域。注册服务器可能收

到一个有 proxy 转发过来的 REGISTER 请求，这个请求中由于 proxy 处理这个请求当作未知请求所以添加了 Record-Route 头域。

一个注册服务器必须知道（例如，通过配置）它所管理的区域。注册服务器一定需要按照接收到的 REGISTER 请求顺序进行处理。

REGISTER 请求必须当作原子请求来处理，意味着给定的 REGISTER 请求要么就完整处理，要么就完全拒绝。每一个 REGISTER 信息的处理都和其他的注册和绑定信息的处理无关。

当接收到一个 REGISTER 请求，注册服务器（registrar）按照如下步骤处理：

1、注册服务器（registrar）检查 Request-URI 来决定是否它属于本注册服务器所管理的区域的 Request-URI。如果不是，并且如果这个服务器同时也作为一个 proxy 服务器，那么这个服务器应当转发这个请求到指定的区域。这个转发是根据 16 节所规定的 proxy 通用信息处理规则来进行的。

2、为了保证注册服务器能够支持所需要的扩展，注册服务器必须遵循 8.2.2 节描述的情况来处理 Require 头域。

3、一个注册服务器应当对 UAC 进行身份认证。SIP UA 的身份认证机制在 22 节描述。注册这个动作需要遵循 SIP 的通用的身份认证框架。如果没有任何认证机制，注册服务器可以使用 From 地址来作为原始请求的信任依据。

4、注册服务器应当检查认证的用户是否通过认证来更改这个 address-of-record 的登记权限。比如，一个登记服务起(registrar)可以通过访问一个认证数据库，这个认证数据库映射了用户名和一个 address-of-record 列表，这些列表是用户可以更改绑定信息的列表。如果认证用户并没有权利修改绑定信息，注册服务器应当返回一个 403（禁止访问）并且跳过后续步骤。在支持第三方认证和注册的架构下，一个实体可以被授权来更新多个 address-of-record 的注册信息。

5、注册服务器(registrar)从 REGISTER 请求的 To 头域中解出 address-of-record。如果这个 address-of-record 并非在这个 Request-URI 指明的区域中合法，那么注册服务器必须发出一个 404(没有找到)的应答，并且跳过后续步骤。接着 URI 必须转换成为标准的格式。所有的 URI 参数都必须删去（包括用户

参数 user-param）,并且任何非法(escaped)字符必须转换成为合法字符(unescaped)格式。最后形成一个可以用于绑定的列表。

6、注册服务器(registrar)检查是否请求包含了一个 Contact 头域。如果没有包含，它跳过到最后一步。如果 Contact 头域包含了，注册服务器检查是否有一个 Contact 头域值是”*”，并且包含了一个 Expires 头域。如果请求有其他的 Contact 头域或者任何有效期的值是非 0 的，这个请求就是非法请求，并且服务器必须送回一个 400（非法请求）的应答，跳过后续步骤。如果没有，那么注册服务器检查是否 Call-ID 复核每一个绑定的值。如果不符合，它必须删除绑定。如果复核，它必须仅仅删除保存的绑定表中 CSeq 值小于请求中的 Cseq 值的记录。否则，更新必须终止，请求失败。

7、现在注册服务器(registrar)可以依次处理 Contact 头域中的联系地址了。对于每一个地址，它根据下边的规则进行有效期检查。

— 如果含有”expires”参数，这个参数值就是最终的有效期。

— 如果没有这个参数，并且请求有一个 Expires 头域，那么这个值就是有效期。

— 如果没有 Expire 头域也没有参数，那么本地的缺省配置应当作为有效期。

注册服务器可以选择一个小于请求中的有效期值作为最后的有效期。当且仅当请求的有效期大于 0 并且小于 1 个小时并且小于一个注册服务器配置的最小有效期的时候，注册服务器可以响应一个 423（有效期过小）的错误来拒绝。这个应答必须包括了一个 Min-Expires 头域来表明本注册服务器所接收的最小有效期，然后跳过所有后续步骤。

允许注册服务器设置注册服务器自己的有效期防止了过分频繁的刷新注册信息，并且也降低了维持和管理注册信息的工作量。在服务的创建的时候，注册信息中的有效期会经常用到。一个例子是 follow-me(跟随我)服务，在这个服务中，一个用户可能在一个终端上只停留一小会儿。因此，注册服务器应当接收简略的注册；一个请求只有当它的有效期过短的时候（短到可能降低注册服务器性能的时候）才应当被拒绝。

对于每一个地址，注册服务器在当前绑定列表中用 URI 比较规则进行搜索。如果绑定不存在，它会暂时性的添加进去。如果绑定存在，注册服务器检查 Call-ID

值。如果 Call-ID 值在这个已经存在的绑定中和本次请求的 Call-ID 值不一样，那么如果绑定的有效期为 0 就删除这个绑定，否则就刷新这个绑定。如果 Call-ID 值一样，那么注册服务器比较 Cseq 值，如果请求中的这个值比现存的绑定值中的 Cseq 高，那么必须更新或者删除这个绑定（如果有效期为 0 就删除，否则就刷新）。如果这个 Cseq 值比现存的 Cseq 值小，那么必须终止更新并且请求失败。这个规则确保了从同一个 UA 过来的请求顺序处理，对于非顺序的请求跳过处理。每一个绑定记录都包含了当时请求的 Call-ID 和 Cseq 的值。

只有当且仅当所有的绑定更新和绑定添加都完成的时候，绑定才可以做 COMMIT 处理（就是说，这次修改对 proxy 和重定向服务器可见）。如果任何更新或者添加失败了（比如，由于后台的数据库 commit 失败），必须给出一个 500（服务器错误）的应答，并且中间进行的临时更新都必须删除。

8、注册服务器（registrar）返回一个 200（OK）应答。这个应答必须包含 Contact 头域，并且这个头域的值中列举了所有当前绑定的注册信息。每一个 Contact 值都必须包含一个”expires”参数，用来标志还有多久这个绑定信息就过期了。应答也必须包含一个 Date 头域。

11 查询能力

SIP 方法 OPTIONS 允许一个 UA 来查询另外一个 UA 或者 proxy 服务器的能力。这个提供个客户端一个手段来查询服务端支持的方法，内容类型，扩展，codecs 等等。这些都不用”ringing”对方。比如，在客户端试图在 INVITE 请求头中增加一个请求字段选项的时候，它并不知道对方 UAS 能否支持这个选项，它就可以用 OPTIONS 来查询一下 UAS，通过检查 OPTIONS 返回的 Supported 头域，就可以知道是否支持这个选项。所有的 UA 都必须支持 OPTIONS 方法。OPTIONS 请求的目标是用 Request-URI 指明的，这个既可以是一个 UA 也可以是一个 SIP 服务器。如果 OPTIONS 指向一个 proxy 服务器，Request-URI 设置成为一个没有用户部分（user part）的，类似 REGISTER 请求中的 Request-

URI 一样。或者，一台服务器收到一个 OPTIONS 请求并且 Max-Forwards 头域值是 0 的时候，它就需要响应这个请求而不需要关心 Request-URI 的内容。
这个机制就像 HTTP/1.1 一样。这个机制可以用来实现类似”traceroute”功能来通过发出一系列的有着增量 Max-Forwards 头域的 OPTIONS 请求来检查每一个途径节点的能力。
就像对一般 UA 机制来说，如果 OPTIONS 没有应答，transaction 层能够返回一个超时错误。这个可能标志着对方无法到达因此无响应。OPTIONS 请求可以作为建立会话的一部分，用来查询对方的能力使用，这样在后续对话中可以使用双方兼容的方式。

11.1 构造 OPTIONS 请求

一个 OPTIONS 请求可以根据 8.1.1 节中的标准构造方法来进行构造。
Contact 头域在 OPTIONS 请求中可以存在，也可以不存在。
Accept 头域应当包含在请求中，用来标志 UAC 希望接收应答中的消息体的类型。通常情况下，这个设置成为 UA 的多媒体兼容能力，比如 SDP(应用/SDP)格式。
对于一个 OPTIONS 请求的应答是假定是在原请求中的 Request-URI 范围内的。但是，仅当一个 OPTIONS 请求作为建立对话的一部分而发送的时候，后续的请求应当由收到并且响应这个 OPTIONS 请求的服务器进行处理。（就是说如果在建立会话的时候使用 OPTIONS 请求，那么 OPTIONS 之后的这些请求都应该由这个 OPTIONS 查询的服务器处理，这样才能保证使用的特性和 OPTIONS 查询出来的能力是一样的）

OPTIONS 请求的例子：

```
OPTIONS sip: carol@chicago.com
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9Hg4bKhjhs8ass877
Max-Forwards: 70
```

To: <sip:carol@chicago.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Cseq：63104 OPTIONS
Contact: <sip:alice@pc33.atlanta.com>
Accept: application/sdp
Content-Length: 0

11.2 处理 OPTIONS 请求

给 OPTIONS 的应答的构造遵循标准的构造规则（8.2.6 节描述）。应答码的选择必须和处理 INVITE 请求一样的应答码（就像处理 INVITE 请求一样的返回）。这就是说，200（OK）应当在 UAS 能够接收请求的时候返回，486（忙）应当在 UAS 如果忙的时候返回。这样 OPTIONS 可以用来检测 UAS 的基本状态，这就是说，我们可以用 OPTIONS 来知道 UAS 能否接收 INVITE 请求。
在一个对话中的 OPTIONS 请求会产生一个 200(OK)的应答，这是和在对话外创建的并且对对话没有任何影响的请求相同。
这个 OPTIONS 的使用有一定的限制，因为对于 proxy 处理 OPTIONS 和 INVITE 请求的不同。一个分支的 INVITE 可以有多个 200（OK）的应答返回，但是一个分支的 OPTIONS 只能有单个 200（OK）应答返回。因为这是由于 proxy 处理 OPTIONS 请求是当作非 INVITE 的处理。参见 16.7 节有详细的说明。
如果 OPTIONS 请求的应答是由 proxy 服务器给出的，proxy 返回一个 200（OK）的应答，列出这个服务器的各种选项和能力。
应答没有消息体
Allow，Accept,Accept-Encoding,Accept-Language,和 Supported 头域应当在 200（OK）应答中出现。如果这个是由 proxy 产生的应答，那么 Allow 头域应当忽略，因为 proxy 是方法无关的（也就是说不知道该如何处理方法的）。

Contact 头域可以在 200（OK）的应答中出现，并且与 3xx 应答有相同的语义。这就是说，他们可以列出指向客户的一串名字和方法的集合（用以转发请求）。一个 Warning 头域是可以存在的。消息体也可以存在，消息体的类型是由 OPTIONS 请求的 Accept 头域指明的(application/sdp 是缺省的，如果 Accpet 头域不存在的话)。如果 Accept 头域中包含了一个类型能描述媒体接收能力，UAS 应当在应答中包含一个消息体用于这个用途。详细的 application/sdp 包体说明在[13]中描述。

UAS 生成的 OPTIONS 应答例子。（对应 11.1 节中的请求例子）

SIP/2.0 200 OK
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKhjhs8ass877
; received = 192.0.2.4
To: <sip:carol@chicago.com>;tag=93810874
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Cseq: 63104 OPTIONS
Contact: <sip:carol@chicago.com>
Contact: mailto:carol@chicago.com
Allow: INVITE,ACK,CANCEL,OPTIONS,BYE
Accept: application/sdp
Accept-Encoding: gzip
Accept-Language: en
Supported: foo
Content-Type: application/sdp
Content-Length:274
(SDP not shown)

12 对话(Dialog)

一个 UA 的核心概念就是对话。对话是表现为两个用户代理（UA）之间的持续一段时间的点对点的 SIP 关系。对话（Dialog）使得用户代理之间的消息顺序传递和两个用户代理之间的请求正确路由更加容易。对话（Dialog）可以认为是对 SIP 消息解释的上下文关系。第 8 节讲述了方法无关的 UA 处理和响应对话（Dialog）外的请求。本节将讨论如何通过请求和应答来创建一个对话（Dialog），并且在对话（Dialog）中如何发起和响应后续的请求。

一个对话在参与对话的 UA 中都有一个 dialog ID 作为标记，这个 ID 由 Call-ID，和一个本地 tag 和远程 tag 组成。各个 UA 的 dialog ID 在对话中是不一样的。特别是，在一边 UA 的本地 tag，在另外一方就是远程 tag。这些 tag 都是互相不透明的，并且使得整个 dialog ID 是唯一的。

dialog ID 同样是和所有的 To 头域中包含了 tag 参数的请求及应答相关。

填写一个消息中的 dialog ID 的规则依赖于 SIP 元素是 UAC 还是 UAS。对于 UAC 来说，dialog ID 中的 Call-ID 的值会填写到消息中的 Call-ID 域中，远程 tag 放在消息中的 To 的 tag 参数中，本地 tag 放在 From 的 tag 参数中。（这些规则对请求和应答都适用）。对于 UAS 来说，dialog ID 的 Call-ID 值放在消息的 Call-ID 头域中，远程 tag 放在 From 头域的 tag 中，本地 tag 放在 To 头域的 tag 参数中。

一个对话包含一些特定的状态用于以后的对话中的消息传送。这个状态由 dialog ID,本地序列号（用来排序 UA 到对方的请求的序列），远程序列号（用来排序请求从远端到本 UA）,本地 URI,远端 URI,remote target,一个布尔类型的标记”secure”，路由集合（一组有序的 URI）组成。

路由集合是由发送请求到对方需要途径的一组服务器列表组成。一个对话可以处于”early”状态，这是由于当这个对话收到了临时应答而创建，并且当收到了 2xx 终结应答的时候转换到”confirmed”状态。对于其他应答，或者没有应答，”early”对话将会终结。

12.1 创建一个对话

对话是由对一组特定请求的没有失败的应答来创建的。在本规范中，只有包含 To tag 的 2xx 和 101－199 应答，并且请求是 INVITE 的，会建立一个对话。当收到一个非终结应答的时候，对话会建立成”early”状态，并且成为 early dailog。创建对话的时候可以使用 Extension 来定义扩展。13 节描述了 INVITE 请求的更多细节。在这里，我们描述与方法无关的对创建对话状态的处理。

UA 必须按照下边描述的方法对 dialog ID 进行赋值。

12.1.1 UAS 行为

当 UAS 响应一个请求给出一个应答，并且这个应答会建立一个对话的时候（比如对 INVITE 的 2xx 应答），UAS 必须拷贝所有的请求中的 Record-Route 头域到应答中去（包括 URI,URI 参数，和其他任何 Record-Route 头域的参数，无论 UAS 是不是认识的参数都需要原样拷贝），并且必须维持这些参数的顺序。UAS 必须增加一个 Contact 头域给应答。这个 Contact 头域包含一个 UAS 在后续对话请求中接收请求的地址（这个包含了给 INVITE 请求的 2xx 应答的 ACK 请求处理的地址）。通常情况下，UAS 会用 IP 地址或者 FQDN 形式来发布自己的这个 Contact 地址。这个在 Contact 头域中的 URI 必须是一个 SIP 或者 SIPS URI。如果创建对话的请求在 Request-URI 中包含的是 SIPS URI，或者在 Record-Route 头域的最上的一个值是 SIPS URI,或者如果请求中没有 Record-Route 头域但是请求中的 Contact 头域是 SIPS URI,那么给出的应答中的 Contact 头域必须是一个 SIPS URI。 这个 URI 应该是全局有效的（就是说，这个 URI 可以用于对话外的消息）。同样的，在请求 INVITE 中的 Contact 头域的 URI 也不应当仅限于这个对话中使用。因此它可以用于对话外的消息中。

UAS 接着创建这个对话的状态。对话状态必须维持直到对话结束。

如果请求是通过 TLS 过来的，并且 Request-URI 包含一个 SIPS URI,”secure” 标志将被赋值成为 TRUE。

路由集合必须设置成为请求中的 Record-Route 的 URI 列表，保留所有的 URI 参数和顺序。如果请求中没有 Record-Route 头域，那么路由集合必须设置成为空。这个路由集合，即便是空的，为了以后的对话中的请求，也要覆盖任何预先存在(pre-existing)的路由集合。remote taget 必须设置成为请求的 Contact 头域中的 URI。

远程序列号必须设置成为请求中的 Cseq 头域的序列号。本地序列号必须设置成为空。dialog ID 中的呼叫标志应该设置成为请求的 Call-ID 头域的值。dialog ID 的本地 tag 必须设置成为对请求的应答包中的 To 头域的 tag，并且 dialog ID 的远程 tag 必须设置成为请求中的 From 头域中的 tag。UAS 必须能够处理接收到的请求中的 From 头域没有 tag 标志，在这种情况下，这个 tag 就是空值。这是为了兼容 RFC2543 协议，它并没有定义 From tag。

远程 URI（remote URI）必须设置成为 From 头域中的 URI，并且本地 URI 必须设置成为 TO 头域中的 URI。

12.1.2 UAC 行为

当一个 UAC 发出一个请求，这个请求能够建立一个对话（比如这个请求是 INVITE），它必须在 Contact 头域中提供一个基于全局的 SIP 或者 SIPS URI（例如，可以在对话外使用的 SIP URI）。如果请求包含一个 Request-URI 或者最上的 Route 头域是 SIPS URI,Contact 头域也必须包含的是 SIPS URI。

当一个 UAC 接收到应答，并且这个应答建立对话的时候，它也同样构造这个对话的状态。这个状态必须维持到对话的结束。

如果这个请求是基于 TLS 发送的，并且 Request-URI 包含一个 SIPS URI,那么”secure”标志被设置成为 TRUE。

路由集合必须设置成为应答中的 Record-Route 头域的 URI 列表，保留所有的 URI 参数和顺序。如果在应答中没有 Record-Route 头域，那么这个路由集合必须设置成为空集合。这个路由集合即便是空的，为了以后的对话中的请求，也要覆

盖任何预先存在(pre-existing)的路由集合。remote taget 必须设置成为应答中的 Contact 头域的 URI。

本地序列号必须设置成为请求中的 Cseq 头域的序列号。远程序列号必须设置成为空（他会由远端的 UA 在对话中发送请求而建立）。dialog ID 中的呼叫标志必须设置成为请求的 Call-ID 头域的值。dialog ID 的本地 tag 必须设置成为请求中的 From 头域的 tag，dialog ID 的远程 tag 必须设置成为应答中的 To 头域的 tag。UAC 必须能够处理接收到的应答的 To 头域中没有 tag 的情况，在这个情况下，tag 值取值成为空。这是为了能够向下兼容 RFC2543，它没有规定 To 的 tag。

remote URI 必须设置成为 To 头域的 URI，local URI 必须设置成为 From 头域的 URI。

12.2 对话中的请求

当两个 UA 之间的对话建立以后，他们都可以在对话中初始化一个新的事务（transaction）。如果 UA 发送请求，将遵循 UAC 的事务规则。UA 接收请求将遵循 UAS 的规则。在建立对话的事务过程中，UA 扮演的角色可能是不一样的。

在对话中的请求可以包含 Record-Route 和 Contact 头域。不过，虽然他们会修改 remote target 的 URI，但是这些请求也不会导致对话的路由集被改变。明确说，如果请求不是刷新 target 的请求，那么这个请求不会更改对话的 remote target URI,如果请求是刷新 target 的请求，那么这个请求才会更改对话的 remote target URI。对于用 INVITE 建立的对话来说，唯一的能够刷新 target 的请求就是 re-INVITE(见 14 节说明)。可能会有其他扩展定义通过其他方法来刷新 target 的请求。

注意 ACK 不是一个刷新 target 的请求。

刷新 target 请求只会更改对话的 remote target URI，并且更改由 Record-Route 指定的路由集合。如果更新路由集合会带来严重的和 RFC2543 向后兼容问题。

12.2.1 UAC 行为

12.1.1.1 产生请求

在对话中的请求是通过用许多对话的状态部分来构造的。在 TO 头域中的 URI 部分必须设置成为对话状态中的 remote URI。To 头域的 tag 参数必须设置成为 dialog ID 中的 remote tag 部分。请求的 From URI 必须设置成为对话状态中的 local URI。From 头域的 tag 参数必须设置成为 dialog ID 的 local tag 部分。如果 remote 或者 local tag 是空值，那么 tag 参数必须分别从 From 或者 To 头域中去除。

在请求序列中的原始请求的 To 和 From 头域的 URI 的使用方法是为了向下兼容 RFC2543 协议的，在 RFC2543 协议中，使用 URI 作为对话的标志。在这个规范中，只有 tags 用于区分对话。有可能在本协议的后续版本中，在对话中的请求必须强制反应原始请求的 To 和 From 头域的 URI 将会去除。

请求的 Call-ID 必须设置成为对话的 Call-ID。在对话中的请求必须严格遵循单个递增的 Cseq 序列号（每次增加 1）（当然要除了 ACK 和 CANCEL，这两个请求中的 Cseq 必须和原始的请求或者确认请求一样）。因此，如果本地序列号（local sequence number）不为空，那么本地序列号码必须依次增加 1，并且这个数值要存放到 Cseq 头域中。如果本地序列号码是空的，那么在 8.1.1.5 节约定的初始值必须填写进去。在 Cseq 头域中的 method 字段必须和请求的方法（method）一致。

通过使用 32 位的长整数，使得即使每秒种产生 1 笔请求，也会要 136 年才会用完这个整数出现重复。这个序列号的初始值的选取是为了让对话中后续的请求序列号不会重复。非 0 的初始值可以考虑采用时间来作为初始的序列号。一个客户端可以用 31 位有符号整数或者 32 位无符号整数来存放时间作为初始化的序列号。

UAC 使用 remote target 和路由集合来构造请求中的 Request-URI 和 Route 头域。如果路由集合是空的，那么 UAC 必须把 remote target URI 放到 Request-URI 中，并且 UAC 不能添加 Route 头域到请求中。

如果路由集合不为空，并且路由集合的第一个 URI 包含 lr 参数（见 19.1.1），那么 UAC 必须填写 remote target URI 到 Request-URI，并且必须包含 Route 头域，这个 Route 头域按照顺序填写路由集合和路由集合的参数。

如果路由集合不为空，并且路由集合的第一个 URI 没有包含 lr 参数，那么 UAC 必须把第一个 URI 放在 Request-URI 中，并且拆去所有不被 Request-URI 允许的参数。UAC 必须增加一个 Route 头域顺序包含所有剩下的路由集合元素，及其参数。UAC 接着必须把 remote target URI 放在 Route 头域的最后一项。

例如，如果 remote targe 是: sip:user@remoteua 并且路由集合包括：

<sip:proxy1>,<sip:proxy2>,<sip:proxy3;lr>,<sip:proxy4>

那么请求应该有下列的 Request-URI 和 Route 头域

METHOD sip:proxy1

Route:

<sip:proxy2>,<sip:proxy3;lr>,<sip:proxy4>,<sip:user@remoteua>

如果路由集合的第一个 URI 不包含 lr 参数，那么对应的说明 proxy 并不能支持本文档所约定的路由机制，而是支持 RFC2543 文档所约定的路由机制，那么在发送信息的时候需要通过替换 Request-URI 为接收到的第一个 Route 头域的值。

将 Request-URI 的值放在 Route 头域的目的是为了保护 Request－URI，使得它经过严格路由的时候不丢失（当请求遇到一个松散路由的时候会返回到 Request-URI 中？？？？？。）

在对话内的任何一个刷新 target 的请求中，都应当包含一个 Contact 头域，并且这个 URI 除非有必要，否则都应当是和对话内上次请求的 URI 值一样。如果”secure”标志设置成为 TRUE,那么 URI 也应当是 SIPS URI。

如果在 12.2.2 节讨论的那样，在刷新 target 请求中的 Contact 头域会更新 remote target URI。这个允许 UA 提供一个新的联系地址（Contact address），表明它在对话中改变了自己的地址。不过，如果请求不是刷新

target 的请求，那么不会影响对话中的 remote target URI。请求中的剩下的部分请按照 8.1.1 节描述的填写。一旦请求被创建了，请求将按照对话外请求发送标准步骤（8.1.2 节）来解析服务器的地址并且发送请求。

8.1.2 节中的步骤一般把请求发送到 Route 头域的最上一个地址，或者如果没有 Route 头域，那么就发送到 Request-URI 地址。由于受到特定的限制，这些步骤也允许把请求发送到另外一个地址（比如在 route set 中没有的缺省的外发 proxy）

12.2.1.2 处理应答

UAC 将会从 transaction 层收到请求的应答。如果客户端的事务层返回一个超时，这会等同于一个 408(请求超时)的应答。UAC 处理 3xx 应答的时候，在这个应答是在对话内的请求的应答的处理方法和在对话外的处理方法是一样的。这个方法在 8.1.3.4 节中描述。需要注意的是，虽然 UAC 会尝试新的地址（处理 3xx 应答的时候），但是它依旧使用对话内的路由集合来构造请求的 Route 头域。

当 UAC 收到一个刷新 target 请求的 2xx 应答的时候，如果对话的 remote target URI 存在，那么它必须用这个应答的 Contact 头域的值来替换对话的 remote target URI。

如果对话那的请求的应答是 481 应答（呼叫/事务不存在 Call/Transaction Does Not Exits）或者一个 408（请求超时），那么 UAC 应当终止这个对话。并且 UAC 应当在请求完全没有应答的时候（客户端 transaciton 将会通知 TU 这个超时）客户端 transaction 终止这个对话。

对于 INVITE 初始化的对话，终止对话需要发送一个 BYE。

12.2.2 UAS 行为

在对话中发送的请求，就像其他请求一样，是原子请求。如果 UAS 收到某个请求，所有的相关状态要么一起改变，要么就一起不变。在某些请求中，请求会影响好几个状态（比如 INVITE 请求）。

UAS 从 transaction 层收到请求。如果请求的 To 头域有 tag 字段，UAS 的处理核心需要校验对话的 ID，拿请求中的 tag 和现存的对话相比较。如果匹配成功，那么就是一个在对话中的请求。在这种情况下，UAS 首先使用 8.2 节中的对话外请求处理的步骤。如果请求 To 头域包括了一个 tag 字段，但是对话的 ID 并不匹配现存的对话，UAS 可能是因为崩溃而重新启动，或者收到了一个另外（可能是错误的）UAS(UAS 可以构造 To 的 tags，这样 UAS 在灾备恢复下，可以把这个 tag 看成它自己的)。还有一种简单的可能是请求发送错误了。在这个基础上，UAS 可以选择接受或者拒绝请求。在允许的情况下，尽量处理这些请求会提供灾难恢复的机制。UAS 如果希望支持这样的特性就必须遵循一些原则，比如用始终使用单调递增的 Cseq 序列号，甚至是在重新启动之后也这样，在重启动后重建路由集合，处理越界的 RTP 时间戳和序列号等等。

如果 UAS 由于不希望重构对话而拒绝这个请求，它必须应答对方一个 481（呼叫/事务不存在。Call / Transaction 不存在）应答。

对于在对话中接收到的，那些不会用任何形式更改对话状态的请求，比如 OPTIONS 请求，他们等同于在对话外的处理请求。

如果远端的序列号（remote sequence number）是空的，它必须设置成为请求中的 Cseq 头域的序列号（sequence number）。如果 remote sequence number 不是空的，但是请求中的 sequence number 小于这个 remote sequence number,请求就是非顺序的，并且必须通过应答 500（服务器内部错误）打回去。如果 remote sequence number 不是空的，并且请求中的序列号大于这个 remote sequence number,请求就是按照顺序的。这个请求中的 Cseq 的序列号可以比 remote sequence number 大不止 1。在这种情况下，并非是错误的，并且 UAS 应当准备接收和处理比上次处理的请求 Cseq 值大于 1 的

请求。UAS 必须设置 remote sequence number 成为请求中的 Cseq 头域中的序列号。

如果一个 proxy 废弃掉一个 UAC 产生的请求，并且 UAC 重新递交这个请求的时候。这个请求是会具有一个全新的 Cseq 序列号。UAS 是不会收到第一个请求的，这样，Cseq 序列号就会出现间隔，这样的间隔并非是一种错误的情况。

当 UAS 接收到一个 target 刷新请求的时候，如果请求中存在 Contact 头域，它必须用 Contact 头域中的 URI 来替换对话的 remote target URI。

12.3 终止对话

在建立对话中的终结对话，跟请求方法无关，如果对话外的请求产生了一个非 2xx 终结应答，任何前边请求创建的”早期对话”(early dialogs)将会终止。在已经建立的对话中，终结对话就是请求方法相关的。在这个定义中，BYE 方法将会终结一个对话。15 节有细致的讨论。

13 初始化一个会话

13.1 概览

当 UAC 希望初始化一个会话（比如，audio,video 或者游戏），它首先构造一个 INVITE 请求。这个 INVITE 请求一个服务器来建立一个会话。这个请求可能会由 proxy 层层转发，最后到达一个或者多个可能能够处理这个邀请的 UAS。这些 UAS 需要看看是否用户接收这个邀请。然后 UAS 可以接收这个请求（也就是会话建立了），通过发送 2xx 应答。如果邀请被拒绝，根据拒绝的原因，3xx，4xx，5xx 或者 6xx 应答将会发送。在发送终结应答之前，UAS 可以发送一些临时应答（1xx）应答给 UAC，以便 UAC 能够掌握建立会话的进度。

当收到了一个或者多个临时应答，UAC 可能收到一个或者多个 2xx 应答或者一个非 2xx 终结应答。由于在 INVITE 终结应答之前，可能有不少时间，在 INVITE

事务的可靠性机制和其他的请求不同（比如 OPTIONS）。当 UAC 收到了终结应答，UAC 需要给每一个 INVITE 的终结应答，发送一个 ACK 请求。发送 ACK 请求的步骤依赖于应答的类别。对于在 300 到 699 的终结应答，ACK 是在 transaction 层处理的，并且遵循一系列规则（17 节）。对于 2xx 应答，ACK 是由 UAC 处理核心产生的。

INVITE 的一个 2xx 应答会建立一个会话，同时也建立了一个基于发送 INVITE 请求的 UA 和产生 2xx 应答的 UA 之间的对话。因此，当从多个远程 UA 收到了多个 2xx 应答（可能由于 INVITE 的分支），每一个 2xx 建立一个不同的对话（dialog）。所有这些对话都是同一个呼叫的组成部分。

本节介绍了 INVITE 请求建立会话的详细过程。支持 INVITE 的 UA 也一定同时支持 ACK,CANCEL 和 BYE。

13.2 UAC 处理

13.2.1 创建一个初始化的 INVITE

由于初始化的 INVITE 请求是一个对话外的请求，它遵循 8.1.1 节的步骤创建。除此之外还有专门针对 INVITE 的附加处理步骤。

在 INVITE 中应当包括一个 Allow 头域（20.5 节）。它用来标志在这个 INVITE 建立的这个对话（dialog）中什么样的方法可以接受。比如，一个 UA 可以在对话中接收和处理 INFO 请求[34]，那么在 INVITE 请求的 Allow 头域中应当列出这个 INFO 方法。在 INVITE 请求中，Supported 头域应当包含。这个头域包含了所有这个 UAC 支持的扩展部分。

在 INVITE 中可以包含一个 Accept 头域（20.1 节）。这个标志了 UA 在后续建立的对话中，能兼容的接收和发送的 Content-Type。Accept 头域支持不同会话描述格式（session descrioption format）的时候特别有用。

UAC 可以通过包含一个 Expire 头域(20.19 节)来限制邀请的有效期限。如果 Expire 头域的时间到了还没有接收到 INVITE 的终结应答，UAC 处理核心应当像 9 节描述的那样产生一个对 INVITE 请求的 CANCEL 请求，

UAC 还可以根据需要增加 Subject(20.36 节)，Organization(20.25 节)和 User-Agent(20.41 节)头域。这些头域都包含了 INVITE 的相关资料。UAC 可以给 INVITE 增加一个消息体。8.1.1.10 节讲述了如何构造 Content-Type 头域来描述消息体。

对于消息体，有一些特别的规定——他们是基于某种磋商机制的，他们对应的 Content-Disposition 是”session”(会话的)。SIP 使用一个请求/应答模型，UA 发出一个会话描述，称作是请求，里边包含了会话的描述。这个请求标志了特定的联系内涵（比如 audio，vidio，game），这些内涵的参数（比如解码器等等），并且从应答方接收媒体信息的地址。对方 UA 会回应另外一个会话的描述，称之为应答，标志了能接受的联系内涵，这些内涵的参数。这个请求/应答的交换实在对话的上下文进行中的，所以如果一个 SIP INVITE 请求导致了多个对话，每一个对话都包含自己独立的请求/应答的交换。请求/应答模型定义了对于请求和应答的限制。（比如在上一个请求尚未处理完成情况下不能发起下一个请求）。这也导致了请求/应答在 SIP 消息中出现的位置限制。在这个规范中，请求和应答只能出现在 INVITE、ACK 请求和其应答中。请求和应答的使用中更进一步被限制。在初始化一个 INVITE 事务中，规则如下：

o 初始化请求必须在 INVITE 中，如果不在 INVITE 请求中，就必须在 UAS 回送给 UAC 的第一个非失败的可靠消息中。在这个规范中，这个应答就是 2xx 应答。

o 如果初始的请求是一个 INVITE，那么应答必须是由 UAS 发送回给对应发出 INVITE 请求的 UAC 的可靠的非失败的消息。在本规范中，只有 2xx 应答对应这个 INVITE 请求。同样相同的应答可能在之前发送的零食应答中存在。UAC 必须把它接收到的第一个会话描述当作是应答，并且必须忽略任何在初始 INVITE 请求中后续的会话描述应答描述。

o 如果初始请求是在第一个可靠的非失败的 UAS 回送给 UAC 的消息中，那么应答必须在这个消息的确认消息中（在本规范中，就是给 2xx 应答的 ACK 确认消息）

o 在发送或者接收到第一个请求的应答之后，UAC 可以同样依据这样的问答方法产生后续的请求。但是只能在收到每一个请求的应答之后才能发起下一个请求。不能在上一个请求尚未收到应答的时候发起下一个请求。
o 当 UAS 发送或者接收到初始化的请求的时候，禁止在它给初始的 INVITE 请求的应答中产生后续的请求（协商会话描述请求）。这就意味着基于本规范的 UAS 在完成初始化的事务之前，不会产生任何会话描述请求。

具体来说，根据本规范，上边的规则分别定义了两种 UA 之间交换信息的方法。请求实在 INVITE 中，应答是在 2xx（可能在 1xx 中也存在，具备相同的值）中，或者请求在 2xx 中，应答在 ACK 中。（这个意思是说，两个 UA 之间建立连接的时候，首先需要协商一下两个 UA 能够支持的消息体正文，那么这个协商关系也是通过问答形式的，也就是通过请求/应答的，这个媒体磋商的请求既可以在 UAC 发起的 INVITE 请求中，也可以在 UAS 回应的 2xx 应答中。同样的，媒体磋商的应答既可以在 UAS 的 2xx 应答或者 1xx 应答中，也可以在 ACK 确认请求中）。
所有的支持 INVITE 请求的 UA 都必须支持两种交换方式。会话描述协议（session description protocol sdp）(RFC 2327[1])在所有的 UA 中都必须得到支持，并且它的用法和请求/应答的构造必须遵循[13]中定义的步骤。
在上边讲述的请求/应答模型中，只能适用于在包头域 Content-Disposition 中的值是”session”的包体情况。因此，有可能会 INVITE 和 ACK 请求中都包含一个包体信息（比如，INVITE 包含一个相片（Content-Disposition:render）并且 ACK 包含一个会话描述(Content-Disposition:session)）。
如果 Content-Disposition 头域不存在，Content-Type 是 application/sdp 的包体实现就等同于 Content-Disposition”session”，其他 Content-Type 的情况就是实现”render”。
当 INVITE 请求创建以后，UAC 遵循对话外请求发送的步骤进行发送（8 节）。这也就是创建一个客户事务并且由这个客户事务发送请求并且处理应答。

13.2.2 处理 INVITE 应答

当 INVITE 请求被传送给 INVITE 的客户事务层进行处理，UAS 等待 INVITE 的应答。如果 INVITE 客户事务层返回一个超时而不是收到一个应答，那么这个 TU 就应当像收到一个 408（请求超时）应答（8.1.3 节）那样进行处理。

13.2.2.1 1xx 应答

有可能在收到一个或者多个终结应答之前，UAC 会收到 0 个或者 1 个或者多个临时应答。INVITE 的临时应答会建立”early dialogs”（早期对话）。如果一个临时应答在 To 头域中有一个 tag 子顿，并且应答的 dialog ID 并不是已经存在的对话的 ID，那么就应当遵循 12.1.2 节定义的步骤创建一个对话（早期对话）。early dialog 只会在下边这个情况中需要：如果一个 UAC 需要在完成初始的 INVITE 事务之前，给对方发送一个对话内的请求的时候，就需要 early dialog。在临时应答中的头域可以在当对话是 early state 的时候都有效（也就是说，比如一个临时应答的 Allow 头域包含的方法，在对话状态是 early state 的时候都是有效的。[由于 Allow 是允许的方法集合，所以，当对话状态是早期对话的时候，这个 Allow 的集合是不会改变的，但是当创建正式的 dialog 之后，Allow 的集合可能会改变哦]。）

13.2.2.2 3xx 应答

一个 3xx 应答可能包含一个或者多个 Contact 头域值，这个头域值提供了被叫方可能存在的地点。UAC 可以根据 3xx 应答的状态码（21.3 节）来决定是否尝试这些新的地址。

13.2.2.3 4xx,5xx,6xx 应答

在 INVITE 请求中，可能会收到单个非 2xx 终结应答。4xx，5xx，6xx 应答如果包含了 Contact 头域，那么这个头域值指示了错误的详细信息的解释地点。后续的终结应答（只有可能在发生错误的情况下），必须被忽略掉。

所有的早期对话都会由于接收到非 2xx 终结应答而结束。

一旦接收到了非 2xx 终结应答，UAC 处理核心就认为 INVITE 事务结束了。

INVITE 客户事务处理生成对这个应答的 ACK（参见 17 节）。

13.2.2.4 2xx 应答

单个 INVITE 请求可能会导致多个 2xx 应答返回给 UAC，这是因为 proxy 可以分支。每一个应答都是由 To 中的 tag 参数来进行区分的，并且每一个应答都代表了一个独立的对话，具备单独的对话 ID。

如果在 2xx 应答中的对话 ID 和一个现存的对话匹配，那么这个对话必须切换到”confirmed”状态，并且对话的路由集合必须基于 2xx 的应答进行重新计算（参见 12.2.1.2）。如果不匹配，那么必须创建一个新的对话，这个对话具备”confirmed”状态，参见 12.1.2 的步骤进行创建。

注意在对话状态中，只有路由集合不需要重新计算。其他部分比如对话内的最大的序列号（远程的和本地的）等都不需要重新计算。路由集合只是由于需要向后兼容而需要重新计算。RFC 2543 并没有要求在 1xx 应答中反射 Record-Route 头域回来，只在 2xx 请求中要求了。我们不能更新对话状态的全部部分，因为在早期对话（early dialog）中可能会存在对话中的请求，比如更改序列号等等。UAC 核心必须为每一个 2xx 应答，产生一个 ACK 请求。除了在 Cseq 和身份认证相关的头域之外，ACK 请求的头域的创建和在对话中的请求创建的方法一样（12 节）。Cseq 头域的序列号部分必须和需要确认的 INVITE 请求一样，但是 Cseq 的方法部分必须是 ACK。ACK 必须包含和 INVITE 请求相同的信任状。如果 2xx 包含一个媒体磋商请求（基于上述的规则），ACK 必须在包体中包含一个媒体磋

商应答。如果 2xx 应答的媒体磋商请求不能被接收，UAC 核心必须产生一个有合法的应答 ACK，并且立刻发送一个 BYE 请求。

当 ACK 创建以后，[附件 4]中规定的步骤用来检测对方地址，端口和 transport。这个请求是直接交给通讯层进行通讯的，而不是交给一个客户事务层进行发送。这是由于 UAC 核心直接处理 ACK 的重发，而不是事务层进行重发的处理。每次收到一个重发的 2xx 终结应答的时候都必须发送一个 ACK 到通讯层。

UAC 核心认为 INVITE 事务在接收到第一个 2xx 应答后的 64×T1 秒后完成。在这个时间点后，所有没有转换成为建立连接状态的早期对话都会被终止。一旦 UAC 确认 INVITE 事务完成了，那么缺省认为不会收到新的 2xx 应答了。如果，在相应了对 INVITE 请求的全部应答之后，UAC 并不希望创建这个对话，那么 UAC 必须通过 15 节描述的那样发送 BYE 请求来结束对话。

13.3 UAS 处理

13.3.1 处理 INVITE

UAS 核心从事务层收到 INVITE 请求。首先根据 8.2 节定义的步骤进行处理请求，8.2 节中定义的是跟对话内外无关的请求的处理。如果处理顺利完成（没有产生应答），UAS 核心根据如下步骤进行额外处理：

1、 如果 INVITE 请求包含一个 Expires 头域，UAS 核心就设置一个时钟计数＝这个头域值。如果时钟到了，这个邀请就过期了。如果在 UAS 尚未产生终结应答的时候就超时了，那么 487（请求终止）应答应当产生给 UAC。

2、 如果请求是一个对话中的请求，12.2.2 节定义的方法无关的处理步骤将首先进行处理。这个处理可能会影响到会话；14 节讲述了细节。

3、 如果请求的 To 头域包含了一个 tag，但是对话的 ID 与现存的任何一个对话都不匹配，那么 UAS 可能是由于崩溃而重新启动的，或者是由于接收到了本应

当发送给另外一个 UAS 的请求（或者就简单是由于请求填写错误）。12.2.2 节提供了这种情况的处理指引。从这开始的处理将假定这个 INVITE 是在对话外的，并且 INVITE 请求的目的是建立一个新的会话。INVITE 请求可能包含一个会话描述，在这种情况下是希望和 UAS 进行会话媒体的磋商。即使 INVITE 请求是对话外发出的，这个 INVITE 参与的用户也有可能正是那个会话中的参与方。这个是由于在多方会议中，某个正在会议中的用户，被其他参与方邀请参加。如果需要鉴别这样的情况，UAS 可以使用会话描述来检查是否重复邀请。比如，SDP 包含了会话的 ID 和版本号。如果这个用户本身就是会话中的一方，并且 session 参数包含的会话描述没有改变，UAS 可能就悄悄接受这个邀请（就是说，在不提示用户的情况下发送 2xx 应答）。

如果 INVITE 并没有包含某个会话描述（session description），UAS 就是被邀请创建一个会话，并且 UAC 已经希望 UAS 来提供这个会话 offer。UAS 必须在它的给 UAC 的第一个非失败的可靠消息中提供这个 offer。在本规范中，给 INVITE 请求的 2xx 应答中就应当提供这个 offer。

UAS 可以提示进度，接受，转发，或者拒绝这个邀请。在这些情况下，它通过按照 8.2.6 节描述的步骤建立应答。

13.3.1.1 提示进度

如果 UAS 不能马上接受或者拒绝邀请，那么它可以提示某种形式的进度给 UAC（比如提示一个回铃声等等）。这是通过一个 101 到 199 的临时应答实现的。这些临时应答建立了早期对话（early dialog）（通过 8.2.6 和 12.1.1）。如果 UAS 愿意，UAS 可以发送多个临时应答。每一个临时应答都必须包含相同的 dialog ID。这些临时应答都并非可靠传送的。

如果 UAS 打算延长一点时间来响应这个 INVITE 请求，它需要请求一个”extension”来防止 proxy 来取消这个事务。proxy 有权利来取消超过 3 分钟未完成的事务。要防止这个取消，UAS 必须每分钟发送一个非 100 临时应答，防止由于 1xx 临时应答的非可靠传输导致的临时应答丢失。

如果呼叫出于等待状态（比如用户设置成为呼叫等待的）或者这个呼叫正在和PSTN 电话系统进行通讯（PSTN 系统允许呼叫没有应答），一个 INVITE 事务是可以被延长处理时间的。

13.3.1.2 INVITE 请求转发

如果 UAS 决定转发这个呼叫，就需要发出 3xx 的应答。300（多重选择），301（永久转移），302（临时转移）应答中应当包含一个 Contact 头域，这个头域包含了一个或者多个表明需要重试的 URI 新地址。这个应答交给 INVITE 服务端事务层，由服务端事务层负责应答的重发。

13.3.1.3 INVITE 请求的拒绝

拒绝 INVITE 请求的常见情景是被叫方不想或者不能在终端系统上接收这个呼叫。486（用户忙）应当在这样的情况下返回。如果 UAS 知道没有其他终端系统能够响应这个呼叫，就应当返回一个 600（Busy Everywhere）。不过，通常情况下 UAS 是不太会知道这个情况的，并且这个应答也是罕见的。这些应答是交给 INVITE 服务端的事务层进行发送的，由这个事务层来保证应答的重发机制的。如果 UAS 拒绝的是 INVITE 请求包含的媒体磋商 offer，UAS 应当返回一个 488（Not Acceptable Here）应答。这个应答应当包含一个 Warning 头域来解释为何 offer 被拒绝。

13.3.1.4 接受 INVITE 请求

UAS 核心产生一个 2xx 应答。这个应答建立一个对话，然后遵循 8.2.6 节和 12.1.1 节的描述进行处理。

响应 INVITE 请求的 2xx 应答包含 Allow 头域和 Supported 头域，并且可能包含 Accept 头域。包含这些头域的目的是为了让 UAC 不需要再次请求就能够知道 UAS 的特性以及 UAS 的扩展支持。

如果 INVITE 请求包含了一个媒体磋商请求 offer，并且 UAS 还没有发送应答，2xx 应答中必须包含针对这个 offer 的应答。如果 INVITE 请求没有包含这个 offer，而且 UAS 也尚未发出 offer，2xx 应答必须包含这个媒体磋商 offer。

当应答构建好了以后，它会交给 INVITE 的服务端事务层进行发送。注意，INVITE 的服务端事务将会由于收到这个终结应答并且交给通讯层进行发送而销毁。因此，有必要在没有收到 ACK 的时候，每隔一定的时间就直接交给通讯层进行发送。2xx 交给通讯层进行发送的时间间隔是从 T1 秒开始，并且每次发送后就加倍，直到到达 T2 秒的时间间隔（T1 和 T2 的时间间隔定义在 17 节）。当收到了针对这个应答的 ACK 请求之后，重发就终止了。这个是与使用什么通讯协议来发送这个应答是无关的。

由于 2xx 的重发是端到端的，并且在 UAS 和 UAC 之间存在采用 UDP 通讯的节点。所以要保证通过这些节点进行可靠的传送，就必须采用间隔时间重发的机制，哪怕 UAS 本身的通讯机制是可靠的。

如果服务端的对 2xx 应答的重发经过了 64×T1 秒还没有收到 ACK 请求，那么 dialog 就认为是 confirmed，但是会话却应当终止。这个是用过 15 节描述的方法发送 BYE 请求来结束。

14 更改已经存在的会话

一个成功的 INVITE 请求（13 节）既会创建一个基于两个用户之间的对话，也会基于请求/应答模式（offer－answer）创建一个会话。12 节讲述了如何通过 target refresh 请求来修改一个现存的会话（比如，修改对话的 remote target URI）。本节描述如何修改实际的会话（session）。

这个修改可以包括修改地址或者端口、增加媒体流、删除媒体流等等。这是通过发起新的 INVITE 请求来完成的，并且这个新的 INVITE 请求是基于建立会话所相同的对话的。在一个现存对话中发出 INVITE 请求就是 re-INVITE.

注意，单个的 re-INVITE 请求可以同时更改对话和会话的参数。

呼叫方或者被叫方都可以更改现存的会话。

在本协议中，UA 检测本地媒体有效性是基于自身的策略的。但是，我们并不建议自动产生 re-INVITE 或者 BYE 请求，因为这样可能会导致网络上的阻塞。在任何情况下，如果某些消息将被自动发送，那么他们应当等待一个随机的时间间隔。
注意，上边的这些描述是特指自动产生的 BYE 和 re-INVITE。如果用户由于媒体不兼容而挂机，UA 应当正常发出 BYE 请求，而不视为自动产生的 BYE。

14.1 UAC 行为

与 INVITE 相同的会话描述磋商 offer-answer 模式（13.2.1 节）在 re-INVITE 中也一样采用。假设 UAS 希望增进一个媒体流，那么 UAC 将会创建一个新的 offer 包含这个媒体流，并且发送 INVITE 请求给他的对方。特别需要注意的是，这个会话的全描述，而不是变化部分需要传送。这个支持无状态的会话处理，并且支持错误恢复机制。当然，UAC 可以发送一个 re-INVITE 请求而不包含会话描述，在这样的情况下，就是在这个 re-INVITE 的第一个可靠的非失败的应答中将会包含这个会话描述 offer（在这个规范中，就是 2xx 应答）。
如果会话描述格式具有版本号码，那么这个磋商的 offer 应当标志这个变化了的媒体描述版本。
re-INVITE 请求中的 To,From,Call-ID,Cseq,Request-URI 头域应当和正常的在对话中的请求构造方法一样（12 节）。
在 re-INVITE 请求中，UAC 可以选择不增加一个 Alert-Infor 头域或者具有 Content-Disposition＝”alert”的消息体。因为 UAS 通常不会要求提示操作者来响应这个 re-INVITE 请求。
和 INVITE 不同的是，INVITE 可以分支（分岔成为多份 INVITE）,re-INVITE 是不会分支的，所以，只会由一个单个的终结应答。re-INVITE 不会分岔的原因是因为 Request-URI 标志的是建立对话的 UA 的目标地址，而不是用户的 address-of-record 地址。
需要注意的是，在相同的对话中，UAC 不能在上一个 INVITE 请求完成前（无论是那一方发起的 INVITE）再次发起一个新的 INVITE。

1、 如果有正在处理的 INVITE 客户事务，TU 必须等待这个事务终结或者完成，才能初始化一个新的 INVITE。

2、 如果有正在处理的 INVITE 服务事务，TU 必须等待这个事务确认或者终结，才能开始处理一个新的 INVITE。

不过，UA 可以在 INVITE 事务正在处理的同时，处理一个普通的事务。也可以在一个普通事务正在处理的同时来初始化一个 INVITE 事务。如果 UA 接收到一个针对 re-INVITE 的非 2xx 终结应答，则会话参数不能改变，应当就像没有收到过这个 re-INVITE 请求一样。注意，就像在 12.2.1.2 节一开始讲的那样，如果非 2xx 终结应答是一个 481（Call/Transaction Does Not Exists）,或者一个 408（Request Timeout）,或者完全没有 re-INVITE 请求的应答（也就是说从 INVITE 客户事务端返回一个超时），UAC 会终止这个对话。

如果 UAC 收到一个 re-INVITE 的 491 应答，他应当启动一个值为 T 的时钟，这个 T 的取值如下：

1、 如果 UAC 是这个 dialog ID 的 Call-ID 的拥有者。（也就是说是 UAC 产生的 Call-ID）,那么 T 取值为一个 2.1 到 4 秒的随机数，单位是 10 毫秒。

2、 如果 UAC 并非是 dialog ID 的 Call-ID 的拥有者，T 应当取值是 0 到 2 秒的随机数，单位是 10 毫秒。

当这个时钟到了，如果 UAC 还希望再次尝试更改会话参数，UAC 应当再次尝试 re-INVITE 请求一次。这个意思是说，如果这个呼叫已经被 BYE 所挂掉了，那么 re-INVITE 请求就没有再发的必要。

发送 re-INVITE 请求的规则，以及针对 re-INVITE 请求产生的 2xx 应答而产生的 ACK 请求的发送规则，等同于初始的 INVITE 请求（13.2.1 节）。

14.2 UAS 行为

13.3.1 节描述了区分初始的 INVITE 请求和 re-INVITE 请求的方法，以及处理对现存的对话中处理 re-INVITE 请求的步骤。

UAS 在发送第一个 INVITE 的终结应答之前，收到第二个 INVITE 请求，并且这个请求的 Cseq 序列号大于第一个 INVITE 请求，那么就应当给第二个 INVITE 请求返回一个 500（服务器内部错误）应答，并且必须包含一个 Retry-After 头域，这个头域中应当包含一个 0—10 秒的随机数。

如果 UAS 正在处理一个 INVITE 请求的时候又收到了一个在同一个对话上的 INVITE 请求必须返回一个 491（Request Pending）应答给接收到的 INVITE。

如果 UA 接收到一个对现存的对话的 re-INVITE 请求，那么就必须检查有关会话描述(session description)的版本标志（version identifiers）,或者，如果没有版本标志，那么就需要检查会话描述的正文看看有没有变化。如果会话描述改变了，UAS 必须由此调整会话参数，在调整参数的时候可能会要求用户确认。

会话描述的版本可以用来提供给会议的新近加入者，增加或者删除媒体，或者由单点会议更改成为多方会议。

如果新的会话描述是不能被 UA 接受的，UAS 可以用返回一个 488（Not Acceptable Here）来拒绝这个 re-INVITE 请求。这个响应应当包含一个 Warning 头域（用来提供给请求方，提供这个拒绝的原因）。如果 UAS 产生一个 2xx 应答，但是没有收到 ACK,它应当产生一个 BYE 来结束这个对话。

UAS 可以不产生 180（Ringing）应答给 re-INVITE，因为 UAC 一般不展示这个信息给用户。同样的，UAS 可以增加一个 Alert-Info 头域或者一个由 Content-Disposition“alert”的消息体来应答给 re-INVITE 来让 UAC 展示这个信息。

如果 UAS 在 2xx 应答中提供了一个媒体磋商 offer（因为 INVITE 没有包含一个 offer），那么它应当当作新呼叫一样的构造这个 offer，就像在[13]中 SDP 描述一样，遵循发送更改现有的会话的 offer 的限制。特别需要注意的是，这个意味着它应当包含全部的 UA 支持的媒体类型和媒体格式。UAS 必须确保会话描述在媒体格式，通讯方式，或者其他要求对方支持的参数上，新的会话描述和旧的会话描述一样。这个可以避免对方拒绝这个会话描述。如果，UAC 任然是不能接受这个

会话描述，UAC 应当产生一个它能够接受的会话描述应答，并且发送一个 BYE 来结束这个会话。

15 结束一个会话

本节描述了结束由 SIP 建立的会话的步骤。会话的状态和对话的状态是密切相关的。当一个会话由 INVITE 建立的时候，每一个由不同 UAS 的 1xx 或者 2xx 的应答创建一个对话，并且当完成了会话描述的请求/应答（offer/answer）交互之后，它也就创建了一个会话。这就是说，每一个会话都和单个对话”相关”—会话是对话所创建的。如果初始化的 INVITE 产生了非 2xx 的终结应答，它也终结了由本次请求创建的任何会话（如果有的话），并且终结了所有的本次请求创建的对话（如果有的话）。由于事务完整性的保证，一个非 2xx 的终结应答同样也防止了本次 INVITE 以后可能创建的会话。BYE 请求用于终结指定的会话或者尝试建立的会话。在这里，特定的会话是一个和与之相对的对话的对方 UA。当在对话中接收到了一个 BYE,任何与该对话相关的会话都应当终止。UA 禁止在对话外发送 BYE 请求。请求方 UA 可以在已经建立好的对话或者早期对话中发起 BYE 请求；被叫方只能在建立好的对话中发起 BYE 请求，不能在早期对话中发起 BYE 请求。不过，在一个建立好的对话中，被叫方的 UA 不能在接收到对应 2xx 应答的 ACK 请求前发送 BYE 请求，或者不能在服务器事务超时前发送 BYE 请求。如果没有 SIP 扩展定义了和这个对话相关的其他应用层状态，这个 BYE 请求同样结束了对话。

在对话和会话中，给 INVITE 的非 2xx 的终结应答，使得使用 CANCEL 比较有吸引力。CANCEL 是尝试强制给 INVITE 请求一个非 2xx 应答（比如，487 应答）。因此，如果 UAS 希望放弃整个呼叫，它可以发送一个 CANCEL。如果 INVITE 会有 2xx 终结应答，这个意味着 UAS 在 CANCEL 正在处理的时候，接收到一个邀请。UAC 可以继续用这个 2xx 应答建立会话，也可以用 BYE 终结这个会话。（这个意思是说，一般情况下，如果 UAC 希望 cancel 这个 INVITE 请

求，那么就会发出 CANCEL 请求，如果接收到了非 2xx 的终结应答，就意味着 CANCEL 掉了，但是如果接收到的还是 2xx 应答，就说明没有 CANCEL 掉，没有 CANCEL 掉呢，就可以选择继续建立会话，或者说发送一个 BYE 来终结会话）

在 SIP 中，并没有一个很好的”hangin up”(挂机中)定义。它属于一个用户界面的普通常见的细节。通常，当用户挂机，它意味着结束建立会话的尝试，并且终止所有已经建立的会话。对于呼叫方的 UA 来说，如果没有收到初始 INVITE 请求的终结应答，这个可能是产生对初始 INVITE 请求的一个 CANCEL 请求，并且收到终结应答之后给每一个建立好的对话发出一个 BYE。对于被叫方的 UA,就是很普通的 BYE；粗略来说，当用户（因为响应振铃）摘机，就会产生一个 2xx 应答，于是挂机会在收到 ACK 请求之后发送一个 BYE。这不是说在收到 ACK 之前用户不能挂机，这只是表达在用户的电话中的软件，需要保持一小会儿状态，来正确释放状态。如果某个 UI（用户界面）允许用户在不摘机的情况下拒绝呼叫，可以用 403（Forbidden）来作为 INVITE 的应答，在这样的情况下，BYE 就不能发送。

15.1 使用 BYE 请求终止一个会话

15.1.1 UAC 行为

BYE 请求就像其他在对话内的请求一样的构造，参见 12 节的描述。

当 BYE 请求创建好了之后，UAC 核心处理部分创建一个新的非-INVITE 客户端事务，并且用它来处理 BYE 请求。UAC 必须认为当 BYE 请求一发送到客户端事务，会话就结束了（因此也就停止发送或者接收媒体流）。如果 BYE 请求的应答是 481（Call/Transaction Does Not Exists）或者 408（Request Timeout）或者 BYE 请求压根没有应答（就是说客户端事务返回一个超时），UAC 必须认为会话和对话都已经结束。

15.1.2 UAS 行为

UAS 首先按照通用的 UAS 接收到请求的处理步骤进行 BYE 请求的处理（8.2 节）。UAS 核心处理部分接收到 BYE 请求以后，首先检查它是否和现存的对话匹配。如果 BYE 并不匹配现存的任何一个对话，那么 UAS 应当产生一个 481（Call / Transaction Does Not Exists ）应答，并且传送给服务器事务。

这个规则意味着如果 UAC 发送没有带 tag 标志的 BYE 请求会被拒绝。这个是一个对 RFC2543 的改动，RFC2543 允许 BYE 不带 tag 标志。

UAS 核心处理部分接收到 BYE 请求，并且发现和现存的对话匹配，那么它必须遵循 12.2.2 的步骤来处理请求。一旦处理完成，UAS 应当终止这个会话（因此停止发送和接收媒体流）。唯一一个可以选择不终止的情况是多方会话，在多方会话中参与者允许对话中的其他参与方终止他们自己的会话。不管是否终止会话中的参与方，UAS 核心处理都必须给 BYE 产生 2xx 的应答，并且必须由服务器的通讯层进行传输。

UAS 必须依旧响应在这个对话中接收到的未决的请求。我们建议用 487（请求终止）来给这些未决的请求以应答。

16 proxy 行为

16.1 概述

SIP 代理服务器是路由 SIP 请求到 UAS 的，并且路由 SIP 应答到 UAC 的。一个请求可能通过多个 proxy 到达 UAS。每一个都会作出路由决定，在发送给下一个节点前对请求做一点修改。应答会通过和请求相同的 proxy 路径，只是顺序是逆序的。

proxy 是一个 SIP 逻辑上的概念。当接收到一个请求，在做代理服务器之前，首先应该有一个部件来决定是否自身需要响应这个请求。例如，在作为代理服务器处理请求之前，首先判定请求可能是非法的或者请求需要一个信任状。这个元素可以使用任何合适的错误代码来响应这个请求。当直接应答请求的时候，这个元素（proxy）将作为 UAS 角色，并且必须遵循 8.2 节描述的 UAS 行为规范。

proxy 对于每一个新的请求来说，既可以作为有状态的也可以作为无状态的模式来处理。当作为无状态的处理模式的时候，proxy 就是简单的转发。它转发每一个请求下行到一个由请求所决定的目的地。并且简单转发从上行流取得的应答。一个无状态的 proxy 在处理完一个消息之后就会丢弃这个消息的相关资料。有状态的 proxy 会保留这些信息（尤其是事务信息），保留每一个接收的请求和每一个接收请求的应答的相关信息。它保留这些信息用于处理与这个请求相关的后续消息。一个有状态的 proxy 可以选择”分支”一个请求，路由它到多个地点。任何被路由到多个地点的请求都必须当作有状态的处理。在某些情况下，proxy 可以用有状态的通讯协议（比如 TCP）来转发请求，而不用自身成为事务有状态的。例如，proxy 可以简单转发请求从一个 TCP 连接到另外一个 TCP 连接，而不用自身作为有状态的，只要它放置足够的信息在需要转发的消息里，使得能够正确把应答发送到接收到对应请求的连接发送出去。如果在不同传输通讯协议之间进行请求的转发，那么就必须要求 proxy 的 TU 采用某种手段来保证可靠的从一个协议有状态的转到另一个协议。

有状态的 proxy 可以在处理请求中的任何时候转换成为无状态的，只要它不作任何可能导致不能无状态的操作（比如分支，比如产生 100 应答）。当做这样的转换的时候，所有的状态就只是简单的废弃掉。proxy 不应当发起一个 CANCEL 请求。

在作为无状态的或者有状态的时候，处理请求的步骤是一样复杂的。接下来的步骤是从有状态的 proxy 角度来些的。最后一节是讲述无状态 proxy 的区别。

16.2 有状态的 proxy

作为有状态的 proxy，它必须是一个纯粹的 SIP 事务处理引擎。它在这里的定义遵循 17 节讲述的服务端和客户端的事务处理规定。有状态的 proxy 有一个服务端事务，这个事务与一个或者多个客户端事务相关，这些客户端事务是由高层 proxy 处理元素产生的（图 3），这些高层 proxy 处理元素就是 proxy 处理核心。一个输入的请求是通过一个服务端事务来处理的。请求由服务端事务交给

proxy 处理核心进行处理。proxy 处理核心检查请求应当路由到哪里，选择一个或者多个下一个节点。每一个发往下一个节点的外发请求都由客户端事务进行处理。proxy 处理核心从客户端事务中得到请求的应答并且把他们的应答交给服务端事务进行发送。

有状态的 proxy 为每一个接收到的新的请求创建一个服务端事务。任何请求的重复都是由这个服务端事务来处理（参见 17 节）。proxy 处理核心必须遵循 UAS 的模式，发送一个直接临时应答（比如 100 trying）到这个服务端事务上（8.2.6 节）。因此，有状态的 proxy 不应当给非 INVITE 请求产生 100（trying）应答。

这是一个 proxy 的模型，并非软件实现。在实现上可以扩展并且复用用这个模型定义。

对于所有的新请求来说，包括那些未知方法的请求，proxy 处理请求必须：

1、 验证请求（16.3）

2、 预处理路由信息（16.4）

3、 决定请求的目的(targets)(16.5)

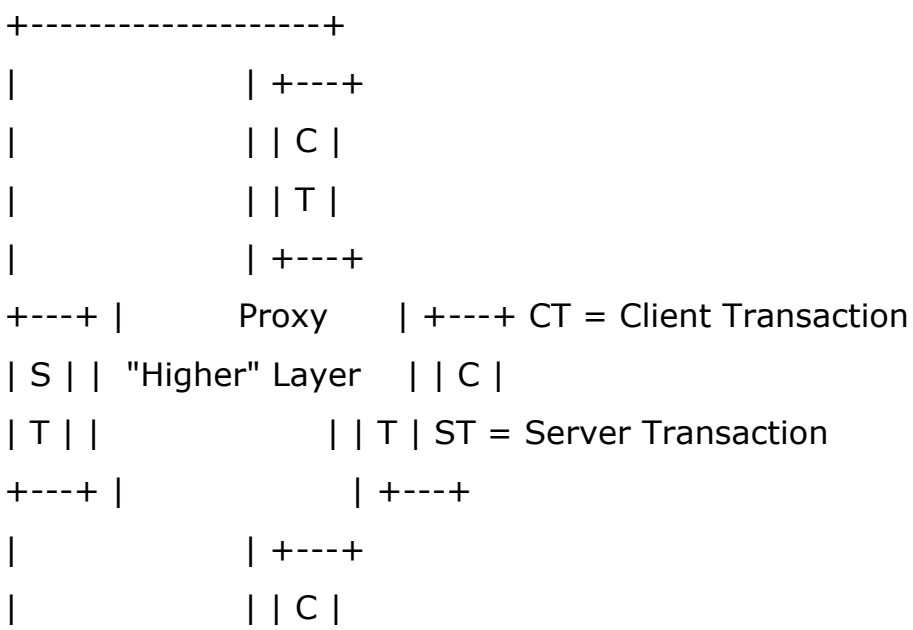


Figure 3: Stateful Proxy Model

4、 转发请求到每一个目的地（16.6）

5、 处理所有的应答（16.7）

16.3 验证请求

在 proxy 转发请求之前，它必须检查消息的合法性。一个合法的消息必须经过如下的检查：

1、 合法的语法

2、 URI scheme

3、 最大转发次数

4、 (可选)循环检测（loop detection）

5、 proxy-require

6、 proxy-authorization

如果任何一步失败了，proxy 都必须作为 UAS（8.2）一样，应答一个错误码。注意 proxy 并不要求检查合并的请求，并且不能把合并的请求当作一个错误的情况。终端接收到合并的请求会根据 8.2.2.2 节讲述的内容进行分解。

1、 合法的语法。

请求要能够被服务端事务处理，那么请求就必须是语法无误的。请求中的任何与检查相关的部分或者与请求转发节相关的部分都必须语法严格无误。在检查中，其他部分的严格与否，在检查中都被忽略，并且在转发消息过程中保持不变。例如，proxy 不会由于非法的 Date 头域而拒绝请求。同样的 proxy 不会在转发请求的时候移去非法的 Date 头域。这个是为了设计成为可扩展的。以后的扩展可能定义新的方法、新的头域。proxy 不能拒绝由于包含了不认识的方法或者不认识的头域而拒绝转发这个请求。

2、 URI scheme 检查

如果 Request-URI 包含一个 proxy 所不能理解的 URI 形式，那么 proxy 应当通过返回一个 416 来拒绝这个请求（Unsupported URI scheme）。

3、 最大转发次数检查

最大转发次数 Max-Forwards 头域（20.22）是用来限制转发的次数的。如果请求没有包含 Max-Forwards 头域，那么这个检查将被忽略。如果请求包含了一个 Max-Forwards 头域，并且这个头域大于 0，那么这个检查就跳过。如果请求包含一个 Max-Forwards 头域，并且这个头域为 0，那么这个 proxy 不能转发这个请求。如果请求是 OPTIONS 请求，那么 proxy 可以作为最终响应者来响应这个请求，并且遵循 11 节讲述的产生应答。否则 proxy 应当返回一个 483（too many hops）应答。

4、 可选的 Loop Detection 检查

proxy 可以检查看看转发是否可能导致循环。如果请求包含一个 Via 头域，并且这个头域值，和 proxy 早先保留的请求的头域值相同，那么这个请求就是以前经过过本 proxy。要么这个请求就是循环处理了，要么就是合法的循环处理。要检测请求是否循环处理，proxy 可以用 branch 参数，根据 16.6 节的第 8 步来计算，并且和接收到的 Via 头域做比较。如果参数相等，那么请求就循环了。如果不等，那么请求就是合理的经过，并且继续处理。如果检测到了循环，那么 proxy 应当返回一个 482（LoopDetechted）应答。

5、 Proxy-Require 检查

本协议的以后的扩展可能会要求额外的 proxy 特性。所以终端会在请求中包含一个 Proxy-Require 头域来表明会使用到那些特性，这样 proxy 就可以根据 Proxy-Require 判定自己是否能够支持这些特性。

如果请求包含一个 Proxy-Require 头域(20.29)并且有一个或者多个本 proxy 不能理解的 option-tags。那么这个 proxy 必须返回一个 420（Bad Extension）错误，并且这个错误应答必须包括一个 Unsupported(20.40)头域列明了那些 option-tags 这个 proxy 不能支持。

6、 Proxy-Authorization 检查

如果 proxy 要求在转发请求之前进行身份认证，那么必须根据 22.3 节中描述的那样进行请求的检查。22.3 节也定义了 proxy 应当怎样处理检查失败的情况。

16.4 路由信息预处理

proxy 必须检查请求中的 Request-URI 部分。如果 Request-URI 包含了一个本 proxy 早先放在 Record-Route 头域中的值（参见 16.6，4），proxy 必须用 Route 头域中的最后一个值来替换 Request-URI，并且从 Route 头域中删去这个值。proxy 必须接着按照个修改后的请求进行处理。

这个只会在某元素发送请求到 proxy（proxy 本身可能是一个终端），并且这个请求是基于严格路由的。在接收到请求后重写这个字段是必须的，因为要保持向后兼容性。同时也允许按照本规范实现的元素保护 Request-URI 通过严格路由的 proxy（12.2.1.1 节）。

这个要求并没有强制 proxy 保留状态来检查其早先放在 Record-Route 头域中的 URI。作为替换的方法，proxy 只需要保留足够的信息在那些 URI 里边，这样，在以后出现的时候就能识别了。

如果 Request-URI 包含了 maddr 参数，proxy 必须检查这个参数来看看是否在 proxy 配置的可信任的地址列表或者可信任的区域列表中。如果 Request-URI 包含一个 maddr 参数，并且这个参数包含了一个 proxy 可以信任的地址，并且这个请求是通过 Request-URI 中（指明或者缺省）的端口和协议接收到的，proxy 必须抽掉 maddr 和其他非缺省的端口和通讯参数，并且继续处理。

proxy 可能收到一个带有匹配 maddr 的请求，但是不是在 URI 中指出的端口和 transport 接收到的。这个请求需要通过指明的 port 和 transport 转发到对应的 proxy。

如果 Route 头域的第一个值就是这个 proxy，那么 proxy 必须从请求中把它移去。

16.5 确定请求的目的

接着，proxy 计算请求的目的（或者多个目的地）。目的地集合可以由请求的内容决定或者由绝对位置服务提供。目的地集合中的每一个目的地都由 URI 来表达。

如果请求中的 Request-URI 包含了 maddr 参数，必须把 Request-URI 放在目标集合中，并且是作为唯一一个目标 URI，并且 proxy 必须按照 16.6 节中的约定进行处理。

如果 Request-URI 的区域并非本 proxy 负责的区域，那么 Request-URI 必须放在目标集合中，并且作为唯一一个目标 URI，并且 proxy 必须按照 16.6 节中的约定进行处理。

有很多种情况都会导致 proxy 收到并非本 proxy 负责的区域的请求。一个防火墙 proxy 处理外发的请求（就像 HTTPproxy 处理外发的请求）就是一个典型的例子。

如果请求的目标集合没有像上边讲述的这样预先设定，那么这就意味着 proxy 是负责 Request-URI 所指明的区域的，并且 proxy 可以用任何机制来决定往哪里发送这个请求。这些机制都可以归结成为访问一个绝对位置服务的形式。这个可能由从 SIP 注册服务器创建的位置服务器获得信息、读取数据库、查阅服务器、利用其他协议、或者就简单的替代 Request-URI 来实现。当访问由注册服务器创建的位置服务的时候，在作为索引查询之前，Request-URI 必须首先根据 10.3 节进行规范处理。这些机制的输出结果将作为目的地集合。

如果 Request-URI 没有提供足够的信息来让 proxy 能够产生目的地集合，它应当返回一个 485（Ambiguous）应答。这个应答应当包含一个 Contact 头域包

含一些应当尝试的新位置。比如，一个到 sip:John.Smith@company.com 的 INVITE 可能在某一个 proxy 是不明确的，因为这个 proxy 有多个 JohnSmith。21.4.23 节有细节描述。

任何与这个请求有关的，或者与 proxy 当前环境有关的信息都可以用来构造目的地集合。例如，由于请求的内容不同或者包头域的不同，可以有不同的目的地集合，又或者不同时间到达的请求也可以有不同的目的地集合，或者不同的时间间隔，上一次失败的请求，甚至是当前 proxy 的利用率都可以导致目的地集合的不同。

通过这些机制，我们可以有一个可能的目的地列表，他们的 URI 被增加到目的地集合。每一个目的地只能在目的地列表中出现一次。如果目的 URI 已经在这个集合中存在了（基于 URI 类型的相等定义），那么它不能再次增加。

如果原请求的 Reuqest-URI 指明的区域并非本 proxy 所负责的区域，那么本 proxy 不能增加任何额外的目的地到目的地集合。

如果 proxy 负责 Request-URI 所指明的区域，那么这个 proxy 只可以在转发的时候改变请求的 Request-URI。如果 proxy 并非负责这个 URI，那么它不会在 3xx 或者 416 应答的时候查生递归。

如果原始请求的 Request－URI 是属于本 proxy 负责的区域的，那么 proxy 可以在请求转发的时候增加目的地。他可以在处理过程中，用任何可以获得的信息来决定新的目的地。 例如，proxy 可以选择把一个转发应答（3xx）所包含的联系地址合并到目的地集合中。如果 proxy 使用一个动态的信息源来构造目的地集合（例如访问 SIP 的注册服务器），它应当在处理请求的过程中监测这个信息源。当有新的目的地出现的时候，就应当加到这个目的地集合里边。就像上边说得，每一个 URI 只能在集合中出现 1 次。

只能出现 1 次的原因是可以降低网络冲突，在合并重定向请求的联系地址的情况下可以防止无限递归的出现。

举例来说，一个简单的位置服务是一个”no-op”(无操作的)，返回的目的 URI 就是输入的请求 URI。请求将送到一个特定的下一个节点 proxy。在 16.6 节/6 小节定义请求转发中，通过 SIP 或者 SIPS URI 表达的，下一个节点的身份， 将在 Route 的头域最上一层插入。
如果 Request-URI 是这个 proxy 所负责的，但是在本 proxy 中找不到，那么 proxy 必须返回 404（Not Found）应答。

如果目的集合经过上边的处理依旧是空的，那么 proxy 必须返回一个错误应答，这个错误应答应当是 408（暂时不可用）。

16.6 请求转发

当目的地集合不是空的时候，proxy 可以开始转发这个请求。有状态的 proxy 可以按照任意的顺序处理这个目的地集合。它可以顺序处理多个目的地，上一个完成前下一个不能开始。也可以采用并行的处理多个目的地。也可以通过分组的形式，每组之间是串行的，组内是并行的。

通常的处理顺序机制是使用一个 Contact 头域的 qvalue 参数来处理（20.10 节）。目的地从最高的 qvalue 开始处理到最低的 qvalue。相同 qvalue 的目的地可以并行处理。

有状态的 proxy 必须包含针对目的地集合的一个接收到应答和转发出去的原始请求进行匹配的机制。为了完成这样的目的，这个机制是一个由 proxy 层在转发第一个请求前创建的”response context”(应答上下文)来保障的。

对于每一个目的地，proxy 转发请求都遵循下列步骤：

1、 拷贝一个接收到的请求

2、 更新 Request-URI

3、 更新 Max-Forwards 头域

4、 可选增加一个 Record-Route 头域

5、 可选增加附加的头域

6、 路由信息后处理

7、 决定下一个节点地址、端口、通讯协议。

8、 增加一个 Via 头域值

9、 如果需要，增加一个 Content-Length 头域

10、 转发这个新的请求

11、 设置定时器 C

下面详细介绍每一步。

1、拷贝请求

proxy 首先把接收到的请求做一个拷贝。拷贝必须包含接收到的请求的全部头域。在接下来的处理步骤中未提及的头域不能删除。拷贝应当保留接收到的请求的头域的顺序。proxy 不能用合并的域名来进行域值的重新排序（参见 7.3.1）。proxy 不能增加、修改、删除消息体。

实际上，在实现中并非只是做一个拷贝；首要的事情是为每一个下一个节点准备一个相同的请求。

2、Request-URI

在拷贝好的请求中的 Request-URI 必须用目的地的 URI 进行替换。如果这个目的 URI 包含任何在 Request-URI 中所不能允许的参数，那么这些参数必须被删去。

这个步骤是 proxy 的本质步骤。proxy 通过这个机制来把请求转发到目的地。在某些情况下，接收到的 Request-URI 会不作更改的添加到目的地集合中。对于这样的目的地来说，上边讲的替换就等于是没有任何操作。

3、Max-Forwards

如果拷贝的头域包含一个 Max-Forwards，proxy 必须把这个域值减一。

如果拷贝的头域没有包含一个 Max-Forwards 头域，proxy 必须自己增加一个头域，缺省值是 70。现在有一些 UA 不会在请求中填写 Max-Forwards 头域。

4、Record-Route

（假设 proxy 接收到的这个请求会创建一个对话的情况下），如果希望保留这个请求创建的对话中，后续的请求依旧是要经过本 proxy，那么本 proxy 必须增加一个 Record-Route 头域值在这个拷贝中，并且增加的这个头域值应当是在其他现存的 Record-Route 头域之前。通过请求建立的对话可以包含一个预置的 Route 头域。

如果这个请求已经是一个对话的一部分，proxy 如果希望以后这个对话的请求依旧经过本 proxy，那么 proxy 应当增加一个 Record-Route 头域值。在 12 节描述的普通的终端操作中，这些 Record-Route 头域值不会对终端使用的路由集合造成任何影响。

如果请求本身已经在对话中的话，如果 proxy 不增加一个 Record-Route 头域在请求的包头，后续的请求也会经过本 proxy。但是，如果当终端中断并且重新构造这个对话的时候，本 proxy 就会从对话所经过的节点中删去。

一个 proxy 可以在任何请求中增加这个 Record-Route 头域值。如果请求并没有初始化一个对话，终端将会忽略这个头域值。12 节讲述了终端如何使用 Record-Route 头域来构造 Route 头域的。

在请求路径上的每一个 proxy 都是独立的决定是否增加一个 Record-Route 头域值的－在请求的 Record-Route 头域上的值并不影响这个 proxy 决定增加还是不增加 Record-Route 头域值。

在 Record-Route 头域中防止的 URI 必须是 SIP 或者 SIPS URI。这个 URI 必须包含一个 lr 参数（参见 19.1.1）。这个 URI 可以和请求将被转发的地方不同。这个 URI 不应当包含通讯参数，除非该 proxy 确认在后续请求将会经过的下行节点中，都支持这个通讯参数（比如本地网络等等）。

本 proxy 提供的这个 URI 可能会让其他元素（其他 proxy）作出路由决定。本 proxy，通常，并不知道其他节点的处理能力，所以，它必须严格律己，让自己遵循规范的 SIP 实现：SIP URI 和 TCP 或者 UDP 通讯协议。

在 Record-Route 中的 URI 必须指向插入它的元素（或者替代元素），这个意思是说通过附件[4]的服务器定位步骤可以顺利找到这个元素，这样后续的请求才能顺利到达同一个 SIP 元素。如果 Request-URI 包含一个 SIPS URI，或者 Route 头域的最上的值（经过后续第 6 步的处理）包含一个 SIPS URI，那么插入 Record-Route 头域的值必须是一个 SIPS URI。而且，如果请求不是基于 TLS 接受的，那么 proxy 必须增加一个 Record-Route 头域。在相似的情况下，proxy 如果从 TLS 上接收的请求，但是产生的是一个在 Record-Route 中或者 Route 头域最上值中没有 SIPS URI 的请求（在第 6 步后处理之后），必须在 Record-Route 头域中增加一个非 SIPS URI。

在安全范畴内的 proxy 必须在对话中保持这个安全范畴。

当 Record-Route 头域的 URI 在应答中又重新到达的时候，如果这个 URI 值需要重写的时候，这个 URI 必须是能够唯一确定的 URI。（就是说，请求可能会经

过这个 proxy 好几次，造成一个或者多个 Record-Route 头域值的增加）。16.7 节的第 8 步提供了一个能够让这个 URI 唯一的一个机制。

这个 proxy 可以在 Record-Route 头域中增加一些参数。这些参数在某些请求的应答中会被反射（echo）回来，比如给 INVITE 请求的 200（OK）应答。通过在消息的参数中保持状态比在 proxy 中保持状态更加有效。

如果 proxy 想在任何类型的对话中都保持在请求的路径上（比如在跨越防火墙的对话中），它需要给每一个接收到的请求中，都增加 Record-Route 头域，即使是它所不能理解的方法的请求也要增加，因为这些方法可能是对话相关的，具有对话语义的方法。

在 Record-Route 头域中增加的 URI 只是在当这个请求创建对话的时候有效。举一个例子，对于一个对话－有状态的 proxy（dialog-stateful proxy），当在对话结束后，如果再收到一个请求，这个请求的 Request-URI 的值中包含这个 URI，那么它就可以选择拒绝这个请求。一个对话状态无关的 proxy，当然，没有对话结束的概念，但是他们可以再这个值中填写足够多的信息，这样就可以在以后的请求来到的时候做对话的 ID 的比较，并且可以选择拒绝不匹配这个信息的请求。终端不能在对话外使用这个对话中的 Record-Route 头域的 URI。参见 12 节描述的终端使用 Record-Route 头域的细节。

当 proxy 需要查看所有对话中的消息的时候，我们就需要 Record-routeing。但是，他会降低处理性能和影响扩展性，因此 proxy 应当只在特定情况下使用 record-route。任何初始化一个对话的 SIP 请求都可以适用 Record-Route。在本文档中，只有 INVITE 请求是可以适用的，以后的扩展文档可能包含其他的方法。

5、增加附加的头域

在这一步，proxy 可能增加其他适当的头域。

6、处理路由信息

proxy 可以有一个本地的策略，这个策略要求请求在传递到目的地之前，必须经历一个 proxy 集合。这样的 proxy 必须能够确保所有的类似的 proxy 都是松路由（loose routers）的。通常，只有当这些 proxy 都是在相同的区域管理的时候，我们才可能知道这些 proxy 是否都是松路由的。这个 proxy 的集合是通过一组 URI 的集合表示（每一个都包含一个 lr 参数）。这个集合必须被放置到 Route 头域中，并且放置在其他头域值之前。如果 Route 头域不存在，必须增加一个 Route 头域，包含这组 URI 的列表。

如果 proxy 有一个本地策略要求请求经过一个指定的 proxy，在压栈 Route 头域之外的一个方法就是旁路下边的第 10 步的逻辑转发，而是直接发送这个请求到这个指定的 proxy 的地址，端口，和协议。如果请求有一个 Route 头域，这个额外的方法就不能用了，除非它知道下一个节点 proxy 是一个松路由的节点。否则，使用上边讲的增加 Route 头域的方法会更有效，更灵活，适应性更好，并且操作更一致。而且，如果 Request-URI 包含了一个 SIPS URI,这个 proxy 必须用 TLS 来进行通讯。

如果请求的拷贝中包含了 Route 头域，这个 proxy 必须检查这个 Route 头域的第一个值。如果这个 URI 并没有包含 lr 参数，那么 proxy 必须根据下列步骤修改这个请求：

－ proxy 必须把 Request-URI 放在 Route 头域中的最后一个值。

－ proxy 必须把第一个 Route 头域的值放在 Request-URI 中，并且从 Route 头域中删去。

把 Request-URI 添加到 Route 头域的最后是为了让 Request-URI 的信息能够通过严格路由的 proxy。”Popping”弹掉第一个 Route 头域值到 Request-URI

中是为了能够让严格路由元素能够接收到这个请求（并且用它自己的在 Request-URI 中的 URI 和在 Route 顶部下一个节点的 URI）。

7、确定下一个节点的地址，端口和通讯协议。
proxy 可以有自己的策略来决定发送请求到特定的 IP 地址，端口和 transport，可以和 Route 的值或者 Request-URI 的值无关。当本 proxy 不能确定对应 ip，端口，transport 的服务器是一个松路由（loose router）的时候，这样的策略就不能使用了。但是，除了 Route 头域应当像上边讲的这样使用，我们并不推荐这样的发送请求的机制。

在没有这样一个替代机制的时候，proxy 应用附件[4]的步骤来决定应当向哪里发送这个请求。如果 proxy 重新规格化请求，并且发送到一个像上边 6 点讲的严格路由的元素，proxy 必须应用这些步骤到请求中的 Request-URI。否则，如果 Route 头域存在，proxy 必须应用这些步骤到 Route 头域的第一个值；如果 Route 不存在，proxy 必须应用这些步骤到 Request-URI。这些步骤最终得到一个序列集合（地址，port，transport）。与使用那个 URI 作为附件[4]处理的输入，如果 Request-URI 指定了一个 SIPS URI，那么 proxy 必须把输入[4]的 URI 当作是 SIPS URI 然后遵循[4]的处理步骤进行处理。

就像在附件[4]中讲述的，proxy 必须尝试序列集合中的第一组元素，并且依次尝试序列集合中的每一组元素，直到成功为止。

对于每一组的尝试，proxy 必须按照这组的通讯要求，对消息进行适当的格式化，并且用一个新的客户端事务（第 8 到第 10 点讲述的），进行请求的发送。

由于每一组的发送都是使用心得客户端事务，这就体现了一个新的分支。因而，第 8 步插入的 Via 头域中的分支参数必须每组发送的都不一样。

如果客户端事务报告发送请求失败，或者它自身的状态机超时，proxy 就应当继续处理序列集合中的下一组元素。当遍历完序列集合之后，请求就不能发送到目的地集合。proxy 不需要在应答上下文中放什么应答，然而在别的方面却需要就像从目的地集合收到一个 408（Request Timeout）终结响应一样的操作。

8、增加一个 Via 头域值

proxy 必须在请求的拷贝中增加一个 Via 头域值，并且在其他 Via 头域值之前增加。这个值的构造可以参见 8.1.1.7。这意味着 proxy 需要计算自己的分支参数，并且应当是全局唯一的分支，并且包含必要的 magic cookie。注意这意味着如果请求循环经过本 proxy 的时候（也就是数次经过同一个 proxy），每次的分支参数都不同。

在 proxy 构造分支参数的值上，有一个附加的约束，用来进行循环的检测。一个要检测循环的 proxy 应当创建一个由两部分组成的分支参数。第一部分必须满足 8.1.1.7 的约束。第二部分是用来做循环检测的，并且是从螺旋中判定是否存在循环（请求数次经过同一个 proxy 是正常的，这是螺旋，但是如果是循环，那就不正常了。假定 proxy 是 X,CàX,XàY,YàZ,ZàX,XàA,Aà 目的地是正常的，但是如果 CàX, XàY, YàZ, ZàX, XàY, 这就是循环了）。

循环检测是通过这样的方法检测的：当请求返回给一个 proxy，与处理请求相关的字段并未改变，那么这个就是循环了。这个分支参数的后一部分应当反应所有的这些头域（包括所有的 Route,Proxy-Require 和 Proxy-Authorization 头域）。这是确保如果请求从别处重新路由回来，而且这些字段改变了，那么这就是一个螺旋而不是循环（参见 16.3）。通常建立这个比较值的方法是计算一个 hash 值，通过基于 To tag,From tag,Call-ID 头域，收到请求的 Request-URI（而不是经过处理过后的 Request-URI）,Via 头域的最上一个，Cseq 头域的序列号，任何附加的 Proxy-Require 或者 Proxy-Authorization 头域。具体的

hash 算法是基于实现相关的，但是 MD5(RFC1321[35])，用 16 进制表达，是一个有道理的选择。（基于 64 位表达的是不太合适的）。

如果 proxy 希望检测循环，那么”branch”参数必须用包含可能影响处理请求的全部信息构成，包括输入的 Request-URI 和其他可能会影响 proxy 处理路由的字段，通过计算得到。这是检测循环所必须的，因为如果请求在路由相关的字段改变以后，重新发回这个服务器，那么新的处理可能会发送到另外的地方，而不是造成一个循环。

在 branch 参数的计算上，请求的方法不能计算进去。但是作为特例，CANCEL 和 ACK 请求（给非 2xx 应答的）必须和他们对应的请求有相同的 branch 值。branch 参数用于在服务器处理这些请求的时候体现请求之间的相关性（17.2.3 和 9.2）

9、如果需要，增加一个 Content-Length 头域

如果请求会通过一个基于流的通讯协议发送到下一个节点，并且发送的请求拷贝中没有包含一个 Content-Length 头域，那么 proxy 必须增加一个正确的请求包体大小在这个头域(20.14)。

10、 转发请求

一个有状态的 proxy 必须为这个请求创建一个新的客户端事务（根据 17.1 节描述的那样），并且指示事务层用第 7 步指定的地址，端口和协议进行发送。

11、 设定时钟 C

为了能够处理 INVITE 请求没有产生终结应答的情况，TU 使用一个定时器（称作定时器 C）。当 INVITE 请求被转发的时候，必须为客户端事务设定一个定时器 C。这个定时器 C 必须大于 3 分钟。16.7 节的 2 步讲述了这个定时器是如何根据临时应答来更新的，并且 16.8 节讲述了定时器到时的处理步骤。

16.7 应答的处理

当 proxy 收到一个应答的时候，它首先尝试定位一个与这个应答匹配的客户端事务（17.1.3）。如果没有匹配，proxy 必须作为无状态的 proxy 来处理这个应答（即使这个应答是信息性质的应答）。如果与应答匹配的客户端事务找到了，那么这个应答将转给这个客户端事务进行处理。

将应答转给对应的客户端事务（或者更通常的说法是发出请求的或者相关的事务），并不是为了更强大的处理能力，它是保证了”晚到”的给 INVITE 请求的 2xx 应答能够正确的转发。

当客户端事务把应答交给 proxy 层，将会执行下列步骤：

1、 寻找适当的应答上下文。

2、 用临时应答来更新定时器 C

3、 从最上边移除 Via

4、 在应答上下文中增加应答

5、 检查这个应答是否需要立刻发送

6、 如果需要，从应答上下文中选择最好的终结应答。

如果在与这个应答上下文相关的每一个客户端事务都结束的以后，还是没有终结应答转发，那么 proxy 必须选择从已经收到的应答中，选择转发”best”应答回去。

下列步骤必须在每一个被转发的应答上执行。就像每一个请求有超过一个应答被转发一样：至少有一个终结应答和 0 个或者多个临时应答。

7、 需要合并认证头域值。

8、 可选的重写 Record-Route 头域值

9、 转发应答

10、 产生合适的 CANCEL 请求。

上述每一步在下边有详细的描述：

1、寻找上下文

proxy 通过 16.6 节定义的方法来在寻找转发原始请求前创建的”应答上下文” 。在这个上下文中进行后续的处理步骤。

2、为临时应答更新定时器 C

对于 INVITE 事务，如果应答是一个返回码是 101 到 199 的临时应答（就是说，除了 100 的临时应答），proxy 必须给这个客户端事务重新设置定时器 C。这个定时器可以设置成为其他的值，和原始值不一样，但是这个数字必须大于 3 分钟。

3、Via

proxy 从应答中移去 Via 头域中最上的值。

如果在这个应答中没有这个 Via 头域值，那么应答的含义就是说这个应答不应当被这个 proxy 转发。本节描述的后续处理步骤也不需要继续处理，而是用 8.1.3 节定义的 UAC 处理规则进行处理（传输层处理已经进行）。

这种情况可能会发生，比如，当某一个元素产生一个第 10 节规定的 CANCEL 请求。

4、增加应答到上下文

收到的终结应答都会保存在应答的上下文中，直到收到一个由服务端事务产生的针对这个上下文的终结应答为止。这个应答是从那个服务端事务中收到最佳终结应答

的一个候选。即使这个应答不会被选中作为最佳应答，这个应答的信息也需要用来构造成为最佳应答。

如果 proxy 决定尝试调用 3xx 应答返回的联系地址，并且把他们添加到目的地集合，它必须在把这个应答添加到应答上下文之前把联系地址从应答中移除。不过，proxy 不应当在源请求的 Request-URI 是一个 SIPS URI 的情况下，尝试调用非 SIPS URI。如果 proxy 尝试每一个 3xx 应答给回的联系地址，proxy 不应当把这个应答添加到应答上下文中。

从应答中删去联系地址的目的是为了防止下一个节点尝试本 proxy 已经尝试的地址。

3xx 应答可能包含 SIP,SIPS 和非 SIP URI。proxy 可以自行决定自己调用那些 SIP 或者 SIPS URI，并且把剩下的放在应答上下文中返回。

如果 proxy 收到一个对于一个 Rquest-URI 并非 SIP URI 的请求的 416（不支持的 URI scheme）应答，但是原始请求的 Request-URI 是 SIP 或者 SIPS（就是说，proxy 在转发请求的时候自己调换了请求的 SIP 或者 SIPS 为一个什么其他的东西），proxy 应当增加一个新的 URI 到目的地集合。这个 URI 应当是刚才尝试的非 SIP URI 的 SIP URI 版本。对于电话 URL 来说，这个就是把电话 URL 的电话号码部分放在 SIP URI 的用户部分，并且设置 SIP URI 的主机部分成为当前请求发送者的区域。19.1.6 节有电话 URL 到 SIP URI 的转换细节。

在 3XX 应答的情况下，如果 proxy 在 416 上会产生”递归”（因为尝试 SIP 或者 SIPS URI 而导致递归），那么应当在应答上下文中增加这个 416 应答。

5、检查转发的应答

当终结应答到达服务端事务的时候，下列应答包必须立刻转发。

— 任何非 100（trying）的临时应答
— 任何 2xx 应答。
如果收到一个 6xx 应答，那么就不立刻进行转发，如果是有状态的 proxy，那么还需要 cancel 所有的依赖于这个事务的客户端（在 10 节中描述的那样），并且不能在上下文中创建新的分支。

这个是和 RFC 2543 的不同之处，2543 要求 proxy 立刻转发 6xx 应答。对于一个 INVITE 事务来说，如果立刻转发 6xx 应答，会使得 2xx 应答到达别的分支。这个结果就是让 UAC 在 2xx 应答之后收到一个 6xx 应答，这个是不允许发生的。在新的规则下，基于接收到一个 6xx 应答，proxy 应当产生一个 CANCEL 请求，那么这个会给所有等待的客户端事务一个 487 应答，这就是 6xx 应答应当给上行队列的一个结果。

在服务端事务上发送了终结应答之后，下列的应答应当立刻被发送：
— 任何给 INVITE 请求的 2xx 应答。

一个有状态的 proxy 必须不能立刻转发其他的应答。特别是，一个有状态的 poxy 必须不能转发任何 100(Trying)应答。这些应答是作为后续将被转发”最佳”应答的候选，通过上边的”在上下文中增加应答”的步骤增加到应答上下文中。

任何将被立刻发送的应答都必须遵照”7、 需要合并认证头域值。”和”8、可选的重写 Record-Route 头域值”来处理。

这一步，合并下一步，确保有状态的 proxy 能够精确转发一个终结应答到一个非—INVITE 请求，或者给一个 INVITE 请求的非 2xx 应答或者一个或者多个 2xx 应答。

6、选择最佳的应答

对于一个有状态的 proxy 来说，如果根据上边的步骤，没有任何终结应答被立刻发送，并且在客户端事务中的所有的客户端服务都已经终结，那么这个 proxy 必须发送一个终结应答到一个应答上下文的服务端事务层。

那么这个有状态的 proxy 就必须从接收到的应答上下文中选择一个”最佳”的终结应答。如果在上下文中没有一个终结应答，那么 proxy 就必须返回一个 408（请求超时）的应答到服务端事务层。

如果应答上下文中有终结应答，那么 proxy 就必须从这个应答上下文中取得应答来发送。如果应答上下文中有 6xx 应答，那么就必须选择这个 6xx 应答。如果没有 6xx 应答，那么 proxy 应当选择最小的应答（应答返回代码比较小）。proxy 可以选择对应最小应答系列中的任意一个应答（比如 2xx 系列中的任意一个应答）。proxy 应当给那些提供对影响请求的应答更多的机会，比如在 4xx 系列中，选择 401，407，415，420 或者 484 应该稍稍优先一些。

当 proxy 收到 503（Service Unavailable）应答的时候，不应当转发到上行队列中，除非它能够知道这个后续的请求队列都能产生 503 的应答。换句话说，就是转发 503 就意味着 proxy 确实不能处理任何请求，不仅仅是 Request-URI 里边的这个地址不能处理请求。如果只有某个应答会产生 503，proxy 应当产生 500 应当到上行队列中。

被转发的应答都必须遵照”7、 需要合并认证头域值。”和”8、可选的重写 Record-Route 头域值”来处理。

例如，如果一个 proxy 转发一个请求到 4 个地方，并且收到了 503，407，501，和 404 应答，它可能选择 407（Proxy Authentication Required）应答。

1xx 和 2xx 应答可能和建立对话有关。当请求没有包含一个 To tag，UAC 使用在应答中的 To tag 来区分请求创建的对话的多个应答。如果请求中没有包含 To 的 tag，那么 proxy 必须不能为 1xx 或者 2xx 应答增加这个 tag 到 To 头域。一个 proxy 不能修改 1xx 或者 2xx 应答中的 To 头域的 tag 字段。

在请求的 1xx 应答中，如果应答没有 To 头域的 tag 字段的时候，由于 proxy 不能添加 tag 字段到这个 To 头域，它就不能产生它自己的非 100 临时应答。但是它可以把这个请求分支到其他一个 UAS 上，这个 UAS 可以和 proxy 共享同样的元素。这个 UAS 可以返回它自己的临时应答，进入请求创建早期对话中。这个 UAS 并没有必要作为一个 proxy 的严格处理步骤存在。它可以是一个在 proxy 内部的一个虚拟的 UAS 实现。

3 到 6xx 的应答是节点到节点传递的。当产生了一个 3—6 系列的应答的时候，每一个节点都作为 UAS 一样，产生它自己的应答，通常基于下行队列的应答产生自己的应答。对于每一个节点来说，在转发 3 到 6 系列应答回去的时候，如果这个应答没有包含 To tag，那么这个节点也应当不改变这个 to tag。

当收到的应当包含了一个 To tag,那么这个 proxy 不能够修改这个 To tag。

恩，实际上在 proxy 转发 3 到 6 系列应答的时候，如果替换了 To tag 也不会让上行队列所经过的节点有影响，保留原始的 tag 值可以有助于调试。

当 proxy 需要合并多个应答的信息的时候，从这些应答中选取 To tag 的方式是任意的，并且产生一个新的 To tag 可能可以使得调试更加容易。举例来说，当合并 401（Unauthorized）和 407（Proxy Authentication Required）信息的时候，或者合并一个未加密的 Contact 值和未通过验证的 3xx 应答的时候，产生一个新的 To tag 就会让调试比较容易。

7、合并认证头域值

如果选择的应答是 401（Unauthorized）或者 407（Proxy Authentication Required），那么 proxy 就必须从本应答上下文中的所有其他 401（Unauthorized）和 407 应答中搜集 WWWAuthenticate 和 Proxy-Authenticate 头域值。并且把这些信息增加到这个应答中。最后的 401 或者 407 应答中可能会包含多个 WWWAuthenticate 和 Proxy-Authenticate 头域值。

由于这个请求的一个或者多个目的地可能是需要请求身份验证的，所以这个搜集步骤就是必须的。客户端需要接收到这些所有目的者的应答并且在下一次尝试的时候，为每一个目的地提供相关的身份证明。在 26 节有相关的说明。

8、Record-Route

如果最终发送的应答中包含 Record-Route 头域值，并且是这个 proxy 所原创提供的值，那么在发送这个应答前，proxy 可能需要重写这个值。这提供了一个机制，让 proxy 能够给下一个上行节点或者下行节点提供非本机的一个 URI 地址。这种情况是很常见的，比如，在多地址主机系统就非常有用。

如果 proxy 是通过 TLS 收到请求的，并且通过非 TLS 转发出去，proxy 必须重写在 Record-Route 头域中的 URI，重写成 SIPS URI。如果 proxy 通过非 TLS 接收到请求，转发是通过 TLS 转发的，那么 proxy 必须重写 Record-Route 请求头域的 URI 为一个 SIP URI。

proxy 提供的新的 URI 必须满足同样的 Record-Route 头域的 URI 约束（16.6 节的第四步）。并且遵循下列的修改：

URI 不应当包含通讯参数除非 proxy 知道下一个上行（同下行队列对应的）节点，对于后续的请求都支持相关的通讯参数。

如果 proxy 打算修改应答中的 Record-Route 头域，要做的一件事情就是定位插入的 Record-Route 头域值。如果请求是螺旋经过的，并且 proxy 在每次螺旋的时候都插入了 Record-Route 值，在应答中（必须在反向路径中的正确位置）找到正确的需要修改的值就需要一点技巧。上边的规则强调 proxy 在增加 Record-Route 头域值的时候是必须增加唯一的 URI，这样才能找到正确的一个能够重写。我们推荐 proxy 为每一个 URI 的 user portion 增加一个唯一个 proxy 实例标志。

当应答到达的时候，proxy 修改第一个和 proxy 实例标志匹配的 Record-Route。这个修改导致产生一个在 user portion 部分去掉 proxy 实例的 URI。到下一个循环回来处理的时候，同样的算法（用参数从上而下的寻找 Record-Route 头域值）会更改这个 proxy 插入的下一个 Record-Route 头域值。

对于 proxy 增加 Record-Route 头域值的请求来说，并非每一个应答都包含一个 Record-Route 头域。如果应答包含一个 Record-Route 头域，那么就包含这个 proxy 增加的值。

9、转发应答

当”合并认证头域”和”Record-Route”步骤完成以后，proxy 可以对这个应答做其他的附加处理。但是这个 proxy 不能增加、修改、删除消息体。并且除非另有指示，除了 Via 头域值（在 16.7 节 3 步）之外，proxy 不能删除任何头域值。特别是，proxy 不能删除任何可能增加到与处理和这个应答相关的下一个请求的 Via 头域值的”接收到”的参数。proxy 必须把应答传递到跟这个应答上下文相关的服务端事务。这回导致应答发送到最上的 Via 头域值的地方。如果服务端事务不在处理这个发送，这个节点必须作为无状态 proxy 转发这个应答到服务端通讯层。服务端事务可能已经标志这个发送应答失败或者内部状态机已经设置成为超时状

态。这些错误都应当记录下来用于诊断错误，但是协议并没有要求 proxy 做补救措施。

proxy 必须维持应答上下文直到所有相关事务都已经终结，甚至在发送完成终结应答后还需要维持。

10、 产生 CANCEL 请求

如果转发的应答是一个终结应答，proxy 必须给依赖于这个应答上下文的所有客户端事务，产生 CANCEL 请求。在收到 6xx 应答的时候，proxy 同样应当为所有等待在这个应答上下文的客户端事务产生 CANCEL 请求。等待的客户端事务就是收到了临时应答，但是没有收到终结应答（还是出于处理中的状态），并且没有任何 CANCEL 请求与之相关的请求。产生 CANCEL 请求请参见 9.1 节。

对于要求基于转发终结应答而 CANCEL 的客户端事务并没有保证终端不会收到给一个 INVITE 的多个 200（OK）应答。基于多余一个分支的 200（OK）应答可能会在 CANCEL 请求处理前到达。进一步说，后续的扩展可能会改掉这个产生 CANCEL 请求的要求。

16.8 处理定时器 C

如果定时器 C 被出发了，proxy 必须要么用另外一个数值重新设定定时器，要么终结客户端事务。如果客户端事务已经收到了临时应答，那么 proxy 必须产生一个与之匹配的 CANCEL 请求。如果客户端事务还没有收到临时应答，那么 proxy 必须就像收到一个 408（Request Timeout）一样的处理。

允许 proxy 重设定定时器就意味着允许 proxy 基于当前条件（比如服务器利用率等等）动态的扩展事务的生命周期。

16.9 处理通讯层的错误

如果在转发请求（参见 18.4）的时候，通讯层报告了一个错误，那么 proxy 必须就像收到了一个 503（Service Unavailable）应答一样的处理。

如果 proxy 在转发应答的时候接收到错误，那么他就丢弃应答。proxy 不能由于通讯的原因而 cancel 任何和这个应答上下文相关的客户端事务。

如果 proxycancel 了这些客户端事务，那么一个恶意的或者出错的客户端可以用一个 Via 头域导致所有的事务都失败。

16.10 CANCEL 处理

一个有状态的 proxy 可以给他自己产生的其他请求在任何时候都产生 CANCEL 请求（参见 9.1 遵从接收到对应请求的一个临时应答）。在接收到一个匹配的 CANCEL 请求的时候，proxy 必须取消任何与应答上下文相关的客户端事务。

当 INVITE 请求有一个 Expires 头域并且这个头域值已经超时的情况下，一个有状态的 proxy 可以对这个处于 pending 的 INVITE 客户端事务发出 CANCEL 请求。可是，通常来说，这是不必要的，因为相关的终端会发出结束事务的信号。

当有状态的 proxy 在它自己的服务端事务上处理 CANCEL 请求的时候，并没有新的应答上下文会创建。相反，proxy 层寻找与这个 CANCEL 对应请求的现存的应答上下文。如果找到了对应的应答上下文，那么这个节点应当立刻返回一个 200（OK）应答给这个 CANCEL 请求者。在这个情况下，这个节点就像 8.2 节定义的 UAS 一样的工作。进一步说，这个节点应当为每一个依赖于这个上下文的客户端事务产生一个 CANCEL 请求（就像在 16.7 节第 10 步描述的那样）。

如果一个应答上下文没有找到，这个节点就无法 CANCEL 这个请求。它就必须像无状态 proxy 一样转发这个 CANCEL 请求（可能这个节点把被 CANCEL 的请求在先前也当作无状态的 proxy 转发了）。

16.11 无状态的 proxy

当作为无状态的时候，proxy 就是一个简单的消息转发者。很多无状态的处理步骤和有状态的时候很类似。不同的地方在下边描述。

一个无状态的 proxy 并没有事务的概念，或者用于描述有状态 proxy 行为的应答上下文。相反的是，无状态的 proxy 处理消息，无论是请求还是应答，都是直接从通讯层处理的（参见 18 节）。当然，无状态 proxy 自己也不重发这些消息。他们只是转发他们收到的任何重发的消息（他们本身并没有能力来分辩那些消息是重发的，那些消息是原始消息）。进一步说，当无状态的处理一个请求的时候，这个节点并不产生它自己的 100（Trying）或者其他临时应答。

无状态的 proxy 必须用 16.3 节描述的那样来验证一个请求。

无状态的 proxy 必须遵从 16.4 到 16.5 节定义的步骤来处理请求，有如下几点例外：

o 无状态的 proxy 必须从目的地集合中，选择一个并且只能选择一个目的地。这个选择必须是根据消息的头域并且是和服务器时间无关的。特别是，一个重发的请求必须能够每次都转发到相同的目的地。进一步说，CANCEL 和非路由的 ACK 请求必须和他们相关的 INVITE 请求有相同的转发目的地。

一个无状态的 proxy 必须遵循 16.6 节定义的请求处理步骤，并且有下列的不同：

o 无状态 proxy 的 branchID 来说，必须要求在时间上和空间上都是唯一的。也就是说，无状态的 proxy 不能简单的使用一个随机数产生器来计算 branchID 的第一个部分（16.6 节 8 步）。这是由于请求的重发需要相同的值，并且无状态的 proxy 不能区分重发的请求和原始请求。因此，branch 参数的组成部分要求唯一，这样使得重发的时候能够填写相同的值。对于无状态的 proxy 来说，branch 参数必须作为一个重发无关的消息处理参数存在。

我们没有规定无状态 proxy 采用何种手段保证 branchID 的唯一性。不过，下列步骤是推荐的方法。proxy 检查在接收到请求的最上 Via 头域值的 branchID。如果它是由 magic cookie 打头的，那么 branchID 的第一个部分就是当作接收到的 branchID 的 hash 值。否则 branchID 的第一个部分就当作是 Via 头域的最上值、To 头域的 tag、From 头域的 tag，Call-ID 头域，Cseq 序列号（除了方法部分），接收到的请求的 Request-URI 的一个 hash 值。这些头域值在不同事务中总是不一样的。

o 所有其他的消息转换（16.6 节）必须保证转发重发的请求的时候能够转发到相同的节点。特别是，如果 proxy 在 Record-Route 头域中增加了值，或者在 Route 头域中增加了值，proxy 必须在转发重发的请求的时候增加相同的值。至于 Via 的 branch 参数，这就意味着转发必须是基于时间无关的配置或者请求重发无关的属性。

o 一个无状态 proxy 决定转发的地点是像 16.6 节 10 步描述的有状态的 proxy 一样。但是请求是直接交给通讯层发送的，而不是交给客户端事务。

由于一个无状态的 proxy 必须转发重发的请求到相同的地方，并且增加标志性的 branch 参数，它只能用消息中本身的信息和时间无关的配置来计算。如果配置状态不是时间无关的（比如，如果路由表更新了），这个改变相关的请求，在这个改动开始以后，到在事务超时的时间范围内，就不能作为无状态的转发了。这个处理

这段时间的请求是实现相关的。通常处理的方法，是把这些请求当作事务有状态的进行转发。

无状态的 proxy 必须不能对 CANCEL 做特别的处理。CANCEL 的处理就像对其他请求的处理一样进行。特别是，一个无状态的 proxy 使用相同的 Route 头域来处理 CANCEL 请求，就像处理其他请求一样。

对于 16.7 节中定义的应答处理，对于无状态 proxy 来说，并不适用。当一个应答到达一个无状态 proxy，proxy 必须检查最上的 Via 头域值的 sent-by 参数。如果这个地址和这个 proxy 一样（就是和 proxy 插入的先前的请求中的值一样），那么这个 proxy 必须从应答中移除这个头域值，并且转发这个应答到下一个 Via 头域值。这个 proxy 必须不能增加，修改或者删除消息体。除非有特别的说明，proxy 必须不能移除其他的头域值。如果地址不匹配本 proxy，消息就必须简单的悄悄扔掉。

16.12 Proxy Route 处理的总结

在没有本地策略的情况下，proxy 对于包含 Route 头域的请求处理可以归结于如下的步骤：

1、 proxy 会检查 Request-URI。如果它指向的是本 proxy 所负责的区域，那么 proxy 会用位置服务的结果来替换这个 URI。否则，proxy 不改变这个 URI。
2、 proxy 会检查 Route 头域的最上 URI。如果这个 URI 指向这个 proxy，这个 proxy 从 Route 头域中移除（这个路由节点已经到达）。
3、 proxy 会转发请求到最上的 Route 头域值所标志的 URI，或者 Request-URI(如果没有 Route 头域)。proxy 通过附件[4]的步骤来产生地址，端口，通讯协议等等用来转发请求所必须的参数。

如果在请求的路径中，没有严格路由节点，Request-URI 会始终标志着请求的目的地。

16.12.1 例子

16.12.1.1 基本 SIP 四边形

本例子是一个基本的 SIP 四边传送，U1->P1->P2->U2，使用 proxy 来传送。下边是过程。

U1 发送：

INVITE sip:callee@domain.com SIP/2.0

Contact: sip:caller@u1.example.com

发给 P1,P1 是一个外发的 proxy。P1 并不管辖 domain.com，所以它查找 DNS 并且发送请求到那里。它也增加一个 Record-Route 头域值：

INVITE sip:callee@domain.com SIP/2.0

Contact: sip:caller@u1.example.com

Record-Route: <sip:p1.example.com; lr>

P2 收到这个请求。这是 domain.com 所以它查找位置服务器并且重写 Request-URI。它也增加一个 Record-Route 头域值。请求中没有 Route 头域，所以它解析一个新的 Request-URI 来决定把请求发送到哪里。

INVITE sip:callee@u2.domain.com SIP/2.0

Contact: sip:caller@u1.example.com

Record-Route: <sip:p2.domain.com; lr>

Record-Route: <sip:p1.example.com; lr>

在 u2.domain.com 的被叫方接收到这个请求并且返回一个 200OK 应答：

SIP/2.0 200 OK

Contact: sip: callee@u2.domain.com

Record-Route: <sip:p2.domain.com;lr>
Record-Route: <sip:p1.example.com;lr>

u2 的被叫方并且设置对话的状态的 remote target URI 为：
sip: caller@u1.example.com并且它的路由集合是：
（<sip:p2.domain.com;lr>,<sip:p1.example.com;lr>）

这个转发通过 P2 到 P1 到 U1。现在 U1 设置它自己的对话状态的 remote target URI 为：sip:calle@u2.domain.com 并且它的路由集合是：
（<sip:p1.example.com;lr>,<sip:p2.domain.com;lr>）

由于所有的路由集合元素都包含了 lr 参数，那么 U1 构造最后的 BYE 请求：
BYE sip:callee@u2.domain.com SIP/2.0
Route: <sip:p1.example.com;lr>,<sip:p2.domain.com;lr>

就像其他所有的节点（包括 proxy）会做的那样，它会使用 DNS 来解析最上的 Route 头域的 URI 值，这样来决定往哪里发送这个请求。这就发到了 P1。P1 发现 Request-URI 中标记的 URI 不是它负责的域，于是它就不改变这个 Request-URI。然后看到它是 Route 头域的第一个值，于是就从 Route 头域中移去，并且转发这个请求到 P2：
BYE sip:callee@u2.domain.com SIP/2.0
Route: <sip:p2.domain.com;lr>
P2 也发现它自己并非负责这个 Request-URI 的域（P2 负责的是 domain.com 并非 u2.domain.com）,于是 P2 并不改变它。它看到自己在 Route 的第一个值，于是移去这个，并且向 u2.domain.com 转发（根据在 Request-URI 上查找 DNS）：
BYE sip:callee@u2.domain.com SIP/2.0

16.12.1.2 穿越一个严格路由 proxy

在这个例子中，对话建立通过 4 个 proxy，每一个增加 Record-Route 头域值。第三个 proxy 是由严格路由实现的（RFC 2543）。

U1->P1->P2->P3->P4->U2

INVITE 请求到达 U2 包括了：

INVITE sip:callee@u2.domain.com SIP/2.0
Contact: sip:caller@u1.example.com
Record-Route: <sip:p4.domain.com;lr>
Record-Route: <sip:p3.middle.com>
Record-Route: <sip:p2.example.com;lr>
Record-Route: <sip:p1.example.com;lr>

并且 U2 返回了一个 200 OK。接着，U2 根据第一个 Route 头域值发送下边的 BYE 请求到 P4：

BYE sip:caller@u1.example.com SIP/2.0
Route: <sip:p4.domain.com;lr>
Route: <sip:p3.middle.com>
Route: <sip:p2.example.com;lr>
Route: <sip:p1.example.com;lr>

P4 并不管辖 Request-URI 指出的域，于是就不更改这个 Reqeust-URI。它发现自己在第一个 Route 头域中，于是把自己从 Route 头域移除。然后准备发送请求到现在的第一个 Route 头域值：sip:p3.middle.com,但是它发现这个 URI 并没有包含 lr 参数，于是在发送前，它把这个请求更改成为：

BYE sip:p3.middle.com SIP/2.0
Route: <sip:p2.example.com;lr>
Route: <sip:p1.example.com;lr>
Route: <sip:caller@u1.example.com>

P3 是一个严格路由，于是它转发到 P2：

BYE sip:p2.example.com;lr SIP/2.0

Route: <sip:p1.example.com;lr>

Route: <sip:caller@u1.example.com>

P2 看到 Request-URI 是它放在 Record-Route 头域中的值，于是在进一步处理前，它把这个请求改写为：

BYE sip:caller@u1.example.com SIP/2.0

Route: <sip:p1.example.com; lr>

P2 自己并不管辖 u1.example.com，于是它根据 Route 头域的值，转发这个请求到 P1。

P1 发现自己在 Route 头域的最上，于是把自己移除，得到：

BYE sip:caller@u1.example.com SIP/2.0

由于 P1 并不管辖 u1.example.com 并且没有其他的 Route 头域，P1 会基于 Request-URI 转发这个请求到 u1.example.com。

16.12.1.3 重写 Record-Route 头域值。

在这里例子中,U1 和 U2 是在不同的私有域空间中，并且他们通过 proxy P1 开始一个对话，这个 P1 作为不同私有 namespace 的一个网关存在。

U1->P1->U2

U1 发送：

INVITE sip:callee@gateway.leftprivatespace.com SIP/2.0

Contact: <sip:caller@u1.leftprivatespace.com>

P1 使用自己的定位服务并且发送下边的信息到 U2:

INVITE sip:callee@rightprivatespace.com SIP/2.0

Contact: <sip:caller@u1.leftprivatespace.com>

Record-Route: <sip:gateway.rightprivatespace.com;lr>

U2 发送 200 OK 应答回给 P1：

SIP/2.0 200 OK

Contact: <sip:callee@u2.rightprivatespace.com>

Record-Route: <sip:gateway.rightprivatespace.com;lr>

P1 重写它的 Record-Route 头域参数，提供成为 U1 能够使用的参数，并且发送给 P1：

SIP/2.0 200 OK

Contact: <sip:callee@u2.rightprivatespace.com>

Record-Route: <sip:gateway.leftprivatespace.com;lr>

稍后，U1 发送接下来的 BYE 到 P1:

BYE sip:callee@u2.rightprivatespace.com SIP/2.0

Route: <sip:gateway.leftprivatespace.com;lr>

P1 转发到 U2：

BYE sip:callee@u2.rightpriatespace.com SIP/2.0

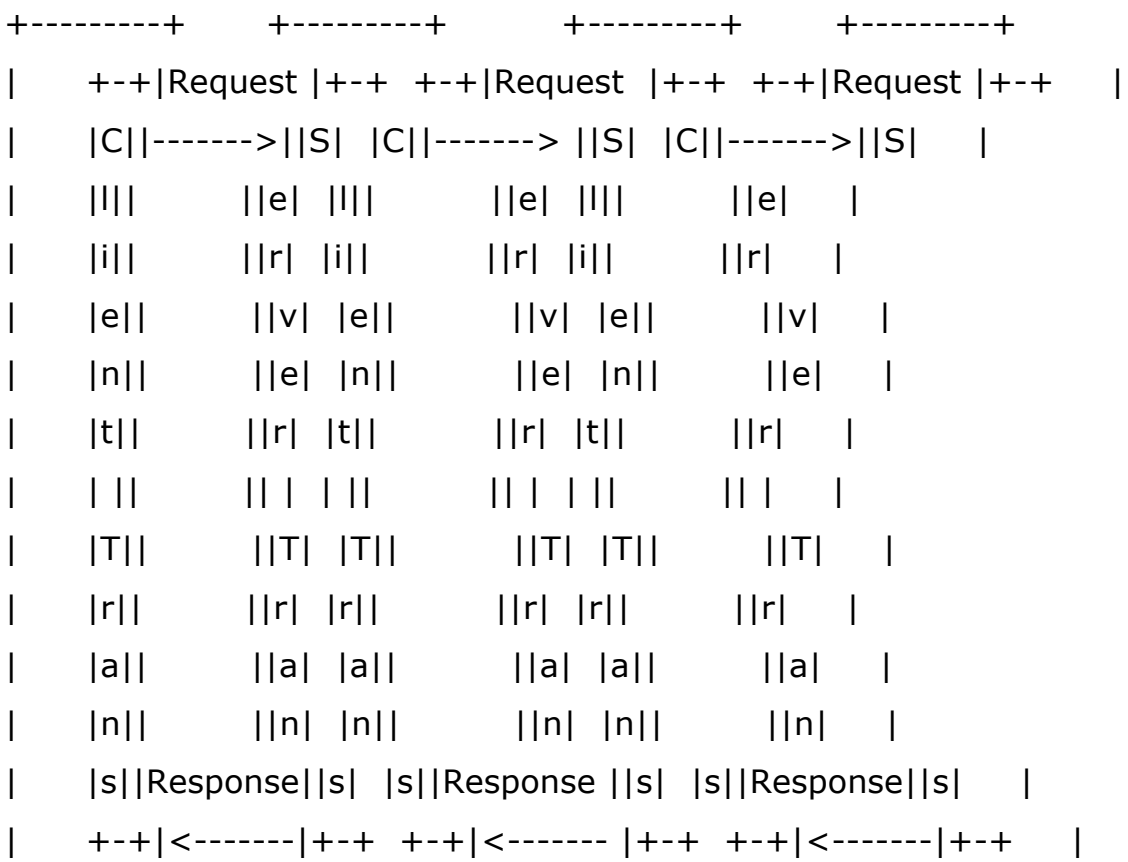
17 事务

SIP 是一个基于事务处理的协议：部件之间的交互是通过一系列无关的消息交换所完成的。特别是，一个 SIP 事务由一个单个请求和这个请求的所有应答组成，这些应答包括了零个或者多个临时应答以及一个或者多个终结应答。在事务中，当请求是一个 INVITE（叫做 INVITE 事务），当终结应答不是一个 2xx 应答的时候，事务还包括一个 ACK。如果应答是一个 2xx 应答，那么 ACK 并不认为是事务的一部分。

这个分开的原因是基于传递全部 200(OK)应答到 UAC 的 INVITE 请求的重要性所决定的。要把所有的 200 应答全部发给 UAC，那么 UAS 独自负责这些应答的

重新传送（参见 13.3.1.4），UAC 独子负责挨个 ACK 确认（参见 13.2.2.4）。由于 ACK 的重传只由 UAC 发起，所以在自己的事务中进行重传会比较有效。

事务分为客户端和服务端两方。客户端的事务是客户端事务，服务器端的事务就是服务端事务。客户端事务发出请求，并且服务端事务送回应答。客户端和服务端事务都是逻辑上的概念，他们可以被无数部件所包含。特别是，他们在 UA 中和有状态的 proxy 服务器中存在。以第四节的例子来说明。在这个例子中，UAC 执行客户端事务，它的外发 proxy 执行服务端事务。外发 proxy 同时也执行客户端事务，把请求发送到一个那发 proxy 的服务端事务。这个 proxy 也同时执行一个客户端事务，把请求发到一个 UAS 的服务端事务上去。这个在图四中比较明白：



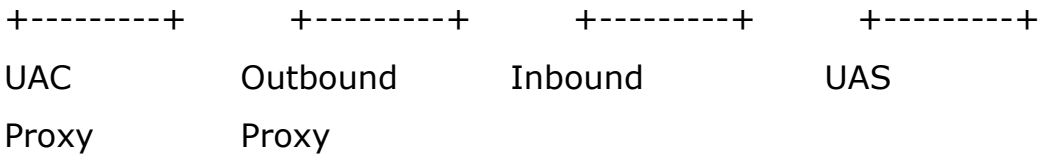


图 4： 事务关系

无状态的 proxy 并没有客户端或者服务端的事务。事务是一边基于 UA 或者有状态的 proxy，另外一边也基于 UA 或者有状态的 proxy。在 SIP 事务范畴下，无状态的 proxy 是用作透明转发很有效。客户端事务的用处是用于从一个元素中接收一个请求，这个客户端是内嵌的（这个元素就是”事务用户”或者 TU；它可以是一个 UA 或者有状态的 proxy），并且可靠的把这个请求传送到一个服务端事务。

客户端事务也负责接收应答并且把应答转交 TU 处理，过滤掉重发的应答或者不允许的应答（比如给 ACK 的应答）。另外，在 INVITE 请求的情况下，客户端事务也负责产生给 2xx 应答的 ACK 请求。

类似的，服务端事务也负责从通讯层接收请求并且转发这个请求到 TU。服务端事务过滤重发的请求。并且服务端事务从 TU 接收应答并且转发到通讯层来发送。在 INVITE 事务的情况下，它需要接收给非 2xx 应答的终结应答的 ACK 请求。

2xx 应答和它的 ACK 请求通过特定的方式来接收和处理。这个应答只会被 UAS 重发，并且它的 ACK 只由 UAC 产生。由于呼叫者知道整个已经接收呼叫的用户集合，所以需要这种端到端的处理。由于这样的特别处理，2xx 应答的重发是基于 UA 核心的，并非基于通讯层。类似的，给 2xx 应答的 ACK 处理也是由 UA 核心处理的，每个路径上的 proxy 仅仅转发这些 INVITE 的 2xx 应答以及他们的 ACK。

17.1 客户端事务

客户端事务是通过维持一个状态机来提供服务的。

TU 和客户端事务通过一个简单的接口进行通讯。当 TU 希望初始化一个新的事务，它创建一个客户端事务并且通过设置 ip 地址，端口和 transport 来把一个 SIP 请求交给它传送。然后客户端事务开始执行它自己的状态机。合乎规格的应答会从客户端事务传送给 TU。

总共有两种类型的客户端事务状态机，根据 TU 传递的请求的方法不同来区分的。一个用于处理 INVITE 请求。这种状态机对应的是一个 INVITE 客户事务。另外一个是用来处理其他所有的非 INVITE 请求的。它对应的是非 INVITE 客户事务。对于 ACK 来说，是不存在客户事务的。如果 TU 希望送一个 ACK 请求，它直接交给通讯层进行通讯处理。

INVITE 事务和其他事务是不同的，因为它的时间周期很长。通常，对于 INVITE 请求的应答来说，都需要人的参与，这样会导致在应答 INVITE 请求之前会有很长的延时。在三方握手（人，两方机器）的时候也会有很长的延时。在另一方面，其他请求的响应都是很快就完成的。因为其他非 INVITE 请求事务是双方的握手，TU 能够立刻对非 INVITE 请求作出应答。

17.1.1 INVITE 客户事务

17.1.1.1 INVITE 事务概述

INVITE 请求包含了一个三方的握手。客户端事务发送一个 INVITE，服务端事务回送一个应答，客户端事务发送一个 ACK。对于非可靠传输（比如 UDP），客户端事务每隔 T1 重发请求，每次重发后间隔时间加倍。T1 是一个估计的循环时间（round-trip time，RTT），缺省设置成为 500ms。几乎所有的事务定时器都以 T1 为单位，并且调整 T1 的值也就调整了那些定时器的值。请求不会在可靠的通讯协议上重新发送。在接收到 1xx 应答以后，重发机制完全停止，并且客户端等待更进一步的应答。服务端事务可以发送附加的 1xx 应答，这个应答并非由服

务端事务可靠传输。最后，服务端事务会发送一个终结应答。对于非可靠的传输协议，应答会间隔时间来重发，对于可靠的传输协议，它只发送 1 次。对于客户端事务所接收的每一个终结应答，客户端事务都发送一个 ACK，用于终止应答的重发送。

17.1.1.2 正式的描述

INVITE 客户端事务的状态机在图 5 中展示。初始状态，”calling”，必须保证 TU 是用 INVITE 请求来初始化一个新的客户端事务。客户端事务必须把请求发送到通讯层来进行发送（18 节）。如果使用的是非可靠传输的通讯层，客户端事务必须启动一个定时器 A 并且由缺省值 T1 组成。如果是一个可靠的通讯协议，那么客户端事务不应当启动定时器 A（定时器 A 控制请求的重发送）。对于任何通讯协议来说，客户端事务必须启动一个定时器 B 并且有着 64×T1 秒的缺省值（定时器 B 控制事务的超时）。

当定时器 A 触发了，客户端事务必须重发这个请求，把请求交给通讯层进行发送，并且重新设置定时器为 2*T1。在传输层中重传的定义是指把刚才通过传输层发送的消息，再次交给传输层重新发送一次。

当定时器 A 在 2×T1 后触发了，请求必须再次重传（如果客户端事务依旧还是在这个状态的话）。这个处理必须持续下去，这样请求才能每重发一次以后定时器延时 1 倍。重发机制只有当客户端事务在”calling”状态的时候才能进行。

缺省的 T1 是 500ms。T1 是一个 RTT 的估计时间，是在客户端和服务端的一个事务处理的估计时间。节点可以（不推荐）使用更小的 T1 值，比如私有网络，并不接到 INTERNET 的网络可以设置小一点。T1 也可以设置成为大一点的值，并且我们建议如果当我们知道 RTT 值比较大的时候（比如高延时的网络）应当设置

T1 成为大一点的值。不管 T1 如何取值，本节要求的重传机制要求的指数延时是必须使用的。

当定时器 B 触发的时候，如果客户端事务是依旧在”calling”状态，那么客户端事务应当通知 TU 发生了超时。客户端事务必须不能产生 ACK。64×T1 是和在不可靠通讯链路上传输 7 个请求的时间相同。

如果客户端事务在”calling”状态接收到一个临时应答，那么就把状态切换到”proceeding”状态，客户端事务不应当再次重新发送请求了。进一步说，临时应答必须传送给 TU。在”proceeding”状态的任何临时应答都必须传送给 TU。

当在”calling”或者”proceeding”状态的时候，如果接收到一个应答码是 300-699 的应答，那么就必须把状态切换到”Completed”。客户端事务必须把收到的应答转给 TU，并且客户端事务必须传生 ACK 请求，即使通讯层是可靠传输的（在 17.1.1.3 节中有描述怎样根据应答创建一个 ACK 请求）并且把 ACK 交给传输层进行传送。ACK 必须和原始请求发送到相同的地址，端口和用同样的 transport。当客户端事务进入”Completed”状态的时候，应当开始一个定时器 D，缺省值是在非可靠通讯上是至少 32 秒，在可靠通讯上是 0 秒。定时器 D 反应了服务端事务在非可靠传输的情况下，在”completed”状态维持的时间。这个是和 INVITE 请求服务端事务定时器 H 相同的，定时器 H 的缺省值是 64*T1。不过，客户端事务不知道服务端事务使用的 T1 值，所以我们用绝对值 32 秒来代替 T1 用作定时器 D 的缺省值。

在”completed”状态下，受到的任何终结应答的重传都应当产生一个 ACK 应答到通讯层来重新发送，但是新近收到的应答却不能传送给 TU。一个应答是否是重传的定义是根据这个应答是否和客户端事务按照 17.1.3 定义的规则匹配。

Terminated

TIMER D fires

2xx
2xx to TU
300-600
ACK sent
resp. to TU
300-600
ACK sent
transport error
report to TU
Completed
Proceding
1XX
1XX to TU
2xx
2xx to TU
Timer B fires
or Transport err
inform TU
INVITE from TU
INVITE sent

Timer A fires
Reset A
INVITE sent
1XX
1XX to TU
300-699
ACK sent

resp. to TU
NOTE
transitions
labeled with
the event
over the action
to take

图 5： INVITE 客户端事务

如果在客户端事务状态是”Completed”的时候，定时器 D 触发，那么客户端事务必须转到终结状态。当客户端状态是”calling”或者”proceeding”状态的时候，接收到一个 2xx 应答必须导致客户端事务进入”terminated”状态，并且应答必须交给 TU 处理。处理这个应答的方法依赖于 TU 是否是一个 proxy 核心还是是 UAC 核心。UAC 核心会给应答产生 ACK，proxy 核心会转发一个 200(OK)应答到上行队列。这个在 proxy 和 UAC 之间，对 200(OK)的不同的处理是导致对应答的处理不在事务层进行的原因。

当客户端事务进入”terminate”状态以后，客户端事务必须立刻销毁。这样才能保证正确操作。原因是当给一个 INVITE 请求的 2xx 应答的不同处理；对于 proxy 转发的时候和对 UAC 处理 ACK 的时候是不一样的。因此，每一个 2xx 都需要交给 proxy 核心（这样才能被转发），或者交给 UAC 核心（这样才能被 ACK 确认）。这期间没有事务层的处理。无论应答是否由通讯层收到，如果通讯层找不到匹配的客户端事务（用 17.1.3 的方式），那么应答就应当交给核心处理。这是由于与之匹配的客户端事务已经被第一个 2xx 应答所销毁，后续的 2xx 应当就匹配不成功了，于是就交给核心来处理。

17.1.1.3 构造 ACK 请求

本节定义了在客户端事务中构造 ACK 请求的方法。UAC 核心为 2xx 应答产生 ACK 请求必须使用 13 节描述的方法，而不是用下边的方法。

在客户端事务中构造的 ACK 请求必须包括与原始请求相同的 Call-ID, From, Request-URI 头域值（就是说和在客户端事务发到通讯层的请求中的这些头域值相同）。在 ACK 请求中的 To 头域必须和被确认的应答的 To 头域值相同，因此通常和原始请求有所不同，不同点在增加了附加的 tag 参数。ACK 必须包含一个单个的 Via 头域，并且必须和原始请求的最上边一个 Via 头域值相等。ACK 的 Cseq 头域必须包含和原始请求的 Cseq 的序列号相同，但是方法参数应当是”ACK”。

如果 INVITE 请求的应答是有 Route 头域的，这些 Route 头域必须也在 ACK 中。这是确保 ACK 能够正确路由通过下行队列的无状态的 proxy。

虽然请求可以包含一个包体，但是 ACK 的包体却比较特别，因为请求不能因为不能理解包体而拒绝这个请求。因此，我们不建议在给非 2xx 应答的 ACK 请求中放置包体，但是如果放置了，并且假设给 INVITE 的应答不是 415 应答，那么包体的类型应当严格和 INVITE 请求中定义的那样。如果是 415 应答，那么 ACK 的包体应当和 415 应答中的 Accept 列出的类型一致。

例如：有如下请求

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKkjshdyff
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=88sja8x
Max-Forwards: 70
Call-ID: 987asjd97y7atg
```

Cseq: 986759 INVITE

给非 2xx 终结应答的 ACK 请求应当是：

ACK sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKkjshdyff
To: Bob <sip:bob@biloxi.com>;tag=99sa0xk
From: Alice <sip:alice@atlanta.com>;tag=88sja8x
Max-Forwards: 70
Call-ID: 987asjd97y7atg
Cseq: 986759 ACK

17.1.2 非 INVITE 客户端事务

17.1.2.1 非 INVITE 事务概览

非 INVITE 事务并不使用 ACK。他们只是简单的请求－应答的交互。对于非可靠的通讯来说，请求是间隔倍增 T1 的时间重新传输（直到间隔时间达到 T2）。如果收到了一个临时应答，在非可靠通讯上，重传继续知道达到 T2。只有当重传的请求收到的时候，服务端事务会重传其发出的最后一个应答，既可以是临时的应答也可以是终结应答。这就是为什么请求在收到一个临时应答之后还需要一直重传的原因；他们能够确保收到一个终结应答。

不像 INVITE 事务，非 INVITE 事务不需要对 2xx 应答做特别处理。UAC 对一个非 INVITE 请求来说，只会产生一个单个的 2xx 应答。

17.1.2.2 正式的描述

在图 6 中讲述了非 INVITE 客户端事务的状态机。这个状态机和 INVITE 客户端事务的状态机非常像。

当 TU 用请求来初始化一个新的客户端事务的时候，首先进入的是"trying"状态。当进入这个状态的时候，客户端事务应当初始化一个定时器 F，这个定时器 F 应当有一个初始值 64×T1 秒。这个请求必须交给通讯层来发送。如果使用的是非可靠传输的通讯协议，客户端事务必须还设置定时器 E，初始值是 T1。如果定时器 E 触发了，并且还是在"trying"状态，那么定时器需要设置成为 MIN(2*T1,T2)，并且重新发送；如果再次触发了，那么就再设置成为 MIN(4*T1,T2)，每次都是倍增，知道 T2。这个过程会一直继续，直到重发的间距是 T2 为止。缺省的 T2 是 4 秒，并且它大概是一个在没有立刻响应的情况下，非 INVITE 服务端事务处理一个请求的时间。根据缺省的 T1 和 T2,那么间隔就会是：
500ms,1s,2s,4s,4s,4s 以次类推。

如果定时器 F 触发了，并且客户端事务依旧是在"trying"状态，那么客户端事务应当通知 TU 这个超时，并且转入"terminate"状态。如果在"trying"状态的时候收到了一个临时应答，那么这个应答必须转给 TU 处理，并且客户端事务转到"proceeding"状态。如果在"trying"状态收到了一个终结应答（200－699 的应答码），那么应答必须交给 TU，并且客户端事务必须转到"Completed"状态。

如果定时器 E 在"Proceeding"状态触发了，那么请求必须交给通讯层进行传输，并且定时器 E 必须重新设置成为 T2 秒。如果定时器 F 在"Proceeding"状态触发了，那么必须通知 TU 超时了，并且客户端事务必须转到终结状态。如果在"Proceeding"状态的时候收到了一个终结应答（状态码 200－699），这个应答必须发送给 TU，并且客户端事务必须转到"Completed"状态。

一旦客户端事务进入"Completed"状态，对于非可靠传输的情况，客户端事务必须设置一个定时器 K＝T4 秒，对于可靠传输的情况，设置定时器 K=0 秒。这个"Completed"状态维持的目的是为了缓冲可能会收到的其他重发的应答（这是为什么客户端事务在这里为非可靠传输维持一段时间的原因）。T4 代表了网络在客户端和服务端事务中传输信息可能的时间。缺省的值 T4＝5 秒。当应答具有相

同的事务匹配的时候，根据 17.1.3 的判定，这个应答就是重发的应答。如果定时器 K 在这个状态被触发，客户端事务必须转到”Terminate”状态。

当事务进入终结状态，就必须立刻终止了。

17.1.3 客户端事务匹配应答

当客户端事务的通讯层收到一个应答，他必须决定是否由客户端事务来处理这个应答，这样 17.1.1 和 17.1.2 才能够正确执行。在 Via 头域的最上边的 branch 参数就是用来做这个的。一个应答和一个客户端事务匹配的话，就有两个条件：

1、 如果应答 Via 最上边的 branch 参数和创建这个客户端事务的请求的 Via 最上边的 branch 参数相同。

2、 如果 Cseq 头域的方法参数和创建事务的请求的方法相同。这是因为 CANCEL 方法的事务和源请求的事务不同，但是却有相同的 branch 参数所决定的。

如果一个请求是广播发送的，他可能从不同的服务器上得到不同的应答。这些应答的最上边的 Via 都有相同的 branch 参数，但是在 To tag 中是不同的。当收到了第一个应答，基于上边的规则，将会判定是这个客户端事务的应答，其他的应答将会视同为重发。这并不是错误的情况；多点传送 SIP 只是提供了一个根本的”寻找最接近的单点”服务的方法，这样就限定了只需要处理一个单个应答。详情参见 18.1.1。

17.1.4 处理通讯错误。

timer f
or transport err
inform TU

Terminated
TIMER K fires

Completed
200-699
resp. to TU
Proceding
1xx
resp to TU
1XX
resp. to TU
Timer E
Transport Err
inform TU
request from TU
send request
trying

Timer E
Send Request
TIMER E
send req
200-699
resp. to TU

图 6：非 INVITE 客户端事务

当客户端事务发送一个请求到通讯层发送的时候，如果通讯层报告发送失败，那么需要执行下列步骤。

客户端事务应当通知 TU 这个通讯失败，并且客户端事务应当直接转到”Terminate”状态。TU 处理通讯失败的机制在附件[4]中描述。

17.2 服务端事务

服务端事务是用来传输请求到 TU 并且可靠的传输应答的。它是通过状态机来实现的。服务端事务是当请求到达的时候由核心创建的，事务的处理也是主要围绕着对应请求的（也就是说并非全部都是和对应请求相关）。

和客户端事务对应的，状态机依赖于是否接收的请求是 INVITE 请求。

17.2.1 INVITE 服务端事务

INVITE 服务端事务的状态图在图 7 表达。

当为一个请求创建了服务端事务的时候，服务端事务进入”proceeding”状态。除非服务端事务知道 TU 在 200ms 之内会生成临时或者终结应答（在这种情况下，TU 可能会产生 100Trying 应答），他必须生成 100（Trying）应答。这个临时应答是用来停止客户端重发请求的，这个可以避免网络风暴。这个 100（Trying）应答是根据 8.2.6 节描述的步骤构造的，除此之外： 如果接收的请求头中的 To 头域没有 tag 标志，那么原来描述的**可以**增加 tag 标记，更改成为**不应该**增加 tag 标志。这个请求必须交给 TU 处理。

TU 可以给服务端事务任意数量个临时应答。只要服务端事务在”proceeding”状态，每个临时应答都应当交给通讯层发送。这些临时应答并非被通讯层可靠的发送（他们并不重新发送临时应答）并且临时应答并不改变服务端事务的状态。如果在”proceeding”状态，收到一个请求的重发请求，那么就需要把从 TU 最近收到的那个临时应答重新交给通讯层发送一次。请求是否是重发的请求，是基于 17.2.3 来判定的匹配相同服务端事务的请求。

如果，在”proceeding”状态，TU 发送了一个 2xx 应答给服务端事务，服务端事务必须把这个应答交给通讯层进行发送。这个并非由服务端事务进行重发；对于 2xx 应答的重发是由 TU 处理的。服务端事务必须转到”Terminated”状态。

当在”Proceeding”状态的时候，如果 TU 交给服务端事务一个 300 到 699 的应答，那么应答必须交给通讯层进行发送，并且状态机必须进入”Completed”状态。对于非可靠传输的情况，必须设置定时器 G=T1 秒，对于可靠传输的情况，不设置定时器 G（=0 的情况就是不设置）
这个是和 RFC2543 所不同的，2543 要求应答都要重发，甚至在可靠传输的情况下。

当进入了”Completed”状态，必须为所有的传输，设置一个定时器 H＝64×T1 秒。定时器 H 决定何时服务端事务取消重发应答。这个值和定时器 B 的取值一样，是等同于客户端事务会重试发送请求的时间。如果定时器 G 触发了，那么应答会交给通讯层再次发送，并且定时器设置成为 MIN(2*T1,T2)秒。依此类推，当定时器 G 再次触发，那么定时器 G 的值会翻倍，直到 T2。这个和非 INVITE 客户端事务的”trying”请求的重发机制是一样的。进一步说，当在”Completed”状态的时候，如果接收到重发的请求，服务端事务应当把应答交给通讯层再次发送。

当服务端事务在”Completed”状态的时候，如果收到了一个 ACK 请求，服务端事务必须转到”Confirmed”状态。因为定时器 G 会在这个状态被忽略，所有的应答重发都会被终止。

如果在”completed”状态的时候，定时器 H 触发了，就意味着没有收到 ACK 请求。在这个情况下，服务端事务必须转到”Terminated”状态，并且必须通知 TU 事务失败。

2xx from TU
send response
Timer I fires
-
Terminated
Timer H fires
or Transport Err
Inform TU
ACK
-
Confirmed
INVITE
send response
101-199 from TU
send response
300-699 from TU
send response
INVITE
pass INV to TU
send 100 if TU won’t in 200ms
Proceeding
Timer G fires
send response

图 7：INVITE 服务端事务

设定”Confirmed”状态的目的是为了处理任何附加的 ACK 消息，这是由重发的终结应答所触发的。当进入这个状态，如果是在不可靠传输协议，那么就要设定一个定时器 I＝T4 秒，如果是可靠传输协议，那么就设定 I=0。当定时器 I 触发了，服务端事务必须转到”Terminated”状态。

当服务端事务状态处于”Terminated”状态，这个事务必须立刻销毁。和客户端事务一样，这是为了保证给 INVITE 的 2xx 应答的可靠性。

17.2.2 非 INVITE 服务端事务

对非 INVITE 服务端事务的状态机是在图 8 中表示。

当收到一个不是 INVITE 或者 ACK 的请求的时候，状态机会初始化成为”trying”状态。并且这个请求会交给 TU 处理。当在”trying”状态，任何重发的请求会被忽略。一个请求在通过 17.2.3 节的步骤，匹配现有的服务端事务，将被认为是重发的请求。

当处于”trying”状态，如果 TU 交给服务端事务一个临时应答，服务端事务应当进入”Proceeding”状态。这个应答必须交给通讯层进行发送。在”Proceeding”状态下从 TU 收到的任何应答都必须交给通讯层进行发送。如果一个重发的请求在”proceeding”状态下收到了，那么最近发出的一个临时应答应当再次交给通讯层进行重发。如果在”Proceeding”状态下，TU 交给服务端事务一个终结应答（应答码是 200－699），那么服务端事务必须进入”Completed”状态，并且应答必须交给通讯层进行发送。

当服务端事务进入了”Completed”状态，对于不可靠传输协议来说，必须设定一个定时器 J＝64×T1 秒，对于可靠传输来说，设定为 0 秒（就是不设定定时器）。当在”Completed”状态下，当服务端事务收到了一个重发的请求的时候，

服务端事务必须交给通讯层终结应答来重新发送。在”Completed”状态下，任何其他 TU 传递下来给服务端事务的终结应答都必须被抛弃。服务端事务保持这个状态直到定时器 J 触发，当定时器 J 触发了以后，服务端事务必须进入”Terminated”状态。

17.2.3 为服务端事务匹配请求。

当服务端从网络上收到一个请求以后，他必须和现有的事务进行判定。这个是根据下边的规则来判定的。

首先要检查请求中的 Via 头域的最上一个 branch 参数。如果他以”z9hG4bk”开头，那么这个请求一定是由客户端事务根据本规范产生的。因此，branch 参数在该客户端发出的所有的事务中都是唯一的。根据下列规则我们可以判定请求是否和事务匹配：

1、 请求中的最上的 Via 头域的 branch 参数和创建本事务的请求的最上的 Via 头域的 branch 参数一样，并且：
2、 请求的最上的 Via 头域的 sent－by 参数和创建本事务的请求的最上的 Via 头域的 send-by 参数一样，并且：
3、 请求的方法和创建本事务的方法一样。这有一个例外，就是 ACK,ACK 对应的创建本事务的请求方法是 INVITE。
这个匹配规则用于 INVITE 和非 INVITE 事务。

send-by 参数被用于匹配过程，这是因为有可能存在无意/恶意的相同的不同客户端传来的 branch 参数。

如果最上的 Via 头域的 branch 参数不存在，或者没有包含那个”z9hG4bk”，那么就用下列步骤进行判定。这是为了和 RFC2543 进行兼容的。

如果是 INVITE 请求，并且这个 INVITE 请求的 Request-URI,To tag,From tag,Call-ID,Cseq,和最上的 Via 头域都和创建事务的 INVITE 请求的这些字段匹配，那么这个 INVITE 请求就是匹配这个事务的 INVITE 请求。在这个情况下，INVITE 就是创建这个事务的 INVITE 请求的一个重发。ACK 请求在匹配创建事务的 INVITE 请求的 Request-URI, From tag, Call-ID ,Cseq 序列号（非方法字段）， 最上的 Via 头域，并且 To tag 和服务端事务发出的应答的 To tag 相同，这个 ACK 就是这个事务的 ACK。当这些头域比较完成，那么这个匹配也就完成了。在 ACK 比较中包含 To tag 的比较是为了在 proxy 上能够区别给 2xx 的 ACK 和给其他应答的 ACK，这个 proxy 可能会转发全部的应答（这个会在某种罕见的情况下发生。特别是，当一个 proxy 分支一个请求，接着宕机了，应答会转发到别的 proxy，这个 proxy 可能会终止转发多重应答到上行队列）。一个匹配 INVITE 请求事务的 ACK 请求，如果这个 INVITE 请求已经被前一个 ACK 请求所匹配，那么这个 ACK 请求就是上一个 ACK 请求的重发。

transport Err
inform TU
timer J fires
-
Request
send response
transport Err
inform TU
Request
send response
Request received

pass to TU
1xx from TU
send response
1xx from TU
send response
200-699 from TU
send response
200－699 from TU
send response
Completed
Terminated
Trying

图 8： 非 INVITE 服务端事务

对于所有的其他请求方法，如果请求的 Request-URI,To tag,From tag，Call-ID， Cseq（包括 Cseq 中的方法字段），以及 Via 头域的最上值，都和创建服务端事务的请求想匹配，那么这个请求就是这个事务的匹配请求。匹配是基于针对每一个头域值的判定进行的。当非 INVITE 请求和现有事务匹配了，那么它就是创建这个事务的请求的一个重发。

由于匹配规则中包含了 Request-URI,服务器不能匹配应答对应到事务。所以当 TU 传送了一个应答到服务端事务，它必须为这个应答指定传送到那个服务端事务。

17.2.4 处理通讯错误

当服务端事务发送一个应答到通讯层要发送的时候，如果通讯层报告发送失败，那么就需要执行下列的步骤：

首先，附件[4]的步骤需要执行，这就是说需要把应答发送一个备份的地点。如果这个也失败了，基于[4]中对失败的定义，服务端事务应当通知 TU 发送失败，并且把状态切换到终止状态。

18 通讯（transport）

通讯层负责请求和应答在网络上的实际传输。这包括了在面向连接的通讯方式下的请求和应答所使用的连接管理。

通讯层负责管理像TCP/SCTP之类通讯协议的长连接，或者在这些协议上的TLS连接，并且包括管理打开这些连接的使用者的管理。这包括了客户端或者服务端通讯层打开的连接，这样在客户端服务端通讯函数可以共享这些连接。这些连接采用一组用远端的地址，端口，通讯协议标志的索引来进行管理。当通讯层打开了一个连接，这个连接的索引就设置成为远端的IP,端口,还有打开这个连接的通讯层的实例[2]。当通讯层接收了一个连接，那么这个连接的索引就被设置成为连接方的源IP地址，port，还有通讯层的实例transport。注意，由于源端口port通常是临时创建的，但是由于通过附件[4]的步骤不能知道它是临时创建的还是配置的，所以通讯层被动接收的连接通常是不被重复使用的。这就是说，如果两个proxy再一个”peering”(点对点)的关系中，使用一个面向连接的通讯协议通常有两个连接要使用，每个都是自己作为主动方连接的。

我们建议在实现中，当发送（或者接收）完成最后一个消息之后，依旧维持这个连接一段时间（这段时间可以是实现自己定义的时间）。这段时间应当是至少等于本

节点的事务从创建到结束的最长时间。这是为了让事务能够在他们所创建的同一个连接上完成（比如，在这个连接上完成请求，应答的处理，在 INVITE 的情况下的给非 2xx 的 ACK 应答等等）。这通常意味着至少 64×T1 秒（参见 17.1.1.1 中关于 T1 的定义）。不过，如果当本程序的 TU 使用的是一个比较大的定时器 C（参见 16.6 节 11 步）的时候，也可以选取一个比较大的值。

所有的 SIP 元素都必须实现基于 UDP 和 TCP 的通讯。SIP 元素还可以实现其他的协议。

要求 UA 支持 TCP 是对 RFC2543 的一个重要改进。这是由于需要处理更大的消息，就像接下来讲到的那样，必须使用到 TCP 协议。因此，即使是 SIP 元素不要发送大的消息，但是由于它可能收到大消息并且处理这些消息，所以，要求支持 TCP。

18.1 客户 Clients

18.1.1 发送请求

通讯层的客户端负责发送请求和接收应答。通讯层的用户把请求交给通讯层的实例进行处理，包括 IP 地址端口，通讯层实例，还有可能有多点广播的 TTL。

如果请求的大小和 MTU 差是在 200 个字节以内的，或者它是大于 1300 字节的，并且路径 MTU 的大小是未知的，那么请求必须遵循 RFC2914[43]控制阻塞的传输协议，比如使用 TCP。如果这导致了 Via 最上边指定的通讯协议的改变，那么 Via 最上边的值就必须也随之改变。这使得在 UDP 传输上的消息的分割，并且也提供了大消息的传输阻塞控制。不过，在实现上，必须能够支持达到最大包大小的消息的处理。对于 UDP 来说，包含了 IP 和 UDP 头的大小是 65535 个字节。

在消息的大小和 MTU 之间的 200 个字节的”buffer”，提供了一个机制使得在 SIP 的应答中，可以超过请求的大小。比如在 INVITE 请求的应答中，增加了 Record-Route 头域值。有了这个额外的 buffer，应答可以大概比请求大 170 个字节，而且在 Ipv4 上不用进行分块传输（假设没有 IPSec，大概 IP/UDP 会使用 30 个字节）。当 MTU 是未知的时候，选取 1300 是基于假设 Ethernet 的 MTU 是 1500 字节的基础上。

如果 SIP 元素是因为消息大小的限制，所以基于 TCP 发送一个请求，并且消息如果不是因为大小的限制，会使用 UDP 来发送，并且如果建立连接产生一个 ICMP 协议不支持的错误，或者导致 TCP reset，那么这个元素就应当用 UDP 重试这个请求。这只是为了向后兼容 RFC 2543 针对不支持 TCP 的实现。在本规范以后的改动中，这部分内容会有修订。

如果客户端向多个地址发送请求，那么必须增加”maddr”参数到 Via 头域值上，并且这个参数值指定多个目的地址，对于 Ipv4 来说，应当增加”ttl”参数＝1，IPV6 的多点传送在本规范中没有定义，会在后续的标准中描述。

这些规则定义了 SIP 的多点传送。首要的目的是为了提供”寻找最接近的单点”服务（”single-hop-discovery-like”），这个服务将请求转发到一组类似的服务器，并且只需要处理其中任意一个服务器的应答。这个功能主要用于注册服务。实际上，基于 17.1.3 的事务处理规则，客户端事务会接收第一个应答，并且因为其他应答包含同样的 Via 的 branch 参数，而视这些应答为重发应答。

在请求发送嵌，客户端通讯层必须在 Via 头域中增加一个”sent-by”栏。这个字段包含了一个 IP 地址或者主机名，端口。我们推荐使用 FQDN 方法描述这个主机名。这个字段在某些特定情况下，用于发送应答。如果端口不存在，缺省的值依赖于通讯协议。对于 UDP，TCP 和 SCTP 来说是 5060，TLS 是 5061。

对于可靠传输协议，应答通常简单的通过连接发送，并且这个连接是收到对应请求的连接。因此，客户端传输层必须准备在发出请求的同一个连接上接收应答。在出现错误的情况下，服务端可能会尝试新建立一个连接来发送应答。为了能够处理这种情况，通讯层必须准备接收一个从源 IP 建立的新连接，这个连接的 IP 是请求发起的源 IP，port 是在”sent-by”字段中指定的 port。这也同样要求准备接收从任意地址和端口来得新连接上接收应答，这个端口是由服务器根据附件[4]的 5 节所讲述的步骤来选取的。

对于非可靠的传输协议，客户端通讯层必须准备从发送请求的那个原始 IP 地址上接收应答。（因为应答会送到原始地址去），并且端口号是在”sent-by”字段的端口好。进一步说，和可靠传输一样，早某些情况下，应答会发往不同的地方。客户端必须能够准备从其他地址和端口上接收应答，这个端口是由服务器根据附件[4]的 5 节所讲述的步骤来选取的。

对于多点传送的情况来说，客户端通讯层必须准备从相同的多点传输组上接收应答，这个组的地址和端口和发出请求的组相同（就是说，它必须是发送请求的那个多点传输组的一个成员）。

如果请求发送的目的 IP 地址，端口和 transport 都和现有的一个连接相同，那么建议使用这个连接来发送请求，同时也允许新建立一个连接来发送。

如果请求通过多点发送，那么它发送的一组地址，端口和 TTL 都是由通讯层的用户提供。如果请求是通过不可靠通讯协议发送，那么发送的 IP 地址和端口也是由通讯层的用户提供。

18.1.2 接收应答

当应答接收到的时候，客户端通讯层检查最上的 Via 头域值。如果”sent-by”参数不符合客户端通讯层在请求中插入的值，那么这个应答必须悄悄丢弃。

如果由任何客户端事务存在，客户端通讯层使用 17.1.3 的步骤来匹配现存的事务和这个接收到的应答。如果匹配到了，应答必须交给事务层进行处理。否则，应答必须交给核心去处理（无论是有状态的 proxy，还是无状态的 proxy，还是 UA 的核心）。处理这些”stray”(迷路)的应答是基于核心的策略的（如果是 proxy 就会转发，如果是 UA 就会忽略，等等）。

18.2 服务端

18.2.1 接收请求

一个服务器应当能够接收从任何 IP 地址、端口和协议上过来的请求。他们是通过对这个服务器的 SIP 或者 SIPS URI（附件[4]）的 DNS 查找，得到这个服务器的地址然后连接和发送的请求的。在这里，”handing out”（发布）包含了在 REGISTER 请求或者转发应答的 Contact 头域中放一个 URI，或者在请求或者应答的”Record-Route”头域中放一个 URI。这个 URI 可以通过放在网页或者名片上被”handing out”(发布)。同样的我们也建议服务器在公网上监听缺省的 SIP 端口（TCP/UDP 是 5060，5061 是在 TCP 上的 TLS）。如果是在局域网上，或者私有网上，或者一个物理服务器上运行好几个服务实例，那就很自然的可以设置成不同的。对于服务器监听 UDP 的任何端口和界面，都必须在 TCP 上也进行同样的监听。这是因为可能消息还需要通过 TCP 进行传输，比如消息过大的情况。所以，在相反的情况下就不需要了。如果一个服务器在 TCP 监听了，那么它不一定需要在 UDP 上也进行相应的监听。当然服务器也可以因为某些原因在特定地址和端口上监听 UDP。当服务端事务从任意一个通讯层上接收到一个请求的时候，它必须检查最上的 Via 头域的”sent-by”参数。如果”sent-by”参数的主机部分包含了一个主机名，或者它包含的 IP 地址和包的源地址不同，服务器必须增加一个”received”参数到这个 Via 头域值中。这个参数必须包含收到的包的原地址。由于服务端必须把应答发送给收到请求的那个源 IP 地址，所以这个可以用来帮助服务端通讯层发送应答。

一个服务端通讯层收到的请求可能是这样的（部分）：

INVITE sip:bob@Biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060

请求是从源 IP：192.0.2.4 收到的。在请求转交到上层之前，通讯层增加了一个”received”参数，这样请求的部分就是：
INVITE sip:bob@Biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060；received=192.0.2.4

接着，服务端通讯层尝试和服务端事务做匹配。这个使用的是 17.2.3 节定义的规则。如果匹配上一个服务端事务，那么请求就交给那个事务去处理。如果没有匹配到事务，请求就交给核心去处理，可能会创建一个新的服务端事务来处理。注意当 UAS 核心给 INVITE 请求发送一个 2xx 应答的时候，服务端事务已经销毁了。这就是说，当 ACK 收到的时候，不会有匹配的服务端事务，并且基于这个规则，ACK 回交给 UAS 核心来处理。

18.2.2 发送应答

服务端事务使用最上边的 Via 头域值来决定把应答发送到哪里。它必须遵从如下步骤来发送：

o 如果”sent-protocol”是一个可靠的传输协议比如 TCP 或者 SCTP，或者在其上的 TLS，应答必须用现存的到原始请求（创建这个事务的请求）的连接进行发送（如果连接还存在的情况下）。这个要求服务端通讯层保留服务端事务和通讯层连接的相关性。如果连接不存在了，服务端应当创建一个新的连接，如果存在”received”参数，就用对应的在”received”参数中指定的 IP 地址。如果存在”sent-by”参数，那么就用”sent-by”指定的 port，如果不存在，那么就用缺省

的 port。如果对应的连接已经失效，那么服务器应当采用附件[4]的步骤来决定使用那个 IP 地址和端口来建立连接并且发送应答。

o 否则，如果 Via 头域包含一个”maddr”参数，就必须把应答转发到 maddr 所指明的地址，并且使用”sent-by”所指定的端口，如果没有 sent-by 参数，那么就使用 5060 缺省参数。如果地址是一个多点地址，应答应当使用”ttl”参数所指定的 TTL，或者如果没有指定”ttl”参数，则使用 TTL=1 的参数。

o 否则（对于非可靠传输），如果 Via 的最上头域包含一个”received”参数，那么应答必须发送到”received”参数所指定的地址，并且使用”sent-by”所指定的端口，如果没有 sent-by 参数，那么就使用 5060 缺省参数。如果这步失败了，比如，如果得到一个 ICMP”端口不能到达”的错误，那么就应当根据附件[4]的第 5 节的步骤来决定应当把应答发送到哪里。

o 否则，如果没有 receiver-标记，那么应答应当使用附件[4]的第 5 节指定的步骤，送到”sent-by”参数指定的地址。

18.3 分块

在面向消息的通讯协议中（比如 UDP），如果消息有一个 Content-Length 头域，那么消息体就有可能包含很多字节。并且收到的包中除了这个消息体的 Content-Length 字节意外，还有通讯层附加的通讯包字节，那么这部分额外的字节应当被丢弃。如果通讯包在没有收到完整的 Content-Length 字节的消息体就终止了，这就意味着出错了。如果这个消息是一个应答，那么这个消息必须被丢弃。如果消息是一个请求，那么本程序应当给出一个 400（Bad Request）应答。如果消息没有包含一个 Content-Length 头域，消息体的结束点就是消息体的结束点。

在面向流的通讯协议中（比如 TCP），Content-Length 头域标志这包体的大小。在面向流的通讯协议中，必须使用 Content-Length 字段。

18.4 错误处理

错误的处理取决于出现错误的消息是请求还是应答。

如果通讯层的用户要求在一个非可靠传输协议上发送一个消息，并且结果是一个 ICMP 错误，那么错误处理的方法依赖于 ICMP 错误类型。当通讯层遇到主机、网络、端口或者协议无法到达的错误，或者参数错误的时候，应当通知通讯层的用户发送失败。Source quench 和 TTL exceeded ICMP 错误应当被忽略。

如果通讯层用户要求在一个可靠传输协议上发送一个请求，并且结果是一个连接错误，通讯层应当通知通讯层用户这个发送错误

19 常见消息部件(Common Message Components)

在 SIP 消息中，有一些很长用的部件。（甚至在 SIP 消息外这些部件也存在）。这些部件值得我们单独讨论一下。

19.1 SIP 和 SIPS 统一资源标记

SIP 或者 SIPS 的 URI 用来标记一个通讯用的资源。就像其他所有的 URI 一样，SIP 和 SIPS URI 可以放在网页上，email 消息里，或者打印出来的名片上等等。在这些 URI 里边包含了足够的信息来发起和维持到这个资源的一个通讯会话。

一个通讯资源的例子包含下列内容：

o 一个在线服务的用户

o 一个多线电话

o 消息系统中的邮箱

o 网关服务的 PSTN 电话号码

o 一个组织中的一个部门（比如”销售”,或者”helpdesk”）

SIPS URI 定义了对资源的访问是安全的。这就意味着，特别是，在 UAC 和这个资源的主机之间的通讯是基于 TLS 的。从资源的主机到用户之间的通讯是加密安全的，这个安全机制是依赖于主机的实现的。任何用 SIP URI 描述的资源，只要想通过加密的形式进行通讯，都可以通过简单改变一下资源描述府就可以”升级”成为一个 SIPS URI。

19.1.1 SIP 和 SIPS 部件

“sip:”和”sips:”描述符是遵循 RFC2396[5]的规范定义的。他们使用类似 mailto URL 的格式定义，允许有 SIP 请求头域字段和 SIP 消息体的规范。这使得在网页上或者 email 中，可以用 URI 来初始化一个会话，这个会话有特定的主题，媒体类别，紧急类型。这个 SIP 或者 SIPS URI 的格式规范在 25 节定义。一个 SIP URI 的通常格式是这样的：

sip: user:password@host:port;uri-parameters?headers

这个和 SIPS URI 的格式是相同的，只是 SIPS 用”sips”来代替 sip。这些符号，和符号的扩展，具有下列意义：

user： 这是在主机的特定资源地址。”主机”(host)在这里通常指的是一个域名。URI中的”userinfo”包含了这个用户域，口令域，并且包含其后的一个@。URI的用户信息部分是可选的，或者说是可以没有的；当目的主机没有用户的概念或者主机本身就是资源的目标，那么这个URI的用户信息部分就是可以没有的。如果在

SIP或者SIPS，那么用户部分必须不能为空的。如果主机部分可以处理电话号码地址，比如说是一个internet电话网关，那么根据RFC2806[9]定义的电话号码域应当出现在用户信息部分。在 19.1.2 节有关于在SIP或者SIPS URI中的电话号码描述域的额外说明 URI中有@

password：password 字段是和用户相关的。SIP 或者 SIPS URI 语法允许增加 password 这个字段，这种用法我们是不推荐的，因为把身份认证信息放在明码表示的地方（比如 URI）会带来很大的安全风险。比如，通讯层在这个字段用了一个 PIN 码，那么就会暴露这个 PIN 码而带来安全隐患。

注意密码字段只是一个用户信息扩展的一部分。实现上并没有标记一个特别的密码部分，可以简单的把”user:password”当作一个简单的用户串来对待。

host：主机提供了 SIP 资源。host 部分包含了一个完整的主机名字或者 IPV4/IPV6 的地址。我们强烈建议如果可能，就使用完整格式的主机名字。

port： 端口号是请求将被送出的端口。

URI 参数：请求将使用这个 URI 来构造。

URI 参数在 hostport 部件之后增加，用分号分开。
URI 参数有如下格式：

参数名’=’参数值

虽然在同一个 URI 中允许有任意多个 URI 的参数，但是同一个参数名只能出现 1 次。

这个扩展机制包括了 transport，maddr,ttl,user,method 和 lr 参数

transport 参数决定在[4]中定义的发送 SIP 消息的通讯机制。SIP 可以使用任何网络通讯协议。参数名字是为 UDP（RFC 768[14]）,TCP(RFC 761 [15])和 SCTP（RFC2960[16]）定义的。对于一个 SIPS URI,transport 参数必须指向一个可靠的通讯协议。

maddr 参数指明了联系这个用户的服务器的地址，它会覆盖在 host 域中的地址。当给定了一个 maddr 参数，URI 中的 port 和 transport 部件将会在 maddr 中指出。[4]描述了正确的 transport，maddr,hostport 规范，用于获得发送请求到目的地所需要的目的地址，端口，通讯协议。

maddr 字段用作简单的去掉源路由的方法来使用的。它允许一个 URI 指定一个必须经过的 proxy 来到达目的地。我们强烈建议不要把 maddr 参数用于这个目的（我们反对把 maddr 用于这个机制）。在实现上应当使用本文中描述的 Route 机制，如果有需要，则建立一个 pre-existing（预先设置的）路由集合（参见 8.1.1.1）。它提供了一个完整的 URI 来描述需要经过的节点。

ttl参数决定了UDP多点报文的生存周期，并且只能用于maddr是一个多点地址并且通讯协议是UDP的情况。例如，为了指定一个到alice@atlanta.com的呼叫，使用多点广播到 239.255.255.1，并且ttl＝15，那么应该使用下边的一个URI：

sip:alice@atlanta.com;maddr=239.255.255.1;ttl=15

有效的电话描述（telephone-subscriber）的字串一个集合是有效的用户字串的子集。我们用用户URI参数来区别电话号码和用户名（长得像电话号码的用户名）。如果用户串使用了电话号码描述的字串，用户参数值”phone”应当增加。

即使没有这个参数，如果本地用户名的命名限制机制允许的情况下，SIP和SIPS URI的接受方也可以把这个@以前的部分解释为电话号码。

从 URI 中构建 SIP 请求所需要的 method 域，可以由 method 参数指定。

如果指定了 lr 参数，就标志着这个资源的拥有者是根据本规范来实现的路由机制。这个参数回用于 proxy 放在 Record-Route 头域的 URI 中，也可以出现在 pre-existing(预先设置)的路由集合中。

这个参数是用来和 RFC2543 定义中的严格路由机制向后兼容所使用的，并且 rfc2543bis 改变为 bis-05。如果一个元素准备发送一个基于没有包含这个参数的 URI 请求，那么我们可以假定这个请求的接受方是根据严格路由的规范实现的，并且会重新规格化这个消息来保护在 Request-URI 中的内容。

由于 URI 参数机制是可以扩展的，SIP 元素应当悄悄跳过那些不认识的 uri 参数。

Headers：头域是从给定 URI 创造的请求的头域部分。
在 SIP 请求中的头域可以在 URI 中用”?”来给出。头域名(hname)和头域值(hvalue)都是用&符号间隔的头域名=头域值的格式。特定的头域名”body”的头域值就是 SIP 请求的消息体。

表 1 总结了在 URI 的不同情况下 SIP 和 SIPS URI 的部件用法。扩展的列描述了在 SIP 消息歪的 URI，例如在网页上或者名片上的情况。项目中的’m’是强制必须的意思，’o’是可选的，’-‘是不允许的。处理 URI 的元素应当忽略掉 URI 中出现的任何不允许的部件。在表格中的第二列是如果该元素不存在的时候的缺省值。’--' 表示本元素不是可选的，或者没有缺省值的意思。

在 Contact 头域中的 URI 在头域出现的不同地方有着不同的约束。一个是在消息建立和维持一个对话的时候（INVITE 请求以及它对应的 200(OK)应答），一个是在注册和转发消息的时候（REGISTER,以及对应的 200（ok）应答，以及给任何方法的 3xx 系列的应答）

19.1.2 Character Escaping Requirements（字符转码要求）

	default	Req-URI	To	From	reg./redir. Contact	dialog Contact R-R/Route	external
user	- -	o	o	o	o	o	o
password	- -	o	o	o	o	o	o
host	- -	m	m	m	m	m	m
port	(1)	o	-	-	o	o	o
user-param	ip	o	o	o	o	o	o
method	INVITE	-	-	-	-	-	o
maddr-param	- -	o	-	-	o	o	o
ttl-param	1	o	-	-	o	-	o
transp.-param	(2)	o	-	-	o	o	o
lr-param	- -	o	-	-	-	o	o
other-param	- -	o	o	o	o	o	o
headers	- -	-	-	-	o	-	o

(1):缺省的通讯端口是依赖于通讯协议的。对于使用 UDP,TCP,SCTP 的 sip 来说，是 5060,对于使用基于 TCP 的 TLS 来说，是 5061。

(2) 缺省的通讯协议是和 sip/sips 相关的，对于 sip 来说，是 UDP,对于 sips 来说，是 TCP。

表 1：对于 SIP 头域值，Request-URI 及其引用的使用和缺省值。

基于 RFC2396[5]的要求和指引，当需要把字符串封装到 SIP URI 的时候，使用””%” HEX HEX”机制来进行转码。根据 RFC2396[5]:

任何指定 URI 部件保留的字符集都是由这个部件定义的。通常，如果 URI 的语义由于组成字符被它的 US-ACII 编码[5]的 escape 码替代而改变的时候，这个字符就是保留字符。除了 USASCII 字符（RFC2396[5]）之外，比如空格和控制字符，以及 URI 所使用的分隔符，必须进行转码。URI 必须不能包含任何未经转码的空白和控制字符。

对于每一个部件来说，由 BNF 扩展的合法字符集合规定了那些字符是可以不经转码的。其他字符都必须经过转码。

比如，”@”不是user部件的字符集中的字符，所以，userj@sOn，必须把@符号进行编码，成为”j%40sOn”

在 25 节中的 hname 和 hvalue 的符号展示了在 URI 的保留字符中，在头域名和头域值中所有需要被转码的字符集合。

user 部件的电话描述(telephone-subscriber)部分由特别的转码考虑。在 RFC2806[9]关于电话描述部分中未被保留的字符集，由很多字符组成，他们在 SIP URI 中的不同语法部分的时候，都需要做转码。在电话描述中出现的任何字符，只要不在 BNF 针对 user 部分的扩展规则中出现的，都需要做转码。

注意在 SIP 或者 SIPS URI 中，host 部分不允许做字符的转码（%不在它的扩展部分中）。在以后的 Inernationalized Domain Names 完成以后，这个限制可能就会改了。当前实现中不允许把在 host 部分收到的转码字符进行转码处理。因为这个转码处理和 IDN 要求的处理不一样。

19.1.3 SIP 和 SIPS URI 例子

sip:alice@atlanta.com
sip:alice:secretword@atlanta.com;transport=tcp
sip:alice@atlanta.com?subject=project%20x&priority=urgent
sip:+1-212-555-1212:1234@gateway.com;user=phone
sips:1212@gateway.com
sip:alice@192.0.2.4
sip:atlanta.com;method=REGISTER?to=alice%40atlanta.com
sip:alice;day=Tuesday@atlanta.com

最后一个 URI 例子有一个 user 域”alice;day=Tuesday”。上边定义的转码规则中允许”;”在这个字段中不进行转码。在本协议的设计概念中，这个字段是不透明的。这个字段的值只对负责这个资源的 SIP 元素有用。

19.1.4 URI 比较

在本规范中，部分操作需要比较两个 SIP 或者 SIPS URI 是否相等。比如，在这个规范中，注册服务器需要比较在 REGISTER 请求中绑定的 Contact URI（参见 10.3）。SIP 和 SIPS URI 根据如下步骤进行比较：

o SIP 和 SIPS URI 永远不等。
o SIP/SIPS URI 的 userinfo 是大小写敏感的。这包括了含有 password 或者按照电话描述格式的 userinfo 的比较。对于 URI 的其他部分的比较，除了有特别指出之外，都是大小写不敏感的。

o 参数的顺序和头域的顺序对于比较 SIP/SIPS URI 不起作用。
o 在保留字符集之外的字符（参见 RFC2396[5]），等同于他们的””%” HEX HEX”格式。
o IP 地址就算是等同于通过 DNS 查找到的主机名对应的 IP 地址，IP 地址也不能和主机名等同。
o 两个 URI 如果相同，那么 user，password，host，port 部分必须相同。

有 user 部分的 URI 和没有 user 部分的 URI 是不相等的。有 password 部分的 URI 和没有 password 部分的 URI 也是不同的。

一个不带可选部件的 URI 和带了这些部件但是值是缺省值的 URI 是不等的。例如，如果一个 URI 省略了 port 部件，并不等于一个定义了 5060port 部件的 URI。同样的规则适用域 transport-参数，ttl-参数，user-参数，method 部件等等。

定义 sip:user@host，和定义 sip:user@host:5060（根据 RFC2543 的变体）不相等。当从 URI 中取得地址的时候，相同的 URI 可以取得相同的地址。sip:user@host:5060 始终可以得到端口 5060。URI: sip:user@host 根据[4]所定义的 DNS SRV 机制，可能可以得出其他的端口来。

o URI uri 参数部件按照如下规则进行比较
- 任何在两个 URI 中出现的 uri 参数都必须一样
- user,ttl,或者方法 uri 参数如果只在一方出现，即使和缺省值相等，也判定为两个 URI 不相等。
- 包含 maddr 参数的 URI 和没有包含 maddr 参数的不相等。
- 其他 uri 参数，如果在一方出现，则在比较的时候忽略。

o URI 头部件的比较是不能忽略的。任何在 header 部分出现的域都必须在双方 URI 中进行匹配和比较。比较规则参见 20 节。

下列 URI 是相等的：

sip:%61lice@atlanta.com;transport=TCP
sip:alice@AtLanTa.CoM;Transport=tcp

sip:carol@chicago.com
sip:carol@chicago.com;newparam=5
sip:carol@chicago.com;security=on

sip:biloxi.com;transport=tcp;method=REGISTER?to=sip:bob%40biloxi.com
sip:biloxi.com;method=REGISTER;transport=tcp?to=sip:bob%40biloxi.com

sip:alice@atlanta.com?subject=project%20x&priority=urgent
sip:alice@atlanta.com?priority=urgent&subject=project%20x

下列 URI 是不相等的：

SIP:ALICE@AtLanTa.CoM;Transport=udp (用户名不同)
sip:alice@AtLanTa.CoM;Transport=UDP

sip:bob@biloxi.com (端口不同)
sip:bob@biloxi.com:5060

sip:bob@biloxi.com (通讯协议不同)

sip:bob@biloxi.com;transport=udp

sip:bob@biloxi.com (通讯协议和端口不同)
sip:bob@biloxi.com:6000;transport=tcp

sip:carol@chicago.com (header 部件不同)
sip:carol@chicago.com?Subject=next%20meeting

sip:bob@phone21.boxesbybob.com (就算是
phone21.boxesbybob.com
sip:bob@192.0.2.4 解析到 192.0.2.4 也不能算相等的。)

注意相等性是不能传递的。
比如 sip:carol@chicago.com 和 sip:carol@chicago.com;security=on 相等
sip:carol@chicago.com 和 sip:carol@chicago.com;security=off 相等
但是：
sip:carol@chicago.com;security=on 和
sip:carol@chicago.com;security=off 不等。

19.1.5 从 URI 中产生请求

对于实现而言，需要能够直接从一个 URI 来构造请求。URI 可以是从名片，网页，或者甚至从某些协议内部得到（比如登记的联系信息等等）。

协议的实现必须包括构造请求的 Request-URI 中的 transport,maddr,ttl,或者 user 参数。如果 URI 包含了 method 参数，那么它的值必须和构造的请求的方法一样。并且 method 参数不能放在 Request-URI 中。不认识的 URI 参数必须放在消息的 Request-URI 中。

实现中应当把 URI 中出现的 header 或者包体部分包含入消息本身，并且当作是请求自己的组成部分。

在实现中，不应当保留那些明显危险的头域字段：From,Call-ID,Cseq,Via 和 Record-Route。

并且实现中，也不应当保留任何请求的 Route 头域值，这样可以避免无知的客户端进行恶意攻击。

实现中也不应当保留那些可能会导致错误登记地址或者误导能力的头域字段，这些包括：Accept,Accept-Encoding,Accept-Language,Allow,Contact(在对话中使用)，Organization,Supported,和 User-Agent。

实现上应当检查每一个请求中所描述的头域的正确性，包括：Content-Disposition, Content-Encoding,Content-Language, Content-Length, Content-Type, Date, Mime-Version, Timestamp。

如果从给定 URI 构造的请求不是一个合法的 SIP 请求，那么这个 URI 就是非法的 URI。实现上禁止处理和传送非法的 SIP 请求。它应当尝试追查为何会有一个非法的 URI。

很多情况都可以得到一个非法的请求。这包括但是不限于，头域的语法错误，非法的 URI 参数合并，或者错误的消息体描述等等。

发送从 URI 构造的请求可能会导致实现上的能力不够。比如：URI 可能指定了尚未实现的通讯协议或者通讯扩展。那个这个具体的实现上来说，应当拒绝发送这些请求，而不是修改这个请求来适应具体实现的处理能力。对于具体实现来说，它不能发送包含它自己不能理解的扩展部分的请求。

比如，从一个包含了未知的或者摆明了不支持的 Request 头域参数或者 method 参数的 URI 中，构造的请求就是不能发送的。

19.1.6 关联 SIP URI 和 tel URL

如果 tel URL(RFC 2806[9])转换成为一个 SIP 或者 SIPS URI,那么 tel URL 的整个电话描述（telephone-subscriber），机器参数，都需要放在 SIP 或者 SIPS URI 的 userinfo 部分。

因此：tel:+358-555-1234567;postd=pp22 会变成：

sip:+358-555-1234567;postd=pp22@foo.com;user=phone

或者

sips:+358-555-1234567;postd=pp22@foo.com;user=phone

而不是

sip:+358-555-1234567@foo.com;postd=pp22;user=phone

或者

sips:+358-555-1234567@foo.com;postd=pp22;user=phone

通常来说，相等的”tel”URL 转换成为 SIP 或者 SIPS URI 以后，不一定能得到相同的 SIP 或者 SIPS URI。因为 SIP 和 SIPS URI 的 userinfo 部分是根据大小写敏感的字串。由大小写不敏感的 tel URL 以及重新排序的 tel URL 参数并不改变 tel URL 的相等性，但是在转换成为 SIP 或者 SIPS URI 之后，却影响了他们的相等性。

例如：

tel:+358-555-1234567;postd=pp22

tel:+358-555-1234567;POSTD=PP22

是等价的，但是

sip:+358-555-1234567;postd=pp22@foo.com;user=phone
sip:+358-555-1234567;POSTD=PP22@foo.com;user=phone
却是不等价的。

类似的：
tel:+358-555-1234567;postd=pp22;isub=1411
tel:+358-555-1234567;isub=1411;postd=pp22
是等价的，但是
sip:+358-555-1234567;postd=pp22;isub=1411@foo.com;user=phone
sip:+358-555-1234567;isub=1411;postd=pp22@foo.com;user=phone
却不等价

为了避免这个问题，在构造放在 SIP 或者 SIPS URI 中的 userinfo 部分的电话描述域的时候，应当转换大小写不敏感的电话描述域为小写，并且除了 isdn-subaddress 和 post-dial,把电话描述的参数按照参数名进行排序， 因为他们需要按顺序出现在参数的第一个。（在下边是除了未来扩展参数意外的全部 tel URL 大小写不敏感的部分）。

根据上边的描述，全部：
tel:+358-555-1234567;postd=pp22
tel:+358-555-1234567;POSTD=PP22

转换成为
sip:+358-555-1234567;postd=pp22@foo.com;user=phone

并且全部：
tel:+358-555-1234567;tsp=a.b;phone-context=5
tel:+358-555-1234567;phone-context=5;tsp=a.b
转换成为：

sip:+358-555-1234567;phone-
context=5;tsp=a.b@foo.com;user=phone

19.2 Option Tags

Option tags 是一个唯一标志，用来指明 SIP 中的新 options（扩展）的。这些 tags 在 Require(20.32 节)，Proxy-Require(20.29 节)，Supported(20.37 节)和 Unsupported(20.40 节)头域中使用。注意这些 options 是以 option-tag=的形式作为这些头域的参数存在的（25 节有关定义符号）。

Option tags 是根据标准的 RFC 扩展定义的。这是和过去的试验有所不同，这是协会为了保证多个厂商之间能够持续互相协作（20.32 节、20.37 节的讨论）。option tags 的 IANA 注册可以保证查找很容易。

19.3 Tags

"tag"参数用于 SIP 消息中的 To 和 From 头域。它作为一个通用的机制的一部分来唯一标志一个对话，这个机制用 Call-ID 和两个从对话参与者的 tag 来标志一个对话。当 UA 在对话外发出一个请求时，它只包含了 From tag,提供了对话 ID 的"一半"。对话根据应答创建完成，这个应答在 To 头域中提供了对话 ID 的另一半。SIP 请求的分支意味着一个单个请求可以创建多个对话。这个也解释了为何需要对话两方的标志；如果没有被叫方的标志，呼叫方不能分辩和消除由单个请求创建的多个对话。

当 UA 产生一个 tag 并且增加进一个请求或者应答的时候，它必须是一个全局唯一的，并且是密码随机数起码是 32 位的随机数。这个要求是为了让 UA 能够在同一个 INVITE 请求中，在给这个 INVITE 的应答中，在 To 头域产生一个不同的 tag，和原始 INVITE 请求在 From 头域中产生的 tag 不同。这是因为 UA 可以邀请自己到一个会话，常见的是在 PSTN 网关的"hairpinning"（发夹）呼叫。类似

的，对不同呼叫的两个 INVITE 也有不同的 From tag,并且给这两个呼叫的两个应答也有不同的 To tag。

在全局唯一要求之外，产生 tag 的算法是实现相关的。Tag 对于容错系统比较有用，在容错系统下，当主服务器故障的时候，对话会在另外一个服务器上进行恢复。UAS 可以产生一个 tag，让备用服务器能够认识到这个请求是在故障服务器上的对话，并且能够决定是否恢复对话和对话相关的状态。

20 头域

头域的语法描述在 7.3 节。本节列出了头域的全部列表，包括了语法注释，含义，和用法。通过本节，我们使用[HX.Y]指当前 HTTP/1.1 的 RFC2616[8]的规范的 X.Y 节。每个头域都有示例给出。

关于与方法和 proxy 处理有关的头域字段在表 2 和表 3 中有处理。

“where”列描述了在头域中能够使用的请求和应答的类型。这列的值是：
R:头域只能在请求中出现；
r:头域只能在应答中出现；
2xx，4xx，等等：一个数字的值区间表示头域能够使用的应答代码。
c：头域是从请求拷贝到应答的。
如果”where”栏目是空白，表示头域可以在所有的请求和应答中出现。

“proxy”列描述了 proxy 在头域上的操作
a：如果头域不存在，proxy 可以增加或者连接头域
m：proxy 可以修改现存的头域值
d：proxy 可以删除头域值
r：proxy 必须能读取这个头域，因此这个头域不能加密。

接下来 6 个栏目与在某一个方法中出现的头域有关：

c：条件；对头域的要求依赖于消息的内容

m：头域是强制要有的。

m*：头域应当被发送，但是客户端/服务端都需要准备接收没有这个头域的消息。

o：头域是可选的。

t：头域应当被发送，但是客户端/服务端都需要准备接收没有这个头域的消息。客户端/服务端都需要准备接收没有这个头域的消息。如果通讯的协议是基于面向流的协议（比如 TCP），那么头域值必须被发送。

*：如果消息体不为空，那么头域值就绪要的。（细节请参见 20.14,20.15 和 7.4 节）

-:这个头域是不适用的。

“Optional”意味着这个元素可以在请求或者应答中包含这个头域，并且 UA 可以忽略在请求或者应答中存在的这个头域（这条规则有一个例外，就是 Require 头域，在 20.32 节有描述）。”mandatory”（强制）头域是必须在请求中存在的头域，并且也必须是 UAS 接收到一个请求时能够理解的头域。一个强制头域必须也在应答中出现，并且 UAC 也能处理这个头域。”Not applicable”（不适用）意味着头域不能在请求中出现。如果一个 UAC 错误的把这个头域放在请求中，在 UAS 收到的时候必须被忽略。同样的，如果应答中的”不适用”的头域，也就是说 UAS 不能在应答中放置的头域，如果出现了，那么 UAC 也必须在应答中忽略掉这个头域。

一个 UA 必须忽略他们所不能处理的扩展的头参数。

本规范也定义了常用的头域名的缩写，用于缩小消息的大小。

在 Contact，From，To 头域中都包含一个 URI。如果这个 URI 包含一个逗号，问号或者分毫，那么这个 URI 必须使用尖括号括起来（<和>）。所有的 URI 参数都必须在这些括号内。如果 URI 并非用尖括号括起来的，那么用分号分开的参数将被视同与 header 参数而不是 URI 参数。

20.1 Accept

Accept 头域的语法定义遵从[H14.1]。除了如果没有 Accept 头域，服务器应当认为 Accept 缺省值是 application/sdp 以外，语义也是和 HTTP/1.1 类似的语义。

空的 Accept 头域意味着不接受任何格式。

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Accept	R		-	o	-	o	m*	o
Accept	2xx		-	-	-	o	m*	o
Accept	415		-	c	-	c	c	c
Accept-Encoding	R		-	o	-	o	o	o
Accept-Encoding	2xx		-	-	-	o	m*	o
Accept-Encoding	415		-	c	-	c	c	c
Accept-Language	R		-	o	-	o	o	o
Accept-Language	2xx		-	-	-	o	m*	o
Accept-Language	415		-	c	-	c	c	c
Alert-Info	R	ar	-	-	-	o	-	-
Alter-Info	180	ar	-	-	-	o	-	-
Allow	R		-	o	-	o	o	o
Allow	2xx		-	o	-	m*	m*	o
Allow	r		-	o	-	o	o	o
Allow	405		-	m	-	m	m	m

Authentication-Info	2xx		-	o	-	o	o	o
Authorization	R		o	o	o	o	o	o
Call-ID	c	r	m	m	m	m	m	m
Call-Info		ar	-	-	-	o	o	o
Contact	R		o	-	-	m	o	o
Contact	1xx		-	-	-	o	-	-
Contact	2xx		-	-	-	m	o	o
Contact	3xx	d	-	o	-	o	o	o
Contact	485		-	o	-	o	o	o
Content-Disposition			o	o	-	o	o	o
Content-Encoding			o	o	-	o	o	o
Content-Language			o	o	-	o	o	o
Content-Length		ar	t	t	t	t	t	t
Content-Type			*	*	-	*	*	*
Cseq	c	r	m	m	m	m	m	m
Date		a	o	o	o	o	o	o
Error-Info	300-699	a	-	o	o	o	o	o
Expires			-	-	-	o	-	o
From	c	r	m	m	m	m	m	m
In-Reply-To	R		-	-	-	o	-	-
Max-Forwards	R	amr	m	m	m	m	m	m
Min-Expires	423		-	-	-	-	-	m
MIME-Version			o	o	-	o	o	o
Organization		ar	-	-	-	o	o	o

表 2： 头域概览，A-O

Header field	where	proxy	ACK	BYE	CAN	INV	OPT	REG
Priority	R	ar	-	-	-	o	-	-
Proxy-Authenticate	407	ar	-	m	-	m	m	m
Proxy-Authenticate	401	ar	-	o	o	o	o	o
Proxy-Authorization	R	dr	o	o	-	o	o	o
Proxy-Require	R	ar	-	o	-	o	o	o
Record-Route	R	ar	o	o	o	o	o	o
Record-Route	2xx,18x	mr	-	o	o	o	o	-
Reply-To			-	-	-	o	-	-
Require		ar	-	c	-	c	c	c
Retry-After	404, 413, 480, 486		-	o	o	o	o	o
Retry-After	500,503 600,603		-	o	o	o	o	o
Route	R	adr	c	c	c	c	c	c
Server	r		-	o	o	o	o	o
Subject	R		-	-	-	o	-	-
Supported	R		-	o	o	m*	o	o
Supported	2xx		-	o	o	m*	m*	o
Timestamp			o	o	o	o	o	o
To	c(1)	r	m	m	m	m	m	m
Unsupported	420		-	m	-	m	m	m

User-Agent			o	o	o	o	o	o
Via	R	amr	m	m	m	m	m	m
Via	rc	dr	m	m	m	m	m	m
Warning	r		-	o	o	o	o	o
WWW-Authenticate	401	ar	-	m	-	m	m	m
WWW-Authenticate	407	ar	-	o	-	o	o	o

表 3：头域概览，P-Z （1）和可能的附加 tag 一起拷贝。

例子：

Accept: application/sdp;level=1,application/x-private,text/html

20.2 Accept-Encoding

Accept-Encoding 头域类似 Accept，但是限定了接收应答中的内容的编码[H3.5]。参见[H14.3]。在 SIP 中的语义和在[H14.3]中的定义是一致的。

一个空的 Accept-Encoding 头域是允许的。他等同于 Accept-Encoding:identity,这就是说，只有 identity 编码，也就是说没有编码的情况，是允许的。

如果没有 Accept-Encoding 头域存在，那么服务端应当使用缺省值：identity。

这个和 HTTP 的定义略有不同，HTTP 指出如果本头域不存在，那么任何编码形式都可以使用，只是推荐 identity 编码而已。

例如：

Accept-Encoding:gzip

20.3 Accept-Language

Accept-Language 头域用来在请求中指定首选的的语言的，这个首选的语言是在应答中的消息体中的的原因分析，会话描述，或者状态报告的。如果没有 Accept-Language 存在，那么服务端应当假设所有的语言客户端都可以接受。

Accept-Language 头域遵从[H14.4]节定义的语法。对于 SIP 来说，也同样支持对语言通过”q”参数来进行排序。

例如：

Accept-Language: da, en-gb; q= 0.8, en;q=0.7

20.4 Alert-Info

当 INVITE 请求有一个 Alert-Info 头域的时候，Alert-Info 头域就包含的是给 UAS 的一个额外的信息。当在 180（Ringing）应答中出现的时候，Alter-Info 头域给出了 UAC 一个额外的回铃信息。这个头域的一个典型用法就是让 proxy 增加这个头域用来体哦你嘎一个与众不同的振铃效果。

Alter-Info 头域可能会带来潜在的安全隐患。这个隐患以及相应的处理在 20.9 节有讲述，这个隐患和 Call-Info 头域的隐患是相同的。

另外，用户应当可以有选择的屏蔽这个特定。

这个可以保护用户不因为使用了未受信任节点发送过来的这个头域而导致的破坏。

例如：

Alter-Info: <http://www.example.com/sounds/moo.wav>

20.5 Allow

Allow 头域列出了 UA 支持的方法列表。

如果要提供 UA 头域，那么所有只要是 UA 支持的方法，包括 ACK 和 CANCEL 都必须列在这个 Allow 头域中。如果没有 Allow 头域出现，一定不能以为 UA 什么方法都不支持。应当解释成为发送这个消息的 UA 并没有告诉大家它支持什么方法。

在应答中提供 Allow 头域比在 OPTIONS 请求/应答中会减小所需要的消息数量。

例如：

Allow: INVITE,ACK,OPTIONS,CANCEL,BYE

20.6 Authentication-Info

Authentication-Info 头域提供了和 HTTP 类别相同的认证方法。UAS 可以在给一个顺利通过认证的请求的 2xx 应答中包含这个头域，并且是使用基于 Authorization 头域的分类。

这个头域的语法和语义遵循 RFC2617[17]的规范。

例如：

Authentication-Info: nextnonce=”47364c23432d2e131a5fb210812c”

20.7 Authorization

Authorization 头域包含了了 UA 进行认证的信任书。22.2 节概述了对 Authorization 头域的用法，22.4 节讲述了和 HTTP 认证一起使用的时候的语法和语义。

这个头域，和 Proxy-Authorization,并不遵循通常的多头域值的规则。虽然它不是由逗号分割的列表，这个头域名可以出现多次，并且不能应用 7.3 节的规则合并成为单个头域。

在下边的例子中，在分类参数两边没有引号括起来。

Authorization:Digest username=”Alice”, realm=”atlanta.com”,
nonce = ”84a4cc6f3082121f32b42a2187831a94”,
response=”7587245234b3434cc3412213e5f113a5432”

20.8 Call-ID

Call-ID 头域用来唯一区别一个特定的邀请或者一个特定客户端的所有注册项。单个多媒体会议可以分解成为多个不同 Call-ID 的呼叫，例如，当一个用户数次邀请单个个体加入同一个会议的时候。Call-ID 是大小写敏感的并且是字节/字节比较的。

Call-ID 头域的简写就是 i

例子：

Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@biloxi.com
i:f81d4fae-7dec-11d0-a765-00a0c91e6bf6@192.0.2.4

20.9 Call-Info

Call-Info 头域提供了对呼叫方或者被叫方的附加信息，如果出现在请求中则是呼叫方的信息，如果出现在应答中则是被叫方的。”purpose”参数中存放了效果图 URI。”icon”参数包含了一个呼叫方或者被叫方的图标。”info”参数描述了简要的呼叫方或者被叫方的信息，例如，通过放置一个网页进行介绍等。”card”参数提

供了一个名片，比如，基于 vCard[36]或者 LDIF[37]格式。如果附加新的标记，那么可以通过 27 节描述的步骤通过在 IANA 注册来附加。

对 Call-Info 的使用可能会带来一些安全隐患。如果一个被叫方接到一个恶意呼叫方提供的 URI，被叫方可能会由显示一个不合适的内容，或者危险的或者非法的内容，等等。因此，我们建议 UA 只显示那些它能够检验并且信任发送方身份的 Call-Info 头域中的内容。这个对于对方 UA 来说不需要。proxy 可以在请求中加入这个头域。

例如：

Call-Info: http://www.example.com/alice/photo.jpg;purpose=icon,
http://www.example.com/alice/;purpose=info

20.10 Contact

Contact 头域提供了一个 URI，这个 URI 的含义取决于是在请求还是在应答中。

Contact 头域包含了一个显示的名字，一个包含参数的 URI，还有 header 参数组成。

本文档定义了一个 Contact 参数”q”和”expires”。这些参数只有当 Contact 头域在 REGISTER 的请求或者应答，或者 3xx 的应答中才有效。在其他规范中可能会定义一个附加的参数。当头域值包含一个显示的名字，那么带参数的 URI 应当用”<”和”>”括起来。如果没有”<”,”>”括起来，所有 URI 后边的参数都将视为 header 参数，而不是 URI 参数。显示姓名可以是符号，或者引号引起来的字符串（如果很长的话）。

即使”display-name”是空的，如果”addr-spec”包含一个逗号或者分号，或者？的话，也必须使用”name-addr”的格式。这在 display-name 和”<”之间可以有也可以没有 LWS(线性空白)

这些关于显示名字，URI 和 URI 的参数，header 参数的规则同样对 To 和 From 头域适用。Contact 头域的的角色很像 HTTP 中的 Location 头域的角色。但是 HTTP 头域只允许 1 个地址，没有其他说明。由于 URI 中可以包含逗号和分号，所以他们在 header 或者参数分隔符上是错误的。

Contact 头域的缩写是 m(“moved”)。
例子：

Contact： “Mr.Watson” <sip:watson@worcester.bell-telephone.com>;q=0.7;
expires=3600,
“Mr. Watson” mailto:watson@bell-telephone.com ;q=0.1

m: <sips:bob@192.0.2.4>;expires=60

20.11 Content-Disposition

Content-Disposition 头域描述了消息体，或者消息的多个部分，或者消息体的一个部分应被 UAC 或者 UAS 怎样解释。这个 SIP 头域扩展了 MIME Content-Type(RFC 2183[18])。

SIP 定义了 Content-Disposition 几个新的”disposition-types”。如果取值”session”意味着消息体位呼叫（calls）或者早期（pre-call）媒体，描述了一个会话。取值”render”表示了消息体可是被显示或者展示给用户。注意”render”比”inline”更适合避免 MIME 消息体作为一个大的消息的一部分做展示（由于 SIP

消息的 MIME 消息体经常不被展示给用户）。出于向后兼容的考虑，如果 Content-Disposition 头域不存在，服务器应当假设 Content-Type 为 application/sdp 的部属方式是”session”，为其他方式的时候是”render”。

部属方式“icon”表示消息体部分包含了一个用于表示呼叫者或者被叫者的 icon 图像，当 UA 收到这个消息，就可以展示一下，或者在对话过程中一致展示。”alert”意味着消息体部分包含了信息，比如是一段声音，应当由 UA 展示给用户提示用户这个请求，通常是初始化对话的请求；这个 altering 消息体可以是一个在 180Ringing 临时应答发出后的一个铃声。

所有需要展示给客户的具有”disposition-type”的 MIME 消息体，都应当只在这个消息有适当的安全认证的时候展示。

处理参数，handling-param,描述了 UAS 在接收到这个内容类型或者部属类型是它所不支持的消息体的时候，应当如何操作。这个参数定了了”optional”和”required”两个值。如果处理参数没有，那么这个处理参数缺省值就是”required”。处理参数在 RFC3204[19]中定义和描述的。

如果这个头域不存在，那么 MIME 类型决定了缺省的内容部属。如果没有 MIME 类型，那么缺省值就是”render”
例如：

Content-Disposition: session

20.12 Content-Encoding

Content-Encoding 头域是对”media-type”(媒体类型)的一个修正。当存在这个头域的时候，它的值就是对包体内容编码的附加说明，并且因此必须根据本字段应用正确的解码机制，这样才能得到正确的 Content-Type 头域指出的媒体类型的

解码。Content-Encoding 首要应用于在不丢失媒体类型标记的情况下对消息体进行压缩处理。

如果包体应用了多个编码，那么包体编码必须按顺序在这个字段中进行列出。

所有的 Content-Encoding 的值都是大小写不敏感的。IANA 是这个编码方式的注册机构。参见[H3.5]获得 Content-coding 的语法定义。

客户端可以在请求中进行包体的内容编码。服务端也可以在应答中进行内容编码。服务端必须只能应用客户端在请求中的 Accept-Encoding 头域中列出的编码类型。

Content-Encoding 简写是 e。
例如：
Content-Encoding:gzip
e: tar

20.13 Content-Language

参见[H14.12].例如：
Content-Language: fr

20.14 Content-Length

Content-Length 头域标志了消息体的大小，给消息的接受者，以 10 进制表示的数字。应用程序应当使用这个字段标志的大小来传送消息体，而不关心消息体的媒体类型是什么。如果是基于流的通讯协议（比如 TCP），那么本头域必须提供。

消息的大小并不包含 CRLF 分开的头域和包体。任何大于或者等于 0 的 Content-Length 都是合法的长度。如果消息中不包含包体，那么 Content-Length 必须设置成为 0

对 Content-Length 的忽略能够简化创建一个类似 cgi 一样动态生成应答的脚本。（？？？）

这个头域的简写是 l

例如：

Content－Length：349

l：173

20.15 Content-Type

Content-Type 头域标志了发给对方的消息体的媒体类型。”media-type”是在[H3.7]中定义的。如果消息体不为空，那么 Content-Type 头域就必须存在。如果消息体是空的，并且笨头域存在，那么就表示了特定类型的媒体的包体是 0 长度（比如空的音频文件）。

本头域的简写是 c

例如：

Content-Type: application/sdp

c: text/html;charset=ISO-8859-4

20.16 Cseq

请求中的 Cseq 头域包含了一个单个的数字序列号和请求的方法。这个序列号必须是表示成为一个 32 位的无符号整数。在 Cseq 的请求方法部分是大小写敏感的。

Cseq 头域是为了在会话中对事务进行排序的，提供事务的唯一标志，并且区分请求和请求的重发。如果序列号相等，并且请求的方法相等，那么两个 Cseq 头域就是相等的。

例如：

Cseq:4711 INVITE

20.17 Date

Date 头域包含了日期和时间。和 HTTP/1.1 不同，SIP 只支持最近的 RFC1123[20]格式的日期。如同在[H3.3]中，SIP 限制了在 SIP－date 中的时区是”GMT”,但是在 RFC1123 中支持任意的市区。RFC1123 的日期是大小写敏感的。Date 头域反应的时间是请求或者应答被发送的那一刻的时间。

Date 头域可以用来简化没有后备电池的终端系统，让他们能够获得当前的时间。但是由于是 GMT 格式的，所以，它要求客户端知道和 GMT 的时差。

例如：

Date：Sate,13 Nov 2010 23:29:00 GMT

20.18 Error-Info

Error-Info 头域提供了对有错误应答码的应答的附加信息。

SIP UAC 具有从弹出的窗口 PC 界面，到只有声音的电话或者网关过来的终端界面。与其强制服务器产生一个错误来选择是发送一个带有详细原因说明的错误代码应答，还是播放一段声音，不如使用 Error-Info 头域把两个都发送。让 UAC 来决定用什么来展示给呼叫方。

UAC 可以把在 Error-Info 中的一个 SIP 或者 SIPS URI 当作是转发的一个 Contact 地址，并且据此产生一个新的 INVITE，这样可以建立预先录制的声明会话。如果是非 SIP URI，那么可以展示给用户。

例如：

SIP/2.0 404 The Number you have dialed is not in service
Error-Info: <sip:not-in-service-recording@atlanta.com>

20.19 Expires

Expires 头域给定了消息（或者内容）过期的相关时间。这个字段的精确定义是方法相关的。对于一个 INVITE 的超时时间并不影响这个 INVITE 请求建立的实际的会话。不过，会话描述协议可以描述在一个会话上的的时间限制。

这个头域的值是一个以秒计数的整数，从 0 到(2**32)-1，从收到请求开始计数。

例如：

Expires:5

20.20 From

From 头域表示了请求的来源地。这个可能和对话的来源的不同，被叫方到呼叫方的请求会在 From 头域使用被叫方的地址。

选项”display-name”是展示给界面的。如果客户标志停留在隐藏状态，那么系统应当使用”Anonymous”作为显示名字。即使是”displayname”是空的，如

果”addr-spec” 包含一个逗号，？，或者分毫，那么就必须使用”name-addr”格式。相关的格式在 7.3.1 节描述。

如果 From 的 URI 相等，并且参数也相等，那么这两个头域就是相等的。如果扩展参数在一个头域中存在，但是在另外一个头域中不存在，那么当这两个头域做比较的时候，这个参数将被忽略。这意味着显示名字的存在与否不影响比较的结果。

参见 20.10 处理显示名字，URI 和 URI 参数，以及头域参数的规则。

From 头域的简写是 f

例子：

From: “A. G. Bell” <sip:agb@bell-telephone.com> ; tag=a48s

From: sip:+12125551212@server.phone2net.com;tag=887s

f: Anonymous <sip:c8oqz84zk7z@privacy.org>;tag=hyh8

20.21 In-Reply-To

In-Reply-To 头域列举了本次呼叫相关的或者返回的 Call-ID。这些 Call-ID 可以备客户端 cache 起来，这样可以在这个头域中返回。

这允许自动呼叫奋发系统来路由这些返回的呼叫到第一个呼叫的原始请求地点。这也允许被叫方过滤呼叫，这样只有在呼叫中原始请求建立的呼叫才会被接受。这个字段不是对请求验证的一个替代。

例如：

In-Reply-To: 70710@saturn.bell-tel.com,17320@saturn.bell-tel.com

20.22 Max-Forwards

Max-Forwards 头域必须在任何一个 SIP 请求中使用，来限制中间转发请求到下一个节点的 proxy 或者 gateway 的个数。这个在客户端 trace 一个请求，如果路由失败或者在中间出现循环的时候特别有用。

Max-Forwards 是一个 0－255 的整数，表明了在这个请求消息中允许被转发的剩余次数。每当服务器转发这个请求一次，这个数字就减一。建议的初始值是 70。当不能确定有无循环路由的时候，必须在头域中增加本头域。比如，一个 B2BUA 应当增加这个头域。

例如：

Max-Forwards:6

20.23 Min-Expires

Min-Expires 头域包含了一个服务器所支持的内部状态（soft－state）的最小的刷新时间间隔。这个包括被登记服务器所登记的 Contact 头域。这个头域包含了一个以秒计数的整数，从 0 到(2**32)-1。在 423（Interval Too Brief）应答中，本头域的用法在 10.28,10.3,和 21.4.17 中有描述。

例如：

Min-Expires:60

20.24 MIME-Version

参见[H19.4.1]

例如：

MIME-Version: 1.0

20.25 Organization

Organization 头域包含了发出请求或者应答的 SIP 节点所属的组织名字。这个字段可以用来让客户端软件过滤呼叫。

例如：

Organization: Boxes by Bob

20.26 Priority

Priority 头域标志了客户端评价的请求紧急程度。Priority 头域描述了 SIP 应当处理人工或者 UA 发过来的请求的优先级。举例来说，这可能是决定呼叫转发和处理的优先要素。对于判定优先级来说，如果消息没有包含 Priority 字段，那么处理的时候应当当作”normal”优先级处理。Priority 头域不影响通讯资源的优先顺序，比如路由上的包转发的优先级或者访问 PSTN 网关的优先级。本头域有”non-urgent”,”normal”,”urgent”,和”emergency”取值，另外的取值可以在别处定义。我们强烈建议”emergency”只用于影响到生命、身体、或者财产危急时候才使用。其他情况下， 本头域没有额外的语义。在 RFC2076[38]中，定义了”emergency”。

例如：

Subject: A tornado is heading our way!
Priority: emergency。

或者

Subject: Weekend plans
Priority: non-urgent.

20.27 Proxy-Authenticate

Proxy-Authenticate 头域用来进行认证使用的。这个头域的用法在[H14.33]中定义。参见 22.3 节关于本字段的细节讨论。

例如：

Proxy-Authenticate: Digest realm=”atlanta.com”,
domain=”sip:ss1.carrier.com”,qop=”auth”,
nonce=”f84f1cec41e6cbe5aea9c8e88d359”,
opaque=””,stale=FALSE,algorithm=MD5

20.28 Proxy-Authorization

Proxy-Authorization 头域允许客户端向一个要求认证的 proxy 证明自己（或者证明它的使用者）的身份。一个 Proxy-Authorization 头域包含了与 UA 认证信息相关的信任书，这个信任书是给 proxy 和/或者本请求相关的域的。

参见 22.3 节关于这个头域的定义。

本头域，连通 Authorization 头域，并不遵循常用的多头域名（多个相同头域名的合并）的规则。虽然不是用逗号分割的列表，这个头域名可以出现多次，并且不能用 7.3.1 描述的通常规则合并成为一个头域。

例如：

Proxy-Authorization: Digest username=”Alice”,realm=”atlanta.com”,
nonce=”c60f3082ee1212b402a21831ae”,
response=”245f23415f11432b3434341c022”

20.29 Proxy-Require

Proxy-Require 头域用来表示请求中一定要求 proxy 支持的相关的特性。参见 20.32 关于这个头域的使用。

例子：

Proxy-Require:foo

20.30 Record-Route

Record-Route 头域是 proxy 在请求中增加的，用来强制会话中的后续请求经过本 proxy 的。本头域的用法在 16.12.1 节有描述。

例子：

Record-Route: <sip:server10.biloxi.com;lr>,
<sip:bigbox3.site3.atlanta.com;lr>

20.31 Reply-To

Reply-To 头域包含了逻辑上返回目的地 URI，这个可以和 From 头域不同。比如，URI 可以用来返回未接电话或者未建立的会话。如果用户希望保留匿名，那么这个头域应当从请求中去除或者改变，这样可以避免透露个人隐私信息。

即使”display-name”是空的如果”addr-spec”包含了逗号、问号、或者分号，那么就需要使用”name-addr”的格式来填写。这个语法在 7.3.1 中定义。

例如：

Replay-To: Bob <sip:bob@biloxi.com>

20.32 Require

Require 头域用于 UAC 告诉 UAS 关于要求 UAS 支持那些特性。虽然这是一个可选的头域，但是如果 Require 头域存在，那就一定不能掠过不处理。

头域包含一个 option tag 的列表，这个列表在 19.2 节中描述。每一个 option tag 定了一个要处理请求要求 UAS 必须支持的 SIP 扩展。通常，这用于定义一个需要支持的扩展头域的集合。复核本规范的 UAC 应当值包含规范的 RFC 扩展。
例如：

Require：100rel

20.33 Retry-After

Retry-After 头域可以用于 500（Server Internal Error）或者 503（Service Unavailable）应答，用来标志大约本服务还会处于不可用状态多久。在 404(Not Found),413(Request Entity Too Large), 480(Temporarily Unavailable),486(Busy Here), 600 (Busy), 或者 603(Decline)应答中用于标志何时被叫方会恢复正常。这个字段的值是一个秒为单位的正整数（十进制），从应答生成开始的一个正整数。

对于回叫的时间，可以有一个附加的说明。”duration”参数标志了被叫方变成正常状态的时间长度。如果没有定义，那么服务可以被看作是永远有效。

例如：

Retry-After: 18000;duration=3600

Retry-After:120 (I’m in a meeting)

20.34 Route

Route 头域用于强制一个请求经过一个 proxy 路由列表。Route 头域的使用在 16.12.1 节定义：

例如：

Route: <sip:bigbox3.site3.atlanta.com;lr>,
<sip:server10.biloxi.com;lr>

20.35 Server

Server 头域包含了关于 UAS 处理请求所使用的软件信息。

服务器的特定软件版本可能会使服务器由于特定软件安全漏洞导致服务器收到攻击。实现上应当使得 Server 头域是一个可以配置的选项。

例如：

Server：HomeServer v2

20.36 Subject

Subject 头域提供了呼叫的一个概览，允许呼叫不用分析会话描述就可以大致过滤。会话描述并不需要和 INVITE 邀请使用相同的主题标志。

Subject 的缩写是 s

例如：

Subject: Need more boxes
s: Tech Support

20.37 Supported

Supported 头域列举了 UAC 或者 UAS 支持的扩展。

Supported 头域包含了一个 option tag 的列表，在 19.2 节描述的 option tag，他们是这个 UAS 或者 UAC 所支持的。遵循本规范的 UA 必须只包含遵循标准 RFC 扩展的 option tag。如果本字段是空的，意味着不支持任何扩展。

Supported 头域的缩写是 k

例如：

Supported: 100rel

20.38 Timestamp

Timestamp 头域描述了当 UAC 发送请求到 UAS 的时间戳。

参见 8.2.6 节关于如何给请求产生一个包含这个头域的应答。虽然没有定义本字段的标准行为，我们允许对扩展应用或者 SIP 应用获得 RTT 预计时间。

例如：

Timestamp:54

20.39 To

To 头域定义了逻辑上请求的接收者。选项”display-name”意味着展示给客户的界面。”tag”参数提供了对话识别机制。

参见 19.3 节关于”tag”参数的些界描述。

对于 To 头域的比较是和对 From 头域的比较相同的。参见 20.10 节的比较规则来比较 display name,URI 和 URI 参数，以及头域的参数。

To 头域的缩写是 t。

下边是一个 To 头域的例子：

To: The Operator <sip:operator@cs.columbia.edu>;tag=287447

t: sip:+12125551212@server.phone2net.com

20.40 Unsupported

Unsupported 头域列出了不被 UAS 支持的特性列表。参见 20.32。

例如：

Unsupported:foo

20.41 User-Agent

User-Agent 头域包含了发起请求的 UAC 信息。本头域的语义在[H14.43]定义。

UA 所使用的版本号情况可能会导致由于这个版本的安全漏洞二遭受攻击。所以在实现上应当使得 User-Agent 头域是可以配置的。

例如：

User-Agent:Softphone Beta1.5

20.42 Via

Via 头域是用来描述请求当前经历的路径的，并且标志了应答所应当经过的路径。Via 头域的 branch ID 参数提供了事务的标志，并且用于 proxy 来检查循环路由。

Via 头域包含了用于发送消息的通讯协议，客户端主机名或者网络地址，可能还有接收应答所用的端口号码。Via 头域还可以包含参数”maddr”,”ttl”,”received”

和”branch”,这些定义在其他节中描述。对于遵循本规范的实现，这个 branch 参数的值必须用 magic cookie”z9hG4bK”打头（8.1.1.7 节）。

这里定义的通讯协议是”UDP”,”TCP”,”TLS”,和”SCTP”，”TLS”意思是基于 TCP 的 TLS。当请求发送到一个 SIPS URI 上时，协议依旧标记着时”SIP”,但是通讯协议是 TLS。

Via: SIP/2.0/UDP erlang.bell-telephone.com:5060;branch=z9hG4bK87asdks7
Via: SIP/2.0/UDP 192.0.2.1:5060 ;received=192.0.2.207 ;branch=z9hG4bK77asjd

Via 头域的缩写是 v

在这个例子中，从多源(multi-homed)主机的消息有两个地址，192.0.2.1 和 192.0.2.207。发送者猜错了发送的网络界面（以为是在 192.0.2.1 上发送的）。Erlang.belltelephone.com 发现了这个不匹配，并且给这个节点的 Via 增加了一个参数，包含了实际包接收到的地址。

在 SIP URI 语法下，并不要求填写主机名或者网络地址和端口号。特别是，允许在”:”或者”/”两遍的 LWS（线形空白）。例如：

Via: SIP / 2.0 / UDP first.example.com: 4000;ttl=16 ;maddr=224.2.0.1 ;branch=z9hG4bKa7c6a8dlze.1

即使本规范要求所有的请求中都包含 branch 参数，本头域的 BNF 描述中，branch 参数是可选的。这就和 RFC2543 元素可以进行互操作，因为 RFC2543 没有添加 branch 参数。

如果他们的发送协议和 sent-by 域相等，都有相同的参数集合，并且参数都相等，那么两个 Via 头域就是相同的。

20.43 警告

Warning 头域用来给应答的状态添加附加说明使用的。Warning 头域值是在应答中包含的，并且包括了一个 3 位的警告代码，主机名，和警告正文。

"warn-text"应当是一个自然语言，给个人用户接收应答时候来响应的。这可以通过现有的各种信息来决定这个 warn-text,比如用户的位置，Accept-Language 域，或者应答重的 Content-Language 等等。缺省语言是 idefault[21]。

下边列出了当前定义的"warn-code",并且有英文描述的推荐的 warn-text。这些井盖描述了会话描述中的各种可能的失败情况。第一个 warn－code 的数字是"3"表示这是一个 SIP 规范的警告信息。警告信息 300 到 329 是保留用于标志在会话描述中的保留字错误的，330 到 339 是会话描述中基本网络服务相关警告，370 到 379 是关于会话描述重的 QoS 参数数量相关的警告，390 到 399 是上边未列除的杂项警告信息。

300 Incompatible network protocol:（不兼容的网络协议），One or more network protocols contained in the session description are not available.（在会话描述中的一个或者多个网络协议不适用）

301 Incompatible network address formats(不兼容的网络地址格式)：One or more network address formats contained in the session description are not available. （会话描述中的一个或者多个网络地址格式不合法）

302 Incompatible transport portocol(不兼容的通讯协议)：One or more transport protocols described in the session description are not available.（会话描述中的一个或者多个通讯协议不存在）。

303：Incompatible bandwidth units（不兼容的带宽单位）: One or more bandwidth measurement units contained in the session description were not understood.（会话描述中的一个或者多个带宽单位不支持）。

304 Media type not available（媒体类型不存在）: One or more media types contained in the session description are not available.（会话描述中的一个或者多个媒体类型不存在）。

305 Incompatible media format（媒体格式不兼容）: One or more media formats contained in the session description are not available.（会话描述中的一个或者多个媒体格式不兼容）。

306 Attribute not understood（媒体属性不支持）: One or more of the media attributes in the session description are not supported.（会话描述中的一个或者多个媒体属性不支持）。

307 Session description parameter not understood（会话描述参数不支持）: A parameter other than those listed above was not understood.（列出的会话描述参数不支持）。

330 Multicast not available（多点传输不允许）: The site where the user is located does not support multicast.（用户定位的这个服务器不支持多点传送）。

331 Unicast not available（Unicast 不支持）: The site where the user is located does not support unicast communication (usually due to the presence of a firewall)。（用户定位的节点不支持 unicast 通讯（通常由于在防火墙之后））。

370 Insufficient bandwidth(带宽不足): The bandwidth specified in the session description or defined by the media exceeds that known to be available.（会话描述的带宽要求或者媒体要求的带宽超过限制）。

399 Miscellaneous warning（杂项警告）: The warning text can include arbitrary information to be presented to a human user or logged. A system receiving this warning MUST NOT take any automated action.（这个警告信息可以包含给用户的任意信息或者做日志记录。接收到这个警告的系统禁止做任何自动操作）。

1xx 和 2xx 消息是 HTTP/1.1 使用的。

附加的”warn-code”是 IANA 定义的，在 27.2 节有附加说明。

例如：

Warning: 307 isi.edu "Session parameter ’foo’ not understood"

Warning: 301 isi.edu "Incompatible network address type ’E.164’"

20.44 WWW-Authenticate

WWW-Authenticate 头域包含了认证信息，参见 22.2 节有关的详细说明。

例如：

WWW-Authenticate:Digest realm=”atlanta.com”,
domain=”sip:boxesbybob.com”,qop=”auth”,
nonce="f84f1cec41e6cbe5aea9c8e88d359",
opaque="", stale=FALSE, algorithm=MD5

21 应答代码

应答码是包含了，并且扩展了 HTTP/1.1 应答码。并不是所有的 HTTP/1.1 应答码都适当应用，只有在这里指出的是适当的。其他 HTTP/1.1 应答码不应当使用。并且，SIP 也定义了新的应答码系列，6xx。

21.1 临时应答 1xx

临时应答，也就是消息性质的应答，标志了对方服务器正在处理请求，并且还没有决定最后的应答。如果服务器处理请求需要花 200ms 以上才能产生终结应答的时候，它应当发送一个 1xx 应答。

注意 1xx 应答并不是可靠传输的。他们不会导致客户端传送一个 ACK 应答。临时性质的（1xx）应答可以包含消息体，包含会话描述。

21.1.1 100 Trying

这个应答表示下一个节点的服务器已经接收到了这个请求并且还没有执行这个请求的特定动作（比如，正在打开数据库的时候）。这个应答，就像其他临时应答一样，种植了 UAC 重新传送 INVITE 请求。100(Trying)应答和其他临时应答不同的是，在这里，它永远不会被有状态 proxy 转发到上行流中。

21.1.2 180 Ringing

UA 收到 INVITE 请求并且试图提示给用户。这个应答应当出世化一个本地回铃。

21.1.3 818 Call is Being Forwarded(呼叫被转发)

服务器可以用这个应答代码来表示呼叫正在转发到另一个目的地集合。

21.1.4 182 Queued

当呼叫的对方暂时不能接收呼叫的时候，并且服务器决定将呼叫排队等候，而不是拒绝呼叫的时候，那么就应当发出这个应答。当被叫方一旦恢复接收呼叫，他会返回合适的终结应答。对于这个呼叫状态，可以有一个表示原因的短语，比如：”5 calls queued;expected waiting time is 15minutes”。服务器可以给出好几个 182（Queued）应答告诉呼叫方排队的情况（比如排队靠前了等等）。

21.1.5 183 会话进度

183（Session Progress）应答用于提示建立对话的进度信息。Reason-Phrase（表达原因的句子）、头域或者消息体可以用于提示呼叫进度的更消息的信息。

21.2 成功信息 2xx

这个应答表示请求是成功的。

21.2.1 200 OK

请求已经处理成功。这个信息取决于不同方法的请求的应答。

21.3 转发请求 3XX

3xx 系列的应答是用于提示用户的新位置信息的，或者为了满足呼叫而转发的额外服务地点。

21.3.1 300 Multiple Choices

请求的地址有多个选择，每个选择都有自己的地址，用户或者（UA）可以选择合适的通讯终端，并且转发这个请求到这个地址。

应答可以包含一个具有每一个地点的在 Accept 请求头域中允许的资源特性，这样用户或者 UA 可以选择一个最合适的地址来转发请求。没有未这个应答的消息体定义 MIME 类型。

这些地址选择也应当在 Contact 头域中列出（20.10 节）。不同于 HTTP，SIP 应答可以包含多个 Contact 头域或者一个 Contact 头域中具有一个地址列表。UA 可以使用 Contact 头域来自动转发或者要求用户确认转发。不过，本规范没有定义自动转发的标准。

如果被叫方可以在多个地址被找到，并且服务器不能或者不愿意转发请求的时候，可以使用这个应答来给呼叫方。

21.3.2 301 Moved Permanently

当不能在 Request-URI 指定的地址找到用户的时候，请求的客户端应当使用 Contact 头域(20.10)所指出的新的地址重新尝试。请求者应当用这个新的值来更新本地的目录，地址本，和用户地址 cache，并且在后续请求中，发送到这个/这些列出的地址。

21.3.3 302 Moved Temporarily

请求方应当把请求重新发到这个 Contact 头域所指出的新地址(20.10)。新请求的 Request-URI 应当用这个应答的 Contact 头域所指出的值。

在应答中的 Expires(20.19 节)或者 Contact 头域的 expires 参数定义了这个 Contact URI 的生存周期。UA 或者 proxy 在这个生存周期内 cache 这个 URI。如果没有严格的有效时见，那么这个地址仅仅本次有效，并且不能在以后的事务中保存。

如果 cache 的 Contact 头域的值失败了，那么被转发请求的 Request-URI 应当再次尝试一次。临时 URI 可以比超时时间更快的失效，并且可以有一个新的临时 URI。

21.3.4 305 Use Proxy

请求的资源必须通过 Contact 头域中指出的 proxy 来访问。Contact 头域指定了一个 proxy 的 URI。接收到这个应答的对象应当通过这个 proxy 重新发送这个单个请求。305（UseProxy）必须是 UAS 产生的。

21.3.5 380 Alternative Service

呼叫不成工，但是可以尝试另外的服务。另外的服务在应答的消息体中定义。消息体的格式在这里没有定义，可能在以后的规范中定义。

21.4 请求失败 4xx

4xx 应答定义了特定服务器响应的请求失败的情况。客户端不应当在不更改请求的情况下重新尝试同一个请求。（例如，增加合适的认证信息）。不过，同一个请求交给不同服务器也许就会成功。

21.4.1 400 Bad Request

请求中的语法错误。Reason-Phrase 应当标志这个详细的语法错误，比如”Missing Call-ID header field”。

21.4.2 401 Unauthorized

请求需要用户认证。这个应答是由 UAS 和注册服务器产生的，当 407（Proxy Authentication Required）是 proxy 服务器产生的。

21.4.3 402 Payment Required

保留/以后使用

21.4.4 403 Forbidden

服务端支持这个请求，但是拒绝执行请求。增加验证信息是没有必要的，并且请求应当不被重试。

21.4.5 404 Not Found

服务器返回最终信息：用户在 Request-URI 指定的域上不存在。当 Request-URI 的 domain 和接收这个请求的 domain 不匹配的情况下， 也会产生这个应答。

21.4.6 405 Method Not Allowed

服务器支持 Request-Line 中的方法，但是对于这个 Request-URI 中的地址来说，是不允许应用这个方法的。

应答必须包括一个 Allow 头域，这个头域包含了指定地址允许的方法列表。

21.4.7 406 Not Acceptable

请求中的资源只会导致产生一个在请求中的 Accept 头域外的，内容无法接收的错误。

21.4.8 407 Proxy Authentication Required

这个返回码和 401（Unauthorized）很类四，但是标志了客户端应当首先在 proxy 上通过认证。SIP 对认证的访问请参见 26 节和 22.3 节。

这个返回码用于应用程序访问通讯网关（比如，电话网关），而很少用于被叫方要求认证。

21.4.9 408 Request Timeout

在一段时间内，服务器不能产生一个终结应答，例如，如果它无法及时决定用户的位置。客户端可以在稍后不更改请求的内容然后重新尝试请求。

21.4.10 410 Gone

请求的资源在本服务器上已经不存在了，并且不知道应当把请求转发到哪里。这个问题将会使永久性的。如果服务器不知道，或者不容易检测，这个资源消失是临时性质的还是永久性质的，那么应当返回一个 404（Not Found）。

21.4.11 413 请求实体过大。

服务器拒绝处理请求，因为这个请求的实体超过了服务器希望或者能够处理的大小。这个服务器应当关闭连接避免客户端重发这个请求。

如果这个情况是暂时的，那么服务端应当包含一个 Retry-After 头域来表明这是一个暂时的故障，并且客户端可以过一段时间再次尝试。

21.4.12 414 Request-URI Too Long

服务器拒绝这个请求，因为 Request-URI 超过了服务器能够处理的长度。

21.4.13 415 Unsupported Media Type

服务器由于请求的消息体的格式本服务器不支持，所以拒绝处理这个请求。这个服务器必须根据内容的故障类型，返回一个 Accept，Accpet-Encoding,或者 Accept-Language 头域列表。UAC 根据 8.1.3.5 节定义的方法处理这个应答。

21.4.14 416 Unsupported URI Scheme

服务器由于不支持 Request-URI 中的 URI 方案而终止处理这个请求。客户端处理这个应答参照 8.1.3.5。

21.4.15 Bad Extension

服务器不知道在请求中的 Proxy-Require(20.29)或者 Require(20.32)头域所指出的协议扩展。服务器必须在 Unsupported 头域中列出不支持的扩展。UAC 处理这个应答请参见 8.1.3.5

21.4.16 421Extension Required

UAS 需要特定的扩展来处理这个请求，但是这个扩展并没有在请求的 Supported 头域中列出。具有这个应答码的应答必须包含一个 Require 头域列出所需要的扩展。

UAS 不应当使用这个应答除非它真的不能给客户端提供有效的服务。相反，如果在 Support 头域中没有列出需要的扩展，服务器应当根据基准的 SIP 兼容的方法和客户端支持的扩展来进行处理。

21.4.17 423 Interval Too Brief

服务器因为在请求中设置的资源刷新时间（或者有效时间）过短而拒绝请求。这个应答可以用于注册服务器来拒绝那些 Contact 头域有效期过短的注册请求。这个

应答的用法和相关的 Min-Expires 头域在 10.2.8,10.3,20.23 节中介绍和说明。

21.4.18 480 Temporarily Unavailable

请求成功到达被叫方的终端系统，但是被叫方当前不可用（例如，没有登陆，或者登陆了但是状态是不能通讯，或者有”请勿打扰”的标记）。应答应当在 Retry-After 中标志一个合适的重发时间。这个用户也有可能在其他地方是有效的（在本服务器中不知道）。Reason-Phrase(原因短句)应当提示更详细的原因，为什么被叫方暂时不可用。这个值应当是可以被 UA 设置的。状态码 486（Busy Here）可以用来更精确的表示本请求失败的特定原因。

这个状态码也可以是转发服务或者 proxy 服务器返回的，因为他们发现 Request-URI 指定的用户存在，但是没有一个给这个用户的合适的当前转发的地址。

21.4.19 481 Call/Transaction Does Not Exist

这个状态表示了 UAS 接收到请求，但是没有和现存的对话或者事务匹配。

21.4.20 482 Loop Detected

服务器检测到了一个循环(16.3/4)

21.4.21 483 Too Many Hops

服务器接收到了一个请求包含的 Max-Forwards(20.22)头域是 0

21.4.22 484 Address InComplete

服务器接收到了一个请求，它的 Request-URI 是不完整的。在原因短语中应当有附加的信息说明。这个状态码可以和拨号交叠。在和拨号交叠中，客户端不知道拨号串的长度。它发送增加长度的字串，并且提示用户输入更多的字串，直到不在出现 484（Address Incomplete）应答为止。

21.4.23 485 Ambiguous

Request-URI 是不明确的。应答可以在 Contact 头域中包含一个可能的明确的地址列表。这个提示列表肯囊个在安全性和隐私性对用户或者组织造成破坏。必须能够由配置决定是否以 404（NotFound）代替这个应答，又或者禁止对不明确的地址使用可能的选择列表。

给带有 Request-URI 的请求的一个应答例子：

sip: lee@example.com:

SIP/2.0 485 Ambiguous

Contact: Carol Lee <sip:carol.lee@example.com>

Contact: Ping Lee <sip:p.lee@example.com>

Contact: Lee M.Foote <sips:lee.foote@example.com>

部分 email 和语音邮箱系统提供了这个功能。这个状态码和 3xx 状态码不同：对于 300 来说，它是假定同一个人或者服务有不同的地址选择。所以对 3xx 来说，自动选择系统或者连续查找就有效，但是对 485（Ambiguous）应答来说，一定要用户的干预。

21.4.24 486 Busy Here

当成功联系到被叫方的终端系统，但是被叫方当前在这个终端系统上不能接听这个电话，那么应答应当回给呼叫方一个更合适的时间在 Retry-After 头域重试。这个用户也许在其他地方有效，比如电话邮箱系统等等。如果我们知道没有其他终端系统能够接听这个呼叫，那么应当返回一个状态码 600（Busy Everywhere）。

21.4.25 487 Request Terminated

请求被 BYE 或者 CANCEL 所终止。这个应答永远不会给 CANCEL 请求本身回复。

21.4.26 488 Not Acceptable Here

这个应答和 606（Not Acceptable）有相同的含义，但是只是应用于 Request-URI 所指出的特定资源不能接受，在其他地方请求可能可以接受。

包含了媒体兼容性描述的消息体可以出现在应答中，并且根据 INVITE 请求中的 Accept 头域进行规格化（如果没有 Accept 头域，那么就是 application/sdp）。这个应答就像给 OPTIONS 请求的 200(OK)应答的消息体一样。

21.4.27 491 Request Pending

在同一个对话中，UAS 接收到的请求有一个依赖的请求正在处理。14.2 描述了这种情况应当怎样解决。

21.4.28 493 Undecipherable

UAS 接收到了一个请求，包含了一个加密的 MIME,并且不知道或者没有提供合适的解密密钥。这个应答可以包含单个包体，这个包体包含了合适的公钥，这个公钥用于给这个 UAS 通讯中加密包体使用的。细节描述在 23.2 节。

21.5 Server Failure 5xx

5xx 应答是当服务器本身故障的时候给出的失败应答。

21.5.1 500 Server Internal Error

服务器遇到了未知的情况，并且不能继续处理请求。客户端可以显示特定的错误情况，并且可以在几秒种以后重新尝试这个请求。

如果这个情况是临时的，服务器应当在 Retry-After 头域标志客户端过多少秒钟之后重新尝试这个请求。

21.5.2 501 Not Implemented

服务器没有实现相关的请求功能。当 UAS 不认识请求的方法的时候，并且对每一个用户都无法支持这个方法的时候，应当返回这个应答。（proxy 不考虑请求的方法而转发请求）。

注意 405（Method Not Allowed）是因为服务器实现了这个请求方法，但是这个请求方法在特定请求中不被支持。

21.5.3 502 Bad Gateway

如果服务器，作为 gateway 或者 proxy 存在，从下行服务器上接收到了一个非法的应答（这个应答对应的请求是本服务器为了完成请求而转发给下行服务器的）。

21.5.4 503 Service Unavailable

由于临时的过载或者服务器管理导致的服务器暂时不可用。这个服务器可以在应答中增加一个 Retry-After 来让客户端重试这个请求。如果没有 Retry-After 指出，客户端必须就像收到了一个 500（Server Internal Error）应答一样处理。

客户端（proxy 或者 UAC）收到 503（Service Unavailable）应当尝试转发这个请求到另外一个服务器处理。并且在 Retry-After 头域中指定的时间内，不应当转发其他请求到这个服务器。

作为 503(Service Unavaliable)的替代，服务器可以拒绝连接或者把请求扔掉。

21.5.5 504 Server Time-out

服务器在一个外部服务器上没有收到一个及时的应答。这个外部服务器是本服务器用来访问处理这个请求所需要的。如果从上行服务器上收到的请求中的 Expires 头域超时，那么应当返回一个 408（Request TimeOut）错误。

21.5.6 505 Version Not Supported

服务器不支持对应的 SIP 版本。服务器是无法处理具有客户端提供的相同主版本号的请求，就会导致这样的错误信息。

21.5.7 Message To Large

服务器无法处理请求，因为消息长度超过了处理的长度。

21.6 Global Failures 6xx

6xx 应答意味这服务器给特定用户有一个最终的信息，并不只是在 Request-URI 的特定实例有最终信息。

21.6.1 600 Busy Everywhere

成功联系到被叫方的终端系统，但是被叫方处于忙的状态，并不打算接听电话。这个应答可以通过增加一个 Retry-After 头域更明确的告诉呼叫方多久以后可以继续呼叫。如果被叫方不希望提示拒绝的原因，被叫方应当使用 603（Decline）。

只有当终端系统知道没有其他终端节点（比如语音邮箱系统）能够访问到这个用户的时候才能使用这个应答。否则应当返回一个 486（Busy Here）的应答。

21.6.2 603 Decline

当成功访问到被叫方的设备，但是用户明确的不想应答。这个应答可以通过增加一个 Retry-After 头域更明确的告诉呼叫方多久以后可以继续呼叫。只有当终端知道没有其他任何终端设备能够响应这个呼叫的势能才能给出这个应答。

21.6.3 604 Does Not Exists Anywhere

服务器验证了在请求中 Request-URI 的用户信息，哪里都不存在

21.6.4 606 Not Acceptable

当成功联系到一个 UA,但是会话描述的一些部分比如请求的媒体，带宽，或者地址类型不被接收。

606（NotAcceptable）应答意味着用户希望通讯，但是不能充分支持会话描述。606（Not Acceptable）应答可以在 Warning 头域中包含一个原因列表，用于解释为何会话描述不能被支持。警告原因代码在 20.43 节中列出。

在应答中，可以出现一个包含媒体兼容性描述的消息体，这个消息体的格式根据 INVITE 请求中的 Accept 头域指出的格式进行规格化（如果没有 Accept 头域，那么就是 application/sdp），就像给 OPTIONS 亲求的 200(OK)应答中的消息一样。

我们希望这些媒体协商不要经常需要，并且当一个新用户被邀请加入已经存在的会话的时候，这个媒体协商可能不需要。这取决于邀请的初始化者是否需要对 606（Not Acceptable）进行处理。

这个应答只有当客户端知道没有其他终端能够处理这个请求的时候才能发出。

22 使用 HTTP 认证

SIP 为认证系统提供了一个无状态的，试错机制，这个认证机制式基于 HTTP 的认证机制的。任何时候 proxy 服务器或者 UA 接收到一个请求（22.1 节例外），它尝试检查请求发起者提供的身份确认。当发起方身份确认了，请求的接受方应当确认这个用户是否式通过认证的。在本文档中，没有建议或者讨论认证系统。

本节描述的“Digest”认证机制，只提供了消息认证和复查保护，没有提供消息完整性或者机密性的保证。上述的保护级别和基于这些 Digest 提供的保护，可以防止 SIP 攻击者改变 SIP 请求和应答。

注意由于这个脆弱的安全性，我们不赞成”Basic”（基本的）认证方法。服务器必须不能接收验证方法式”Basic”类型的信任书，并且服务器必须拒绝”Basic”。这是和 RFC2543 的改变。

22.1 框架

SIP 认证的框架和 HTTP 非常接近(RFC2617[17])。特别式，auth-scheme 的 BNF 范式，auth-param,challenge,realm,realm-value,以及信任书都是一样的（虽然对”Basic”认证方案是不允许的）。在 SIP，UAS 使用 401(Unauthorized)应答来拒绝 UAC 的身份（或者讲是考验 UAC 的身份，如果不通过，就是 401）。另外，注册服务器，转发服务器可以使用 401（Unauthorized）来应答身份认证，但是 proxy 必须不能用 401，只能用 407（Proxy Authentication Required）应答。对于 Proxy-Authenticate 的包含

要求，Proxy-Authroization,WWW-Authenticate,Authorization 在不同的消息中是相同的，如同在 RFC2617[17]中讲述的一样。

由于 SIP 并没有一个规范的 root URL 的概念，所以，需要保护的空间的概念在 SIP 中的解释也不一样。realm 字串单独定义被保护的区域。这个是和 RFC2543 的改变，在 2543 中 Request-URI 和 realm 一起定义了被保护的区域。

这个先前定义的被保护的区域回导致一定程度的混乱，因为 Request-URI 是 UAC 发送的，并且接收到 Request-URI 的认证服务器可能是不同的，并且真正的最终的 Request-URI 的格式可能对 UAC 并不知道。同样，早先的定义依赖于一个 Request-URI 中的 SIP URI，并且看起来不允许其他的 URI 方案（比如 tel URL）

需要鉴别接收到的请求的 UA 使用者或者 proxy 服务器，必须根据下边的指导来为他们的服务器创建一个 realm 字串。

o Realm 字串必须是全局唯一的。我们强调这个 realm 字串必须包含一个主机名或者域名，遵循 3.2.1 节或者 RFC2617[17]的推荐

o Realm 字串应当是一个可读的能够展示给用户的字串。
例如：
INVITE sip:bob@biloxi.com SIP/2.0
Authorization: Digest realm=”biloxi.com”,<…>
通常，SIP 认证对于特定 realm（一个保护区域）是有意义的。因此，对于 Digest 认证来说，每一个类似的保护区域都有自己的用户名和密码集合。如果服务器对特定请求没有要求认证，那么它可以接收缺省的用户名,”anonymous”,并且这个用户名没有密码（密码是””）。类似的,代表多个用户的 UAC，比如 PSTN 网关，可以有他们自己的设备相关的用户名和密码，而不是每一个用户名一个用户

名密码（这就是说，比如网关，有一个网关的用户和密码，而不是说通过网关的每一个实际用户和密码）。

当服务器可以正确处理绝大部分 SIP 请求，有本文档约定了两类请求要求特别的认证处理：ACK 和 CANCEL。在某一个认证方案下，并且这个认证方案是使用应答来放置计算 nonces（比如 Digest），那么对于某些没有应答的情况，就会出现问题，比如 ACK。所以，基于这个原因，一个服务器接受在 INVITE 请求中的信任书，也必须同样接收对应 ACK 的信任书。UAC 通过赋值所有的 INVITE 请求中的 Authorization 和 Proxy-Authorization 头域值来创建一个相关的 ACK 消息。服务器必须接收这个 ACK 请求。

虽然 CANCEL 方法具有应答（2xx），服务器必须不能拒绝 CANCEL 请求，因为这些请求不能被重新提交。通常，如果 CANCEL 请求和被 CANCEL 的请求来自同一个节点（假设某种通讯协议，或者网络层有安全关系 26.2.1 节描述），服务器应当接收 CANCEL 请求。

当 UAC 接收到验证拒绝，并且 UAC 设备并不知道 realm 验证失败的具体原因，它必须展示给用户，验证失败的”realm”参数内容（既可以在 WWW-Authenticate 头域或者 Proxy-Authenticate 头域）。对于给自己的 realm 预先配置信任状的 UA 服务提供商来说，应当注意到这样一点：当被一个预先配置信任状的设备拒绝的时候，用户不会有机会在这个 realm 中展示他们自己的信任状。

最后，注意即使一个 UAC 能够定位与相关 realm 匹配的信任书，也有可能存在这个信任书可能不在有效，或者某个服务器会用什么原因不接受这个信任书（特别是当提供的是没有口令的”anonymous”用户时）。在这种情况下，服务器可能会继续拒绝，或者返回一个 403 Forbidden。UAC 必须不能再次使用刚才被拒绝的信任书进行尝试（如果当前环境没有改变，那么请求可以再次尝试）。

22.2 用户到用户的认证。

当 UAS 收到一个 UAC 发起的请求，UAS 在请求被处理之前进行身份认证。如果请求中没有信任书（在 Authorization 头域），UAS 可以使用 401(Unauthorized)拒绝认证，并且让客户端提供一个认证书。

WWW-Authenticate 应答头域必须在 401（Unauthorized）应答消息中出现。这个头域值包含了至少一个表明认证方式和适用 realm 的参数的拒绝原因。

在 401 中的 WWW-Authenticate 头域例子：

WWW-Authenticate:Digest,
realm=”biloxi.com”,
qop=”auth,auth-int”,
nonce=”dcd98b7102dd2f0e8b11d0f600bfb0c093",
opaque="5ccc069c403ebaf9f0171e9517f40e41"

当原始请求的 UAC 接收到这个 401(Unauthorized)应答的时候，如果可能的话，他应当重新组织这个请求，并且填写正确的信任书。在继续处理之前，UAC 可以要求原始用户输入信任书。一旦信任书（不管是用户输入的，还是内部密钥）提供了，UA 应当把这个给特定 To 头域和”realm”字段的信任书 cache 起来，以备给这个地址下一个请求时候使用。UA 可以用任何方式来 cache 这个信任书。

如果没有找到对应 realm 的信任书，UAC 应当尝试用用户”anonymous”和空口令来重新尝试这个请求。

一旦找到了一个信任书，那么 UA 应当要求在 UAS 或者注册服务器上认证自己，这是通常的情况，但是并非一定要求的，在接收到一个 401（Unauthorized）应答后－可以在请求中增加一个 Authorization 头域然后再认证。Authorization

头域包含了具有这个 UA 到请求的资源所在的 realm（区域）的信任书和所需要的认证支持的参数和重现保护的参数。

Authorization 头域例子：

```
Authorization: Digest username=”bob”,
      realm=”biloxi.com”,
      nonce=”dcd98b7102dd2f0e8b11d0f600bfb0c093",
      uri=”sip:bob@biloxi.com”,
qop=auth,
nc=00000001,
      cnonce=”0a4f113b”,
      response=”6629fae49393a05397450978507c4ef1",
      opaque=”5ccc069c403ebaf9f0171e9517f40e41"
```

当 UAC 在接收到 401（Unauthorized）或者 407（ProxyAuthenticationRequired）应答之后，重新用它的信任书来提交请求，它必须增加 Cseq 头域的值，就像发送一个正常的新请求一样。

22.3 Proxy 到用户的认证

类似的，当 UAC 发送一个请求到 proxy 服务器，proxy 服务器可以在处理请求之前，验证原始请求的认证。如果请求中没有信任书（在 Proxy-Authorization 头域），proxy 可以用 407（Proxy Authentication Required）拒绝这个原始请求，并且要求客户端提供适当的信任书。proxy 必须在 407（ProxyAuthenticationRequired）应答中增加一个 Proxy-Authenticate 头域，并且在这个头域中给出适用于本 proxy 的认证资源。

对于 Proxy-Authenticate 和 Proxy-Authorization 一起在[17]中描述，两者有一个不同。Proxy 不能在 Proxy-Authorization 头域中增加值。所有的 407

（ProxyAuthenticationRequired）应答必须转发到上行队列，遵循发送应答的步骤发送到 UAC。UAC 负责在 Proxy-Authorization 头域值增加适用于这个 proxy 要求认证的这个 proxy 的 realm 的信任书。

如果 proxy 要求 UAC 在请求中增加 Proxy-Authorization 头域并且重新提交请求，那么 UAC 应当增加 Cseq 头域的值，就像一个新请求一样。不过，这样就导致提交原始请求的 UAC 需要忽略 UAS 的应答，因为 Cseq 的值可能是不一样的。

当原始请求的 UAC 接收到一个 407(Proxy Authentication Required)的时候，如果可能，它应当使用正确的信任书重新组织请求。它应当和对前边讲述的 401 应答的处理步骤一样显示和处理”realm”参数。

如果没有找到对应 realm 的信任书，那么 UAC 应当尝试用用户”anonymous”和空口令重新尝试请求。

UAC 也应当 cache 这个在重新发送请求中的信任书。

我们建议使用下列步骤来 cache 一个 proxy 的信任书：

如果 UA 在给特定 Call-ID 的请求的 401/407 应答中，接收到一个 Proxy-Authenticate 头域，它应当合并对这个 realm 的信任书，并且为以后具有相同 Call-ID 的请求发送这个信任书。这些信任书必须在对话中被 cache 住；不过如果 UA 配置的是它自己的本地外发 proxy，那么如果出现要求认证的情况，那么 UA 应当 cache 住跨对话的信任书。注意，这个意味着在一个对话中的请求可以包含在 Route 头域中所经过 proxy 都不需要的信任书。

任何希望在 proxy 服务器上认证的 UA——通常，但是并非必须，在接收到 407（Proxy Authentication Required）应答之后——可以在请求中增加一个 Proxy-Authorization 头域然后再次尝试。Proxy-Authorization 请求头域允许客户端像 proxy 来证明自己（或者使用者）的身份。Proxy-Authorization 头域包含了 UA 提供给 proxy 和/或者请求资源所在的 realm 的身份认证信息的信任书。

一个 Proxy-Authorization 头域值只提供给指定 proxy 验证的，这个 proxy 的 realm 是在”realm”参数中指明的（这个 proxy 可以事先通过 Proxy-Authenticat 头域提出认证要求）。当多个 proxy 组成一个链路的时候，如果 proxy 的 realm 和请求中的 Proxy-Authorization 头域的”realm”参数不匹配，那么这个 proxy 就不能使用本 Proxy-Authorization 头域值来验证。

注意，如果一个认证机制不支持 Proxy-Authorization 头域的 realm，porxy 服务器必须尝试分析所有的 Proxy-Authorization 头域值来决定是否其中之一有这个 proxy 认为合适的信任书。因为这个方法在大型网络上很耗时间，proxy 服务器应当使用一个一个支持 Proxy-Authorization 头域的 realm 的认证方案。

如果一个请求被分支（参见 16.7 节），可能对同一个 UAC 有不同的 proxy 服务器和/或者 UA 希望要求认证。在这种情况下，分支的 proxy 服务器有责任把这些被拒绝的认证合并成为一个应答。每一个分支请求的应答中接收到 WWW-Authenticate 和 Proxy-Authenticate 头域值必须由这个分支 proxy 放置在同一个应答中发送给 UA；这些头域值的顺序并没有影响。

当 proxy 服务器给一个请求发出拒绝认证的应答，在 UAC 用正确的信任书重新发请求过来之前，不会转发这个请求。分支 proxy 可以同时向多个要求认证的 proxy 服务器转发请求。每一个 proxy 在没有接收到 UAC 在他们各自的 realm 的认证之前，都不会转发这个请求。如果 UAC 没有给这些失败的验证提供信任

书，那些发出拒绝通过认证的 proxy 是不会把请求转发给 UA 的目标用户的，因此，分支的优点就少了很多。

当针对包含多个拒绝认证的 401（Unauthorized）或者 407（Proxy Authentication Required）应答重新提交请求时，UAC 应当对每一个 WWW-Authenticate 和 Proxy-Authorization 头域值提供一个信任书。根据上边的说明，一个请求的多个认证书应当用”realm”参数分开。

不过，在同一个 401（Unauthorized）或者 407（Proxy Authentication Required）应答中，可能包含对同一个 realm 的多个验证拒绝。例如，当在相同域的多个 proxy，使用共同的 realm，接收到了一个分支请求，并且认证拒绝了的时候，就会有这样的情况。当 UAC 重新尝试这个请求的时候，UAC 因此会提供多个 Authorization 或者 Proxy-Authorization 头域，包含相同的”realm”参数。并且对于同一个 realm，应当有相同的信任书。

22.4 Digest 认证方案

奔进诶描述了对 HTTP Digest 认证方案的 SIP 修改和简化。SIP 使用了和 HTTP[17]几乎完全一样的方案。

由于 RFC 2543 是基于 RFC2069[39]定义的 HTTP Digest 的，支持 RFC2617 的 SIP 服务器也必须确保他们和 RFC2069 兼容。RFC2617 定义了保证兼容性的步骤。注意，SIP 服务器必须不能接收或者发出 Basic 认证请求。

Digest 认证的规则在[17]中定义，只是使用”SIP/2.0”替换”HTTP/1.1”，并且有如下的不同：

1、URI 有着如下的 BNF：

URI=SIP-URI/SIPS-URI

2、在 RFC 2617 定义中，有一个 HTTP Digest 认证的 Authorization 头域”uri”参数的错误，是没有括号配对的错误。（在 RFC2617 的 3.5 节的例子是正确的）。对于 SIP 来说，’uri’参数必须在引号中引起来。

3、digest-uri-value 的 BNF 是：

digest-uri-value=Request-URI; 在 25 节定义。

4、对 SIP 来说，产生基于 Etag 的 nonce 的步骤例子不适用。

5、对 SIP 来说，RFC2617[17]关于 chache 操作不适用。

6、RFC2617[17]要求服务器检查请求行的 URI，并且在 Authorization 头域的 URI 要指向相同的资源。在 SIP 中，这两个 URI 可以指向不同的用户，因为是同一个 proxy 转发的。因此，在 SIP，一个服务器应当检查在 Authorization 头域值的 Request-URI 和服务器希望接收请求的用户是否一致，但是如果两者不一致，并无必要展示成为错误。

7、在 Digest 认证方案中，关于计算消息完整性保证的 A2 值的一个澄清，实现着应当假定，当包体是空的（也就是说，当 SIP 消息没有包体）应当对包体的 hash 值产生一个 M5hash 空串，或者：

H(entity-body)=MD5(“”)=

"d41d8cd98f00b204e9800998ecf8427e"

8、RFC2617 指出了在 Authorization（以及扩展的 Proxy-Authorization）头域中，如果没有 qop 指示参数，就不能出现 cnonce 值。因此，任何基于 cnonce（包括”MD5-Sess”）的运算都要求 qop 指数先发送。在 RFC2617 中的”qop”参数是可选的，这是为了向后兼容 RFC2069；由于 RFC2543 是基于 RFC2069 的，”qop”参数必须被客户和服务器依旧是当作可选参数存在。不过，服务器必须始终在 WWW-Authentication 和 Proxy-Authenticate 头域值中传送”qop”参数。如果一个客户端在一个拒绝认证的应答中收到一个”qop”参数，他必须把这个”qop”参数放在后续的认证头域中。

RFC 2543 不允许使用 Authentication-Info 头域（在 RFC2069 中使用）。不过我们现在允许使用这个头域，因为他提供了对包体的完整性检测以及提供了相互

认证。RFC2617[17]定义了在请求中使用 qop 属性的向后兼容机制。这些机制必须在服务器使用，用来检测是否客户支持 RFC2617 的，没有在 RFC2069 中定义的新机制，

23 S/MIME

SIP 消息可以加载一个 MIME 消息体，并且 MIME 标准包括了 MIME 内容的保密机制，确保完整性和机密性（包括”multipart/signed”和”application/pkcs7-mime”的 MIME 类别，参见 RFC 1847[22],RFC 2630[23],RFC2633[24]）。实现中应当注意，不管怎样，也会有很少的网络节点（不是典型的 proxy 服务器），会依赖于查看修改 SIP 消息（特别是 SDP），采用加密的 MIME 可以防止这类网络节点操作。

这个实现特别对某些类型防火墙有效。

在 RFC2543 描述的对于 SIP 头域和包体的 PGP 加密机制，已经不建议使用了。

23.1 S/MIME 认证

用于区别服务器使用的S/MIME而进行的最终用户的鉴定，有一种重要的特点－这些信任状的持有者是被最终用户地址鉴定的，而不是由一个特定的hostname所鉴定的。这个地址是由SIP或者SIPS URI的”userinfo””@”和”domainname”组成的（换句话说，就是用的email格式，类似bob@biloxi.com）,最常见的就是和用户的address-of-record对应的地址。

这些认证也同样和用于标记或者加密 SIP 消息包体的密钥相关联。包体由发送方的私钥所标记（这个发送方在适当情况下，可以包含他们的公钥），并且包体是由被接受方的公钥所加密。显然，发送方必须事先知道接受方的公钥，这样才能加密包体。公钥可以在 UA 的一个虚拟的钥匙环上保存。

每一个支持 S/MIME 的 UA 都必须包含用于终端用户验证的一个特定密钥组。这个密钥组应当由 address-of-record 和相应的认证信息的映射组成。在时间上，当用户产生具有相同 address-of-record 的原始的信号 URI（From 头域）时，用户应当使用相同的认证信息。

任何依赖于现存的终端用户认证的机制都是严格受限于：事实上，目前没有一个统一的权威认证机构为终端用户应用提供认证。不过，用户应当从已知的公共认证机构获得认证。作为替代，用户可以创建自标记(self-signed)的信任书。self-signed 信任书在 26.4.2 节中有介绍。在实现中，也可以采用在配置中存放预先配置的信任书，这个配置中存放的时上次 SIP 实体之间的信任关系。

在获得终端用户的认证问题上，很少有广为人知的集中发布终端用户认证的目录机构。信任书的拥有者应当在一些公共的目录机构中，适当的发布自己的信任书。类似的，UAC 应当支持从与 SIP 请求的目标 URI 有关的公共目录机构导入信任书（手工或者自动的）。

23.2 S/MIME 密钥交换

SIP 自身可以根据下列的方法发布公钥。

无论何时 SIP 的 S/MIME 使用了 CMS SignedData 消息，他必须包含由公钥所加密的信任书，用于检查签名。

当 UAC 发送一个创建对话的请求，包含了一个 S/MIME 包体，或者在对话外发送一个非 INVITE 请求，UAC 应当创建一个 S/MIME ‘multipart/signed’ CMS SignedData 包体结构的包体。

如果特定的 CMS 服务时 EnvelopedData(并且知道目标用户的公钥)，UAC 应当在一个 SignedData 消息中发送 EnvelopedData 消息。

当 UAS 接收到一个包含 S/MIME CMS 包体的请求，并且这个包体包含一个信任书，UAS 应当首先检查这个信任书，如果可能，使用以后的 root 信任书来进行信任书检查。UAS 也应当检查信任书的主题（subject）（对于 S/MIME，SubjectAltName 应当包含了适当的标记）并且，把这个值和请求 From 头域进行比较。如果信任书不被通过，因为它是 self-signed(自签发)，或者被位置的信任机构所颁发的，或者它是可信任的，但是他的主题和请求的 From 头域不匹配，UAS 必须把这个通知用户（包括信任书的主题，签发者，和密钥指纹信息），并且在继续处理之前，要求用户确认。如果信任书成功通过认证，并且信任书的主题（subject）和 SIP 请求的 From 头域相同，或者如果用户（在知会之后）批准了这个信任书的使用，那么 UAS 应当增加这个信任书到本地密钥组中（keyring），并且用信任书拥有者的 address-of-record 作为索引。

当 UAS 要发送一个包含 S/MIME 消息体的应答，这个应答回应了在对话中的前一个请求，或者是给对话外的一个非 INVITE 请求的应答，UAS 应当构造一个 S/MIME ‘multipart/signed’ CMS SignedData 包体。如果给定的 CMS 服务要求 EnvelopedData,UAS 应当发送一个 EnvelopedData 的 SignedData 消息

当 UAC 收到一个包含 S/MIME CMS 包体的应答，这个应答中包含了一个信任书，UAC 应当首先检查这个信任书，如果可能，使用任何适当的 root 信任书来进行检查。UAC 应当同样检查信任书的主题（subject），并且和应答的 To 头域进行比较；虽然两个可能完全不同，这个也没有必要当作是不安全的。如果因为 self-signed,或者由未知认证机关颁发而导致信任书不能通过认证，那么 UAC 应当通知它的使用者这个状态（包含信任书的主题，它的签发者，以及密钥的指纹信息），并且在继续操作之前，要求使用者确认。如果信任书通过了认证，并且信任书的主题和应答的 To 头域相同，或者用户（在提示确认后）明确的确认这个信任

书以后，UAC 应当把这个信任书放在本地密钥组中，用这个信任书的拥有者的 address-of-record 作为索引。如果 UAC 没有把自己的信任书事先传送给 UAS，它在下一个请求或者应答时应当使用一个 CMS SignedData 包体。

在以后，当 UA 接收到包伙那一个 From 头域的请求或者应答，并且这个 From 头域和本地的一个密钥组的索引匹配，那么 UA 应当比较消息中的信任书和这个密钥组对应索引的信任书。如果他们不匹配，那么 UA 必须通知用户信任书改变了（更合适的是提示这个可能是一个潜在的安全冲突），并且在继续处理前要求用户的确认。如果用户确认了这个信任书，它应当在本地密钥组的这个 address-of-record 项的前一个值旁边附加这个信任书。

注意，这个值的改变机制，在 self-signed 信任书的情况或者未知机关颁发的信任书情况下，并不保证密钥变更的安全性，并且这个缺陷被广为人知的攻击。在本文的作者看来，它提供的安全性当然比什么也没有要强；实际上可以和广泛应用的 SSH 应用相比。这些限制在 26.4.2 节中有详细描述。

如果 UA 接收到了一个 S/MIME 包体，并且是由本接受方所不知道的公钥加密的，它必须用 493（Undecipherable）拒绝这个请求。这个应答应当回答一个合法的信任书（相应的，如果可能，给被拒绝请求的 To 头域指出的全部地址），这个信任书包含一个‘certs-only’ “smime-type”参数的 MIME 包体。。

一个没有任何信任书信息的 493（Undecipherable）应答表示回答者不能或者不会利用 S/MIME 加密信息，虽然他们可以依旧支持 S/MIME 标记。

注意 UA 接收到一个包含一个 S/MIME 包体的请求（这个请求包含了 Content-Disposition 头域和”required”的”handling”参数），如果这个 MIME 类型是不被支持的，那么就必须用 415 Unsupported MediaType 来拒绝这个请求。当 UA 接收到这个应答，并且这个应答表示的是对方不支持刚才发送的 S/MIME 类

型的请求，那么 UA 应当通知它的使用者对方不支持这个 S/MIME 类型，并且如果可以，他可以在随后的请求中不包含 S/MIME 部分；另外，这个 415 应答可以制造一个下级的攻击。

如果一个 UA 在请求中发送了一个 S/MIME 包体，但是接收到的应答包含了一个未加密的 MIME 包体，UAC 应当提示用户，这个会话可能是不安全的。可是，如果一个支持 S/MIME 的 UA 接收到一个包含未加密的包体的请求，它不应当给出一个包含加密包体的应答，但是如果它希望从发送方获得一个 S/MIME（比如，因为发送方的 From 头域值在它的密钥组中存在关联），UAS 应当通知它的使用者这个会话可能是不安全的。

当信任书管理不正常的情况下，许多条件都可以导致前边讲的要给用户提示以及确认的情况。使用者可能会问在这个情况下应当怎样做。首先，一个信任书的异常改变，或者在需要安全保护的时候不安全，是导致这个警告出现的原因，但是并非表示正在遭受攻击。使用者可以中断现有的连接或者拒绝连接请求；在电话谈话中，他们可以挂机并且回叫。使用者可能会使用另外的方式来联系对方并且确认他们的密钥是否合法的更改了。注意用户有时候需要强制更改他们的信任书，例如当他们怀疑他们的私钥不够安全的情况。当他们的私钥不再私钥，用户必须合法的产生一个新的密钥并且重新和其他持有旧密钥的用户建立信任关系。

最后，如果在对话中 UA 收到了在一个 CMS SignedData Message 中的信任书，并且和早先对话中交换的信任书不一致，那么 UA 必须提示它的使用者这个变化，更好的方法是展示成为一个安全隐患。

23.3 加密 MIME 包体

SIP 中，总共由两种类型的加密 MIME 包体：对他们的使用应当遵循 S/MIME 规范[24]，并且有几点不同：

l multipart/signed”只在 CMS 分离签名的时候有用。这是为了和非 S/MIME 规范接受者兼容。

l S/MIME 包体应当包含一个 Content-Disposition 头域，并且他的”handling”参数的值应当是”required”

l 如果 UAC 在它的密钥组中没有对应要发送请求的接受者的 address-of-record 的信任书，它不能发送加密的”application/pkcs7-mime” MIME 消息。UAC 可以发送一个初始化的请求（比如 OPTIONS 消息），包含一个为了方便对方处理的 CMS 分离的签名（签名应当基于一个”message/sip”包体，类型描述在 23.4 节）。

注意以后的 S/MIME 标准工作可以定义基于非信任书的密钥。

l S/MIME 的发送方应当使用”SMIMECapabilities”(参见 2.5.2 [24])属性来为以后的通讯介绍他们自己的能力和参数。注意，特别是那些可能用”preferSignedData”的发送方希望接受方应答一个 CMS SignedData 消息（例如，当发送一个 OPTIONS 请求的时候）。

l S/MIME 实现必须至少支持 SHA1 作为数字签名算法，并且 3DES 作为加密算法。其他的签名和加密算法也可以支持，实现上应当可以用”SMIMECapabilities”属性进行这些算法的协商。

l 在 SIP 中的每一个 S/MIME 包体应当只有一个信任书的签名。如果 UA 接收到的消息有多个信任书，最外边的信任书应当当作这个包体的信任书。并行的签名应当不能使用。

下边是一个 SIP 中的加密的 S/MIME SDP 包体的例子：

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
```

Max-Forwards: 70
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/pkcs7-mime; smime-type=enveloped-data; name=smime.p7m
Content-Disposition: attachment; filename=smime.p7m handling=required
****
* Content-Type: application/sdp *
* *
* v=0 *
* o=alice 53655765 2353687637 IN IP4 pc33.atlanta.com *
* s=- *
* t=0 0 *
* c=IN IP4 pc33.atlanta.com *
* m=audio 3456 RTP/AVP 0 1 3 99 *
* a=rtpmap:0 PCMU/8000 *
****

23.4 SIP 头隐私和用 S/MIME 的完整性：SIP 地道

作为提供端到端的某种程度认证来说，SIP 头域的完整性或者机密性，SI/MIME 可以把 SIP 消息通过”message/sip”的包体类型，整个装入一个 MIME 包体，并且接着就像处理普通 SIP 包体一样，对这个大的包体用 MIME 安全性来保护。这些被封装的 SIP 请求和应答并不产生另外的对话或者事务，他们只是一个用来检验完整性的或者提供附加信息的”outer”外部包装消息。

如果 UAS 接收到一个请求，包含一个”message/sip”隧道的 S/MIME 包体，它应当把应答用同样的 SMIME-TYPE 封装成一个”message/sip”隧道包体。

任何传统的 MIME 包体（比如 SDP），应当附加为”inner”内部消息，这样他们可以使用到 S/MIME 安全性。注意，如果在请求中需要传送未加密的 MIME 类型的内容，那么”message/sip”包体可以作为 MIME “multipart/mixed”包体的一部分存在，

23.4.1 SIP 头的完整性和机密属性

当应用了 S/MIME 完整性或者机密性机制，那么在”inner”消息和”outer”消息中的值可能会有差异。对于所有头域来说，处理这些差异的规则在本节指出。

注意，由于松散的时间戳，所有的在”message/sip”隧道中的 SIP 消息应当在”inner”和”outer”消息中都包含一个 Date 头域。

23.4.1.1 完整性

无论何时，当完整性被检查，头域的完整性应当通过检查被封装包体中的头域值和”outer”消息的头域值，使用 20 节描述的 SIP 比较规则。

头域可以被 proxy 服务器合法修改的是：Request-URI, Via, Record-Route, Route, Max-Forwards, 和 Proxy-Authorization。如果这些头域不完全相等，实现中不应当认为是一个安全错误。对于其他头域的修改就是破坏了完整性，必须通知使用者关于这个差异。

23.4.1.2 机密性

当消息加密以后，头域可以在加密的包体中存在，而不用在”outer”消息中存在。

有一些头域必须有一个明码格式的版本，因为他们是请求和应答中必须要求的头域－他们是：To, From, Call-ID, Cseq， Contact。 虽然提供一个额外的 Call-ID,Cseq,或者 Contact 的加密版本没有什么用处，我们允许把 To/From 放一份在”outer”消息的 To 或者 From 头域中。注意这些加密包体中的值并不用作鉴别事务或者对话——加密包体中的这些值仅仅作为提示信息使用。如果在加密包体中的 From 头域和在”outer”消息中的值不一样，那么这个加密包体中的值应当显示给用户，但是不能用于后续”outer”消息的头域中。

首先，UA 应当希望加密有着端到端语义的头域，包括: Subject,Replay-To,Organization, Accept,Accept-Encoding,Accept-Language,Alter-Info,Error-Info, Authentication-Info , Expires, In-Replay-To, Require, Supported, Unsupported, Retry-After, User-Agent, Server, 和 Warning。 如果这些头域在加密包体中出现，那么就应当替换在”outer”消息中的头域，而不管这个头域是显示给用户的还是设置 UA 内部状态的。他们应当在后续消息中不在在”outer”消息的头域中使用。

如果存在 Date 头域，那么 Date 头域必须在”inner”和”outer”头域中的值一样。

由于 MIME 包体是在”inner”消息中的，实现中通常会加密 MIME 指定的头域，包括：MIME-Version,Content-Type,Content-Length, Content-Language, Content-Encoding, 和 Content-Disposition。”outer”消息会为 S/MIME 包体使用适当的 MIME 头域。这些头域（和他们开始的任何 MIME 包体）在 SIP 消息中，应当作为普通的 MIME 头域和包体接收。

对下边这些头域的加密并不是特别有用： Min-Expires, Timestamp, Authorization, Priority, 和 WWWAuthenticate。这类头域包含了那些能够被 proxy 服务器所更改的头域（在前边章节有讲述）。如果这些头域不出现

在”outer”消息中，那么 UA 应当不在”inner”消息中包含这些头域。如果 UA 在加密的包体中接收到这些头域，应当忽略到这些加密的值。

注意，SIP 的扩展可能会定义附加的头域；那么这些扩展的作者应当描述这些头域的完整性和机密性特性。如果一个 SIP UA 遇到了一个不认识的头域，并且产生了一个完整性冲突，它必须忽略掉这个头域。

23.4.2 隧道的完整性和身份认证

通过 S/MIME 包体的 SIP 消息隧道可以提供 SIP 头域的完整性保证，只要发送方把这个包整个打包放在用 CMS 分离签名的”message/sip” MIME 包体中。

假设那个”message/sip”包体包含了最基本的对话要素（最小集合）（To，From， Call-ID， CSEq），并且一个签名的 MIME 包体可以提供优先的身份认证。在这个特别的最小集合上，如果接受方不认识用于签名 MIME 包体的信任状，并且不能被检验，那么在同一个信任状拥有者所初始化的会话中，这个信任状拥有者可以稍后发送一个在会话中的请求，包含这个签名来进行确认。如果签名 MIME 包体的接受方选择信任这个信任书（他们可以检验信任书，他们已经从信任列表中检查了，或者他们经常使用这个信任书），那么这个签名就和这个信任书的主题有着相同的身份一致性。

为了排出可能的对实体包头域增加减少的相关困惑，发送方应当把请求的所有头域放在签名的包体中。任何需要完整性保护的包体都必须添加到”inner”消息中。

如果有一个签名包体的消息中包含一个 Date 头域，接受方应当比较它自己的内部时钟和这个头域值。如果检测到了时差（比如超过 1 个小时或者更多），UA 应当警告使用者，并且提示这个可能是安全隐患。

如果接受方检测到消息的完整性破坏了，如果是这个消息是请求，那么应当使用403（Forbidden）来拒绝这个请求，或者终止现存的对话。UA 应当提示用户这个情况，并且要求明确的操作指示。

下边是一个使用隧道”message/sip”的例子：

```
INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: multipart/signed;
protocol="application/pkcs7-signature";
micalg=sha1; boundary=boundary42
Content-Length: 568

--boundary42
Content-Type: message/sip

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <bob@biloxi.com>
From: Alice <alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
```

Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 147

v=0
o=UserA 2890844526 2890844526 IN IP4 here.com
s=Session SDP
c=IN IP4 pc33.atlanta.com
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s;
handling=required
ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42-

23.4.3 隧道加密

把”message/sip”MIME 包体用 CMS EnvelopedData 消息 S/MIME 包体进行加密是值得的，但是在实践中，大部分头域都是用于网络的；而通常使用 S/MIME 进行的加密都是加密类似 SDP 的消息体的，而不是消息头的。有一部分头域的信息，比如 Subject 或者 Organization 或许需要端到端的安全保证。以后的 SIP 应用可能会定义其他的头域，这些头域也或许需要端到端的安全保证。

另一个加密头域的可能的应用是选择性匿名。可以构造一个没有个人信息的 From 头域（比如 sip:anonymous@anonymizer.invalid）。不过，第二个 From 头域包含了真实的请求者的 address-of-record 信息，并且加密存放在”message/sip”MIME 包体中，并且只会在对话的对方节点被看到。

注意如果这个机制用于匿名，那么将接受方将不再用 From 头域来作为密钥组的索引，并且也不用于从密钥组查询合适的发送方的 S/MIME 密钥。这个消息必须首先被解密，并且”inner”From 头域必须当作一个索引。

为了提供端到端的完整性，加密的”message/sip”MIME 包体应当由发送方签名。这个创建了一个包含一个加密包体和一个签名的”multipart/signed” MIME 包体，包体类型都是”application/pkcs7-mime”.。

在下边这个例子中，是一个加密和签名的消息，在*括起来的文字是加密的：

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
To: Bob <sip:bob@biloxi.com>
From: Anonymous <sip:anonymous@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE

Max-Forwards: 70
Date: Thu, 21 Feb 2002 13:02:03 GMT
Contact: <sip:pc33.atlanta.com>
Content-Type: multipart/signed;
protocol="application/pkcs7-signature";
micalg=sha1; boundary=boundary42
Content-Length: 568

--boundary42
Content-Type: application/pkcs7-mime; smime-type=enveloped-data;
name=smime.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7m
handling=required
Content-Length: 231

****
****
* Content-Type:
message/sip *
*
  *
* INVITE sip:bob@biloxi.com
SIP/2.0 *
* Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8 *
* To: Bob
<bob@biloxi.com> *
* From: Alice <alice@atlanta.com>;tag=1928301774 *

* Call-ID:
a84b4c76e66710 *
* CSeq: 314159
INVITE *
* Max-Forwards:
70 *
* Date: Thu, 21 Feb 2002 13:02:03 GMT *
* Contact:
<sip:alice@pc33.atlanta.com> *
*
*
* Content-Type:
application/sdp *
*
*
*
v=0
*
* o=alice 53655765 2353687637 IN IP4
pc33.atlanta.com *
* s=Session
SDP *
* t=0
0 *
* c=IN IP4
pc33.atlanta.com *
* m=audio 3456 RTP/AVP 0 1 3
99 *

* a=rtpmap:0
PCMU/8000 *
****
****

--boundary42
Content-Type: application/pkcs7-signature; name=smime.p7s
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=smime.p7s;
handling=required

ghyHhHUujhJhjH77n8HHGTrfvbnj756tbB9HG4VQpfyF467GhIGfHfYT6
4VQpfyF467GhIGfHfYT6jH77n8HHGghyHhHUujhJh756tbB9HGTrfvbnj
n8HHGTrfvhJhjH776tbB9HG4VQbnj7567GhIGfHfYT6ghyHhHUujpfyF4
7GhIGfHfYT64VQbnj756

--boundary42-

24 例子

在下边这个例子中，由于简略介绍的关系我们经常省略消息体和对应的 Content-Length 和 Content-Type 头域。

24.1 注册

Bob 在启动的时候进行注册。这个消息流在图 9 中展示。注意对于注册服务来说，通常需要认证，而且不像下边描述的这样简单。

REGISTER F1
biloxi.com
registar
Bob’s
Softphone

图 9： SIP 注册例子

F1 REGISTER Bob -> Registrar
REGISTER sip:registrar.biloxi.com SIP/2.0
Via: SIP/2.0/UDP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Bob <sip:bob@biloxi.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0

注册会在 2 小时后超时。注册服务器回应一个 200OK:
F2 200 OK Registar -> Bob
SIP/2.0 200 OK
Via: SIP/2.0/UDP bobspc.biloxi.com:5060;branch=z9hG4bKnashds7
;received=192.0.2.4
To: Bob <sip:bob@biloxi.com>;tag=2493k59kd
From: Bob <sip:bob@biloxi.com>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER

Contact: <sip:bob@192.0.2.4>
Expires: 7200
Content-Length: 0

24.2 建立会话

这个例子包括了 4 节描述的建立会话的细节。消息流在图 1 中展示了。注意这些消息流展示了头域的最小集合——一般来说还需要包含一些其他头域，比如 Allow 和 Supported 头域。

F1 INVITE Alice -> atlanta.com proxy

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(Alice's SDP not shown)

F2 100 Trying atlanta.com proxy -> Alice

SIP/2.0 100 Trying
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1

To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Content-Length: 0

F3 INVITE atlanta.com proxy -> biloxi.com proxy

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP
bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
Max-Forwards: 69
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(Alice's SDP not shown)

F4 100 Trying biloxi.com proxy -> atlanta.com proxy

SIP/2.0 100 Trying
Via: SIP/2.0/UDP
bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1

;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Content-Length: 0

F5 INVITE biloxi.com proxy -> Bob

INVITE sip:bob@192.0.2.4 SIP/2.0
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
Via: SIP/2.0/UDP
bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
Max-Forwards: 68
To: Bob <sip:bob@biloxi.com>
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:alice@pc33.atlanta.com>
Content-Type: application/sdp
Content-Length: 142

(Alice’s SDP not shown)

F6 180 Ringing Bob -> biloxi.com proxy

SIP/2.0 180 Ringing
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
;received=192.0.2.3
Via: SIP/2.0/UDP
bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0

F7 180 Ringing biloxi.com proxy -> atlanta.com proxy

SIP/2.0 180 Ringing
Via: SIP/2.0/UDP
bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>

CSeq: 314159 INVITE
Content-Length: 0

F8 180 Ringing atlanta.com proxy -> Alice

SIP/2.0 180 Ringing
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
Contact: <sip:bob@192.0.2.4>
CSeq: 314159 INVITE
Content-Length: 0

F9 200 OK Bob -> biloxi.com proxy

SIP/2.0 200 OK
Via: SIP/2.0/UDP server10.biloxi.com;branch=z9hG4bK4b43c2ff8.1
;received=192.0.2.3
Via: SIP/2.0/UDP
bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE

Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(Bob’s SDP not shown)

F10 200 OK biloxi.com proxy -> atlanta.com proxy

SIP/2.0 200 OK
Via: SIP/2.0/UDP
bigbox3.site3.atlanta.com;branch=z9hG4bK77ef4c2312983.1
;received=192.0.2.2
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(Bob’s SDP not shown)

F11 200 OK atlanta.com proxy -> Alice

SIP/2.0 200 OK
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds8
;received=192.0.2.1

To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 INVITE
Contact: <sip:bob@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131

(Bob's SDP not shown)

F12 ACK Alice -> Bob

ACK sip:bob@192.0.2.4 SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKnashds9
Max-Forwards: 70
To: Bob <sip:bob@biloxi.com>;tag=a6c85cf
From: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 314159 ACK
Content-Length: 0

在 Alice 和 Bob 之间的媒体会话现在建立了。Bob 首先挂机。注意 Bob 的 SIP 电话维持它自己的 Cseq 号码空间，在这里，是 231 开始的。由于 Bob 发起请求，那么 To 和 From URI 和 tags 交换了。

F13 BYE Bob -> Alice

BYE sip:alice@pc33.atlanta.com SIP/2.0
Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10

Max-Forwards: 70
From: Bob <sip:bob@biloxi.com>;tag=a6c85cf
To: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 231 BYE
Content-Length: 0

F14 200 OK Alice -> Bob

SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.0.2.4;branch=z9hG4bKnashds10
From: Bob <sip:bob@biloxi.com>;tag=a6c85cf
To: Alice <sip:alice@atlanta.com>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 231 BYE
Content-Length: 0

这个 SIP 呼叫流程文档[40]包含了 SIP 消息的更多例子。

25 SIP 协议的 BNF 范式

本协议中定义的机制都用文本和 Backus-Naur Form(BNF)范式定义（RFC2234[10]）。6.1 节和 RFC2234 定义了一个本文档使用的基本核心规则，这里就不赘述了。实现者需要熟悉 RFC2234 协议，这样才能理解整理定义的规范。某些基本规则是用大写字母表示的，比如 SP,LWS,HTAB,CRLF,DIGIT,ALPHA,等等。尖括号定义了规则的名字。

方括号的使用是在语法上可选的。在这里用于特定参数是可选的语义提示。

25.1 基本规则

下列贯穿本规范的规则是用于描述基本的语法结构。US-ASCII 码字符集是在 ANSI X3.4-1986 中定义的。

alphanum = ALPHA/DIGIT

部分规则是和 RFC2396[5]中合并使用的，但是依据 RFC2234[10]做了更新，这些包括：

reserved = ";" / "/" / "?" / ":" / "@" / "&" / "=" / "+" / "$" / ","

unreserved = alphanum / mark
mark = "-" / "_" / "." / "!" / "~" / "*" / "'" / "(" / ")"

escaped = "%" HEXDIG HEXDIG

SIP 头域值可以折叠成为多行，如果每行的开始是一个空格或者一个水平制表符（就是 Tab 键啦）。所有的线形空白，包含折叠的空白，和 SP 有着同样的语义（SP 就是空格啦）。一个接受方在处理头域值之前或者转发消息到下行队列之前，可以把任何线形空白当作一个单个 SP 处理。这个和 RFC2616[8]中描述的 HTTP/1.1 处理方法完全一样。当线形空白是可选的时候，SWS 构造就需要了，通常在两个符号和分隔符之间存在：

LWS = [*WSP CRLF] 1*WSP ; linear whitespace
SWS = [LWS] ; sep whitespace

为了把头域名和头域值分开，就需要使用冒号，根据上边的规则，允许在冒号之前有空白，但是不允许有行分隔符，并且允许在冒号之后有空白，或者行分隔符。HCOLON 有如下定义：

HCOLON = *(SP / HTAB) ":" SWS

TEXT-UTF8 规则只用于描述不被消息分析所分析的域内容和域值。*TEXT-UTF8 包含了 UTF-8 字符集的字符（RFC2279[7]）。TEXT-UTF8-TRIM 规则是用于描述一个 n t 引号引起来的字符串，其前后的空白是无意义的。在这里，SIP 和 HTTP 不同，HTTP 使用的是 ISO 8859-1 字符集。

TEXT-UTF8-TRIM = 1*TEXT-UTF8char *(*LWS TEXT-UTF8char)
TEXT-UTF8char = %x21-7E / UTF8-NONASCII
UTF8-NONASCII = %xC0-DF 1UTF8-CONT
/ %xE0-EF 2UTF8-CONT
/ %xF0-F7 3UTF8-CONT
/ %xF8-Fb 4UTF8-CONT
/ %xFC-FD 5UTF8-CONT
UTF8-CONT = %x80-BF

在 TEXT-UTF8-TRIM 的定义中，CRLF 只允许作为头域的延长部分存在。当 LWS（空格）折叠的时候，在分析 TEXTUTF8-TRIM 之前，会使用单个 SP 代替。

部分协议要素使用了十六进制数字字符。有一些要素（authentication）强制十六进制数字使用小写字母。

LHEX = DIGIT / %x61-66 ;lowercase a-f

许多 SIP 头域值都包含用 LWS 或者特殊符号分开的词。除非有额外的说明，这些符号是大小写不铭感的。当特殊字符作为参数值存在的时候，这些特殊字符必须通过引号引起来。Call-ID 中建立的词组允许使用绝大部分分隔符。

token = 1 * (alphanum / "-" / "." / "!" / "%" / "*"
/ "_" / "+" / "`" / "'" / "~")
separators = "(" / ")" / "<" / ">" / "@" /
"," / ";" / ":" / "\" / DQUOTE /
"/" / "[" / "]" / "?" / "=" /
"{" / "}" / SP / HTAB
word = 1*(alphanum / "-" / "." / "!" / "%" / "*" /
"_" / "+" / "`" / "'" / "~" /
"(" / ")" / "<" / ">" /
":" / "\" / DQUOTE /
"/" / "[" / "]" / "?" /
"{" / "}")

当标志符号或者分隔符用在要素之间是，空白通常允许在这些字符之前或者之后。

STAR = SWS "*" SWS ; asterisk
SLASH = SWS "/" SWS ; slash
EQUAL = SWS "=" SWS ; equal
LPAREN = SWS "(" SWS ; left parenthesis
RPAREN = SWS ")" SWS ; right parenthesis
RAQUOT = ">" SWS ; right angle quote
LAQUOT = SWS "<"; left angle quote
COMMA = SWS "," SWS ; comma
SEMI = SWS ";" SWS ; semicolon
COLON = SWS ":" SWS ; colon
LDQUOT = SWS DQUOTE; open double quotation mark
RDQUOT = DQUOTE SWS ; close double quotation mark

在 SIP 头域中可以使用注释，通过把注释放在圆括号中就可以了。只有在头域的定义中允许”comment”作为他们的头域值的一部分才可以使用注释。在其他头域中，圆括号被视同为头域值的一部分。

comment = LPAREN *(ctext / quoted-pair / comment) RPAREN
ctext = %x21-27 / %x2A-5B / %x5D-7E / UTF8-NONASCII
/ LWS

ctext 包含了除了左右括号和反斜线之外的所有的字符。在双引号引起来的字符串中的字串，被视为单个词。在引起来的字串中，引号(“)和反斜线需要转码。

quoted-string = SWS DQUOTE *(qdtext / quoted-pair) DQUOTE
qdtext = LWS / %x21 / %x23-5B / %x5D-7E
/ UTF8-NONASCII

反斜线(“\”)可以当作单个字符使用，转义机制只有在引号引起来的字符串中或者注释结构中有效。和 HTTP/1.1 不同的是，CR 和 LF 不能通过这个机制来转义，这样可以避免同头的折叠的冲突。

quoted-pair = "\" (%x00-09 / %x0B-0C
/ %x0E-7F)

SIP-URI = "sip:" [userinfo] hostport
uri-parameters [headers]
SIPS-URI = "sips:" [userinfo] hostport
uri-parameters [headers]
userinfo = (user / telephone-subscriber) [":"
password] "@"
user = 1*(unreserved / escaped / user-unreserved)

```
user-unreserved     =    "&" / "=" / "+" / "$" / "," / ";" / "?" / "/"
password            =    *( unreserved / escaped /
"&" / "=" / "+" / "$" / "," )
hostport            =    host [ ":" port ]
host                =    hostname / IPv4address / IPv6reference
hostname            =    *( domainlabel "." ) toplabel [ "." ]
domainlabel         =    alphanum
/ alphanum *( alphanum / "-" ) alphanum
toplabel            =    ALPHA / ALPHA *( alphanum / "-" )
alphanum
IPv4address         =    1*3DIGIT "." 1*3DIGIT "." 1*3DIGIT "."
1*3DIGIT
IPv6reference       =    "[" IPv6address "]"
IPv6address         =    hexpart [ ":" IPv4address ]
hexpart             =    hexseq / hexseq "::" [ hexseq ] / "::"
[ hexseq ]
hexseq              =    hex4 *( ":" hex4)
hex4                =    1*4HEXDIG
port                =    1*DIGIT
```

对于电话描述（telephone-subscriber）的 BNF 说明在 RFC2806[9]中。注意，无论如何，如果在这里允许的字符，如果在 SIP URI 中的 user 部分不许可，那么就一定要用转义。

```
uri-parameters      =    *( ";" uri-parameter)
uri-parameter       =    transport-param / user-param / method-
param
/ ttl-param / maddr-param / lr-param / other-param
transport-param     =    "transport="
```

("udp" / "tcp" / "sctp" / "tls"
/ other-transport)
other-transport = token
user-param = "user=" ("phone" / "ip" / other-user)
other-user = token
method-param = "method=" Method
ttl-param = "ttl=" ttl
maddr-param = "maddr=" host
lr-param = "lr"
other-param = pname ["=" pvalue]
pname = 1*paramchar
pvalue = 1*paramchar
paramchar = param-unreserved / unreserved /
escaped
param-unreserved = "[" / "]" / "/" / ":" / "&" / "+" / "$"

headers = "?" header *("&" header)
header = hname "=" hvalue
hname = 1*(hnv-unreserved / unreserved / escaped)
hvalue = *(hnv-unreserved / unreserved / escaped)
hnv-unreserved = "[" / "]" / "/" / "?" / ":" / "+" / "$"

SIP-message = Request / Response
Request = Request-Line
*(message-header)
CRLF
[message-body]
Request-Line = Method SP Request-URI SP SIP-Version CRLF
Request-URI = SIP-URI / SIPS-URI / absoluteURI

absoluteURI = scheme ":" (hier-part / opaque-part)
hier-part = (net-path / abs-path) ["?" query]
net-path = "//" authority [abs-path]
abs-path = "/" path-segments
opaque-part = uric-no-slash *uric
uric = reserved / unreserved / escaped
uric-no-slash = unreserved / escaped / ";" / "?" / ":" / "@"
/ "&" / "=" / "+" / "$" / ","
path-segments = segment *("/" segment)
segment = *pchar *(";" param)
param = *pchar
pchar = unreserved / escaped /
":" / "@" / "&" / "=" / "+" / "$" / ","
scheme = ALPHA *(ALPHA / DIGIT / "+" / "-" / ".")
authority = srvr / reg-name
srvr = [[userinfo "@"] hostport]
reg-name = 1*(unreserved / escaped / "$" / ","
/ ";" / ":" / "@" / "&" / "=" / "+")
query = *uric
SIP-Version = "SIP" "/" 1*DIGIT "." 1*DIGIT

message-header = (Accept
/ Accept-Encoding
/ Accept-Language
/ Alert-Info
/ Allow
/ Authentication-Info
/ Authorization
/ Call-ID

/ Call-Info
/ Contact
/ Content-Disposition
/ Content-Encoding
/ Content-Language
/ Content-Length
/ Content-Type
/ CSeq
/ Date
/ Error-Info
/ Expires
/ From
/ In-Reply-To
/ Max-Forwards
/ MIME-Version
/ Min-Expires
/ Organization
/ Priority
/ Proxy-Authenticate
/ Proxy-Authorization
/ Proxy-Require
/ Record-Route
/ Reply-To
/ Require
/ Retry-After
/ Route
/ Server
/ Subject
/ Supported

/ Timestamp
/ To
/ Unsupported
/ User-Agent
/ Via
/ Warning
/ WWW-Authenticate
/ extension-header) CRLF

INVITEm = %x49.4E.56.49.54.45 ; INVITE in caps
ACKm = %x41.43.4B ; ACK in caps
OPTIONSm = %x4F.50.54.49.4F.4E.53 ; OPTIONS in caps
BYEm = %x42.59.45 ; BYE in caps
CANCELm = %x43.41.4E.43.45.4C ; CANCEL in caps
REGISTERm = %x52.45.47.49.53.54.45.52 ; REGISTER in caps
Method = INVITEm / ACKm / OPTIONSm / BYEm / CANCELm / REGISTERm
/ extension-method
extension-method = token
Response = Status-Line
*(message-header)
CRLF
[message-body]

Status-Line = SIP-Version SP Status-Code SP Reason-Phrase CRLF
Status-Code = Informational
/ Redirection

/ Success
/ Client-Error
/ Server-Error
/ Global-Failure
/ extension-code
extension-code = 3DIGIT
Reason-Phrase = *(reserved / unreserved / escaped
/ UTF8-NONASCII / UTF8-CONT / SP / HTAB)

Informational = "100" ; Trying
/ "180" ; Ringing
/ "181" ; Call Is Being Forwarded
/ "182" ; Queued
/ "183" ; Session Progress

Success = "200" ; OK

Redirection = "300" ; Multiple Choices
/ "301" ; Moved Permanently
/ "302" ; Moved Temporarily
/ "305" ; Use Proxy
/ "380" ; Alternative Service
Client-Error = "400" ; Bad Request
/ "401" ; Unauthorized
/ "402" ; Payment Required
/ "403" ; Forbidden
/ "404" ; Not Found
/ "405" ; Method Not Allowed
/ "406" ; Not Acceptable

/ "407" ; Proxy Authentication Required

/ "408" ; Request Timeout

/ "410" ; Gone

/ "413" ; Request Entity Too Large

/ "414" ; Request-URI Too Large

/ "415" ; Unsupported Media Type

/ "416" ; Unsupported URI Scheme

/ "420" ; Bad Extension

/ "421" ; Extension Required

/ "423" ; Interval Too Brief

/ "480" ; Temporarily not available

/ "481" ; Call Leg/Transaction Does Not Exist

/ "482" ; Loop Detected

/ "483" ; Too Many Hops

/ "484" ; Address Incomplete

/ "485" ; Ambiguous

/ "486" ; Busy Here

/ "487" ; Request Terminated

/ "488" ; Not Acceptable Here

/ "491" ; Request Pending

/ "493" ; Undecipherable

Server-Error = "500" ; Internal Server Error

/ "501" ; Not Implemented

/ "502" ; Bad Gateway

/ "503" ; Service Unavailable

/ "504" ; Server Time-out

/ "505" ; SIP Version not supported

/ "513" ; Message Too Large

```
Global-Failure          =    "600" ; Busy Everywhere
/ "603" ; Decline
/ "604" ; Does not exist anywhere
/ "606" ; Not Acceptable

Accept                   =    "Accept" HCOLON
[ accept-range *(COMMA accept-range) ]
accept-range            =    media-range *(SEMI accept-param)
media-range             =    ( "*/*"
/ ( m-type SLASH "*" )
/ ( m-type SLASH m-subtype )
) *( SEMI m-parameter )
accept-param            =    ("q" EQUAL qvalue) / generic-param
qvalue                   =    ( "0" [ "." 0*3DIGIT ] )
/ ( "1" [ "." 0*3("0") ] )
generic-param           =    token [ EQUAL gen-value ]
gen-value               =    token / host / quoted-string

Accept-Encoding         =    "Accept-Encoding" HCOLON
[ encoding *(COMMA encoding) ]
encoding                =    codings *(SEMI accept-param)
codings                 =    content-coding / "*"
content-coding          =    token

Accept-Language         =    "Accept-Language" HCOLON
[ language *(COMMA language) ]
language                =    language-range *(SEMI accept-param)
```

language-range = ((1*8ALPHA *("-" 1*8ALPHA)) / "*")

Alert-Info = "Alert-Info" HCOLON alert-param *(COMMA alert-param)
alert-param = LAQUOT absoluteURI RAQUOT *(SEMI generic-param)

Allow = "Allow" HCOLON [Method *(COMMA Method)]
Authorization = "Authorization" HCOLON credentials
credentials = ("Digest" LWS digest-response) / other-response
digest-response = dig-resp *(COMMA dig-resp)
dig-resp = username / realm / nonce / digest-uri / dresponse / algorithm / cnonce / opaque / message-qop / nonce-count / auth-param
username = "username" EQUAL username-value
username-value = quoted-string
digest-uri = "uri" EQUAL LDQUOT digest-uri-value RDQUOT
digest-uri-value = rquest-uri ; Equal to request-uri as specified by HTTP/1.1
message-qop = "qop" EQUAL qop-value

cnonce = "cnonce" EQUAL cnonce-value
cnonce-value = nonce-value
nonce-count = "nc" EQUAL nc-value

nc-value = 8LHEX
dresponse = "response" EQUAL request-digest
request-digest = LDQUOT 32LHEX RDQUOT
auth-param = auth-param-name EQUAL
(token / quoted-string)
auth-param-name = token
other-response = auth-scheme LWS auth-param
*(COMMA auth-param)
auth-scheme = token

Authentication-Info = "Authentication-Info" HCOLON ainfo
*(COMMA ainfo)
ainfo = nextnonce / message-qop
/ response-auth / cnonce
/ nonce-count
nextnonce = "nextnonce" EQUAL nonce-value
response-auth = "rspauth" EQUAL response-digest
response-digest = LDQUOT *LHEX RDQUOT

Call-ID = ("Call-ID" / "i") HCOLON callid
callid = word ["@" word]

Call-Info = "Call-Info" HCOLON info *(COMMA info)
info = LAQUOT absoluteURI RAQUOT *(SEMI info-
param)
info-param = ("purpose" EQUAL ("icon" / "info"
/ "card" / token)) / generic-param
Contact = ("Contact" / "m") HCOLON
(STAR / (contact-param *(COMMA contact-param)))

contact-param = (name-addr / addr-spec) *(SEMI contact-params)
name-addr = [display-name] LAQUOT addr-spec RAQUOT
addr-spec = SIP-URI / SIPS-URI / absoluteURI
display-name = *(token LWS)/ quoted-string

contact-params = c-p-q / c-p-expires
/ contact-extension
c-p-q = "q" EQUAL qvalue
c-p-expires = "expires" EQUAL delta-seconds
contact-extension = generic-param
delta-seconds = 1*DIGIT

Content-Disposition = "Content-Disposition" HCOLON
disp-type *(SEMI disp-param)
disp-type = "render" / "session" / "icon" / "alert"
/ disp-extension-token
disp-param = handling-param / generic-param
handling-param = "handling" EQUAL
("optional" / "required"
/ other-handling)
other-handling = token
disp-extension-token = token

Content-Encoding = ("Content-Encoding" / "e") HCOLON
content-coding *(COMMA content-coding)

Content-Language = "Content-Language" HCOLON

language-tag *(COMMA language-tag)
language-tag = primary-tag *("-" subtag)
primary-tag = 1*8ALPHA
subtag = 1*8ALPHA

Content-Length = ("Content-Length" / "l") HCOLON 1*DIGIT
Content-Type = ("Content-Type" / "c") HCOLON media-type
media-type = m-type SLASH m-subtype *(SEMI m-parameter)
m-type = discrete-type / composite-type
discrete-type = "text" / "image" / "audio" / "video" / "application" / extension-token
composite-type = "message" / "multipart" / extension-token
extension-token = ietf-token / x-token
ietf-token = token
x-token = "x-" token
m-subtype = extension-token / iana-token
iana-token = token
m-parameter = m-attribute EQUAL m-value
m-attribute = token
m-value = token / quoted-string

CSeq = "CSeq" HCOLON 1*DIGIT LWS Method

Date = "Date" HCOLON SIP-date
SIP-date = rfc1123-date
rfc1123-date = wkday "," SP date1 SP time SP "GMT"

```
date1                =    2DIGIT SP month SP 4DIGIT
; day month year (e.g., 02 Jun 1982)
time                 =    2DIGIT ":" 2DIGIT ":" 2DIGIT
; 00:00:00 - 23:59:59
wkday                =    "Mon" / "Tue" / "Wed"
/ "Thu" / "Fri" / "Sat" / "Sun"
month                =    "Jan" / "Feb" / "Mar" / "Apr"
/ "May" / "Jun" / "Jul" / "Aug"
/ "Sep" / "Oct" / "Nov" / "Dec"

Error-Info           =    "Error-Info" HCOLON error-uri *(COMMA
error-uri)

error-uri            = LAQUOT absoluteURI RAQUOT *( SEMI
generic-param )

Expires              =    "Expires" HCOLON delta-seconds
From                 =    ( "From" / "f" ) HCOLON from-spec
from-spec            =    ( name-addr / addr-spec )
*( SEMI from-param )
from-param           =    tag-param / generic-param
tag-param            =    "tag" EQUAL token

In-Reply-To          =    "In-Reply-To" HCOLON callid *(COMMA callid)

Max-Forwards         =   "Max-Forwards" HCOLON 1*DIGIT

MIME-Version         =    "MIME-Version" HCOLON 1*DIGIT "."
1*DIGIT
```

Min-Expires = "Min-Expires" HCOLON delta-seconds

Organization = "Organization" HCOLON [TEXT-UTF8-TRIM]

Priority = "Priority" HCOLON priority-value
priority-value = "emergency" / "urgent" / "normal"
/ "non-urgent" / other-priority
other-priority = token

Proxy-Authenticate = "Proxy-Authenticate" HCOLON challenge
challenge = ("Digest" LWS digest-cln *(COMMA digest-cln))
/ other-challenge
other-challenge = auth-scheme LWS auth-param
*(COMMA auth-param)
digest-cln = realm / domain / nonce
/ opaque / stale / algorithm
/ qop-options / auth-param
realm = "realm" EQUAL realm-value
realm-value = quoted-string
domain = "domain" EQUAL LDQUOT URI
*(1*SP URI) RDQUOT
URI = absoluteURI / abs-path
nonce = "nonce" EQUAL nonce-value
nonce-value = quoted-string
opaque = "opaque" EQUAL quoted-string
stale = "stale" EQUAL ("true" / "false")
algorithm = "algorithm" EQUAL ("MD5" / "MD5-sess"

/ token)
qop-options = "qop" EQUAL LDQUOT qop-value
*("," qop-value) RDQUOT
qop-value = "auth" / "auth-int" / token

Proxy-Authorization = "Proxy-Authorization" HCOLON credentials

Proxy-Require = "Proxy-Require" HCOLON option-tag
*(COMMA option-tag)
option-tag = token

Record-Route = "Record-Route" HCOLON rec-route *(COMMA
rec-route)
rec-route = name-addr *(SEMI rr-param)
rr-param = generic-param

Reply-To = "Reply-To" HCOLON rplyto-spec
rplyto-spec = (name-addr / addr-spec)
*(SEMI rplyto-param)
rplyto-param = generic-param

Require = "Require" HCOLON option-tag *(COMMA
option-tag)

Retry-After = "Retry-After" HCOLON delta-seconds
[comment] *(SEMI retry-param)

retry-param = ("duration" EQUAL delta-seconds)
/ generic-param

```
Route                =    "Route" HCOLON route-param *(COMMA
route-param)
route-param          =    name-addr *( SEMI rr-param )

Server               =    "Server" HCOLON server-val *(LWS server-
val)
server-val           =    product / comment
product              =    token [SLASH product-version]
product-version      =    token

Subject              =    ( "Subject" / "s" ) HCOLON [TEXT-UTF8-
TRIM]

Supported            =    ( "Supported" / "k" ) HCOLON
[option-tag *(COMMA option-tag)]

Timestamp            =    "Timestamp" HCOLON 1*(DIGIT)
[ "." *(DIGIT) ] [ LWS delay ]
delay                =    *(DIGIT) [ "." *(DIGIT) ]

To                   =    ( "To" / "t" ) HCOLON ( name-addr
/ addr-spec ) *( SEMI to-param )
to-param             =    tag-param / generic-param

Unsupported          = "Unsupported" HCOLON option-tag *(COMMA
option-tag)
User-Agent           =    "User-Agent" HCOLON server-val *(LWS
server-val)
```

```
Via                 =    ( "Via" / "v" ) HCOLON via-parm *(COMMA
via-parm)
via-parm            =    sent-protocol LWS sent-by *( SEMI via-
params )
via-params          =    via-ttl / via-maddr
/ via-received / via-branch
/ via-extension
via-ttl             =    "ttl" EQUAL ttl
via-maddr           =    "maddr" EQUAL host
via-received        =    "received" EQUAL (IPv4address /
IPv6address)
via-branch          =    "branch" EQUAL token
via-extension       =    generic-param
sent-protocol       =    protocol-name SLASH protocol-version
SLASH transport
protocol-name       =    "SIP" / token
protocol-version    =    token
transport           =    "UDP" / "TCP" / "TLS" / "SCTP"
/ other-transport
sent-by             =    host [ COLON port ]
ttl                 =    1*3DIGIT ; 0 to 255

Warning             = "Warning" HCOLON warning-value *(COMMA
warning-value)
warning-value       =    warn-code SP warn-agent SP warn-text
warn-code           =    3DIGIT
warn-agent          =    hostport / pseudonym
; the name or pseudonym of the server adding
```

```
; the Warning header, for use in debugging
warn-text              =    quoted-string
pseudonym              =    token

WWW-Authenticate =    "WWW-Authenticate" HCOLON challenge

extension-header    =    header-name HCOLON header-value
header-name         =    token
header-value        =    *(TEXT-UTF8char / UTF8-CONT / LWS)
message-body        =    *OCTET
```

26 安全考虑：威胁模式和安全应用建议。

SIP 不是一个容易进行安全保护的协议。它使用的中间媒体，以及它的多面信任关系，它希望的节点之间交互基于互不信任的关系，它的用户到用户的操作使得安全保证非常重要。今天，我们需要找到基于广泛环境和使用方法的很好的安全解决方案。为了达到这一目标，我们需要建立对 SIP 的不同使用的几种安全机制。

注意 SIP 的安全性本身同 SIP 使用的传输协议比如 RTP 的安全性或者和 SIP 包体的实现的安全性本身没有继承关系（虽然 MIME 安全体系是 SIP 安全体系的一个基石）。任何和一个会话相关的媒介都可以被端到端的加密，并且这个和相关的 SIP 信令无关。媒体加密是在本文档讨论范畴之外的。

首先对一系列的经典的威胁模式的分析可以在很大程度上描绘了 SIP 所需要的安全需要。这些威胁模式所针对的地址需要一些安全保护，我们接下来通过对集中安全集中的详细分析，来讲述如何这些安全机制能够提供对这个地址的安全保护。接

着我们就可以定义 SIP 实现者的需求，并且通过提供一个安全配置样例来描述应用于提高 SIP 安全性的这些安全机制。本节也标注了一些隐私相关的注解。

26.1 攻击和威胁模式

本节讲述了对 SIP 部属来说常见的威胁模式。这些威胁模式是经过特别筛选的，用来体现各个 SIP 所需要的安全保卫服务。接下来的例子并不是完整的针对 SIP 的威胁模式列表；不过他们是"经典"的例子，用来代表各类对 SIP 的攻击。

这些攻击假设攻击者可以从网络上读取任何报文－这是因为 SIP 会通常基于公共网络 Internet。在网络上的攻击者通常可以更改报文内容（可能基于中间某个节点来更改）。攻击者可以希望盗取服务，窃听通讯，或者干扰会话。

26.1.1 注册服务 Hijacking。

SIP 注册机制是提供一个用户 UA 把自己的信息到一个注册服务器上，在这个信息中，可以用 address-of-record 找到这个用户的地址。注册服务器会检查在 REGISTER 消息中的 From 头域所提供的身份说明，来决定是否这个请求可以修改由 To 头域所包含的 address-of-record 的相关联系地址。这两个头域通常是相同的，也会有很多第三方代替用户注册联系地址的情况。

SIP 请求的 From 头域，可以被一个 UA 的拥有者任意修改，这就给恶意注册信息打开了方便之门。一个成功模拟一个 UA，通过检查来修改一个 address-of-record 的相关联系地址，可以，例如先注销某个 URI 的所有联系地址，然后把自己的设备地址注册上去，这样所有对原来 URI 的地址的请求将发往攻击者的设备。

这种攻击属于很常见的对没有请求发前方数字签名的攻击。任何有意义的 SIP UAS（比如对传统电话呼叫的 SIP 网关等等），也许希望通过对收到的请求做认

证来控制对自己资源的访问。就算是最终终端 UA，例如 SIP 电话，也会有对原始请求身份的验证要求。这个威胁表明了 SIP 实体需要对原是请求做安全认证的安全服务需要。

26.1.2 模仿一个服务器

对于请求发送到的区域，一般是用 Request-URI 来标志的。UA 通常直接联系这个区域中的服务器来发送请求。不过总会存在一个可能，就是攻击者把自己模仿成为远端的服务器，这样 UA 的请求可能会被其他人中间截获。

例如，我们考虑这样一个情况：一个转发服务器在一个区域:chicago.com，它模拟的转发服务器在另外一个区域: biloxi.com。用户 UA 要发送一个请求到 biloxi.com，但是 chicago.com 的转发服务器回答了一个伪造的应答，并且有伪造的头域就好像应答是从 biloxi.com 回来的一样。在转发应答中的伪造的联系地址可以引导原始 UA 到一个不合适的或者不安全的资源，或者简单的阻止发送请求到 biloxi.com。

这个常见的威胁有着许多的成员，并且相当严重。作为和注册服务 hijacking 相反的威胁，我们考虑这个情况： 当注册服务信息发送给 biloxi.com 的被 chicago.com 截获，并且回应给注册者一个伪造的 301(Moved Permanently) 应答。这个应答可能看起来是从 biloxi.com 来得，并且指明了 chicago.com 作为新的注册服务。那么这个原始 UA 的所有 REGISTER 请求就会转发到 chicago.com 了。

要想防止这个威胁，那么就需要 UA 能够对接收他们请求的服务器进行身份鉴定。

26.1.3 修改消息包体

当然，SIP UA 路由请求通过信任的 proxy 服务器是一件必然的事情。不管这个信任关系是如何建立的（在本节的其他地方有讲 proxy 的认证），UA 可以信任 proxy 来转发请求，而不是检查怀疑请求中的包体被修改了。

考虑 UA 使用 SIP 消息体来进行媒体会话的会话密钥通讯的情况。虽然它信任本域的 proxy 服务器，但是它也不希望域的管理者能够解密后续的媒体通讯（就是不希望 proxy 得到这个会话密钥）。更糟糕的是，如果这个 proxy 服务器是有恶意的，他可以修改这个会话密钥，就像中间人一样，或者改变原始请求 UA 的安全特性。

这个类型的威胁不仅仅是对会话密钥，对所有 SIP 端到端的内容都有威胁。这可能包含对需要展示给用户的 MIME 包体，SDP 或者电话信令等内容的威胁。攻击者可能试图修改 SDP 包体，例如，给 RTP 媒体流增加一个窃听设备来偷听语音通话信息。

同样需要注意的是，有一些 SIP 头域是对端到端有一定含义的，比如 Subject。UA 可以决定保护这些头域和包体（比如中间恶意的攻击者可以把 Subject 头域更改成为看起来好像是一个恶意邮件）。不过，因为很多头域都是 proxy 服务器在处理请求转发的时候需要合法检查或者更改的，所以不是所有的头域都需要端到端的保护。

基于这些原因，UA 可以加密 SIP 包体，并且对端到端的头域做一定的限制。对包体的安全服务要求包含了机密性，完整性和身份认证。这些端到端的安全服务应当与用于和中间节点交互的安全机制无关或者不依赖。

26.1.4 破坏会话

当对话被初始消息所建立，后续的请求可以用于修改对话并且/或者会话的状态。对于会话的负责者来说，非常重要的事情是确定请求不是由攻击者伪造的。

我们考虑这样一个情况，一个第三者攻击者截获了一些初始信息，这些初始信息是对话的双方在建立会话是交换的参数等等（To tag,From tag,等等），并且这个攻击者在会话中插入了一个 BYE 请求。攻击者可以选择假造一个从会话的任意方的请求。当一个 BYE 被对方接收到，会话就会被提前终止。

类似会话中的威胁，有伪造 re-INVITE 修改会话（可能减少会话的安全性，或者作为窃听攻击转发媒体流）。

对于这种威胁，最有效的对策就是对 BYE 的发送方做身份认证。在这个例子中，接受方只需要确认 BYE 是从建立对话的对方发起的就可以了。同样的，如果攻击者不能够知道会话的参数，他也没有办法伪造 BYE。但是，有些中间节点（比如 proxy 服务器）需要这些参数来判定是否会话已经建立连接。

26.1.5 拒绝服务和扩展。

DOS（拒绝服务）攻击主要是使得一个特定网络节点无法工作，通常是通过转发超大量的网络通讯阻塞它的网络接口。分布式的拒绝服务攻击允许一个网络用户导致大量的网络服务器来对一个目标做洪水攻击。

在很多架构下，SIP proxy 服务器是基于公网的，因为它需要处理全球的 IP 终端的请求。这样 SIP 就给这些分布式拒绝服务攻击者提供了很多机会，这样就必须要求 SIP 系统的设计者和管理者能够识别这样的攻击并且定位这样的攻击者。

攻击者可以发出包含假 IP 地址及其相关的 Via 头域的请求，这个 Via 头域标志了被攻击的主机地址，就像这个请求是从这个主机地址来的一样。然后把这个请求发

给大量的 SIP 节点，这样不幸的 SIP UA 或者 proxy 就会给被攻击的主机产生大量的垃圾应答，从而形成拒绝服务攻击。

类似的，攻击者可以用在请求中伪造的 Route 头域值来标志被攻击的目的主机，并且把这个消息发送到分支 proxy，这些分支 proxy 会放大请求数量发送给目标主机。

Record-Route 头域也可以产生类似的效果。当一个攻击者确定一个被请求初始化的 SIP 对话会向回产生很多事务的时候，那么 Record-Route 头域也可以被用于攻击。

如果 REGISTER 请求没有经过注册服务器进行适当的认证，那么就会有很多拒绝服务的攻击的机会。攻击者可以在一个域中，首先把一些或者全部的用户都注销，从而防止这些用户被加入新的会话。接着一个攻击者可以在注册服务器上注册大量的联系地址，这些联系地址都指向同一个被攻击的服务器，这样可以使得这个注册服务器和其他相关的 proxy 服务器对分布式 DOS 攻击进行放大。攻击者也会尝试通过注册大量的垃圾来耗尽注册服务器可能的内存或者硬盘。

多点传送的 SIP 请求可以非常明显的增大拒绝服务攻击的可能性。

这些展示的问题需要定义一个架构来把拒绝服务攻击造成的影响最小化，并且需要在安全机制中对这类攻击特别留意。

26.2 安全机制

从上边讲述的威胁来看，我们得到了 SIP 协议所需要的基本安全服务，他们是：保护消息的隐私性和完整性，保护重现（replay）攻击或者消息欺骗，提供会话参与者的身份认证和隐私保护，保护拒绝服务攻击。SIP 消息中的包体分别要求机密性，完整性和身份认证。

比为 SIP 重新定义新的安全机制更好的是，SIP 可以重用已经存在的 HTTP 或者 SMTP 的安全机制。

全加密的消息提供了最好的机密保护－它也可以保证消息不被恶心的中间节点更改。不过 SIP 请求和应答不能简单的进行端到端的加密，因为在大多数网络架构下，消息的头域比如 Request-URI,Route,Via 在中间经过的 proxy 中需要可见，这样 SIP 请求才能被正确路由。注意，proxy 服务器需要修改一些消息的特定属性（比如增加 Via 头域），这样才能保证 SIP 正常工作。那么 proxy 服务器就必须被 UA 信任，至少在某种程度上信任。为了这个目标，我们建议为 SIP 提供低层次的安全机制，他们是基于节点到节点的整个 SIP 请求和应答的在线加密，并且语序终端节点校验他们发出请求的 proxy 服务器的身份。

SIP 实体也可以为安全保证，需要验证对方的身份。当 SIP 终端向对方 UA 或者 proxy 服务器，声明它的用户的身份时，这个身份应当是可以通过某种方法验证的。SIP 中的数字签名机制就是为了这个需要的。

SIP 消息体的一个独立的安全（加密）机制提供了另一个端到端的相互认证方式，也降低了 UA 必须信任中间节点的程度。

26.2.1 通讯和网络层的安全

通讯或者网络层的安全机制是加密信号通讯，保证消息机密的和完整的传送。

很多情况下，低层次的安全是通过证书实现的，这些证书可以在很多架构下用于提供身份认证使用。

两个通常使用的方法，提供了通讯层和网络层的安全，他们是 TLS[25]和 IPSec[26]。

IPSec 是一组网络层的协议工具，他们可以一起使用来作为传统 IP 通讯的安全替代。IPSec 最常用于一组主机或者管理的域有一个现存的互相信任关系的架构下。Ipsec 通常由主机的操作系统级别实现，或者在一个提供机密通讯和完整性保证的通讯安全网关上实现（比如 VPN 结构），IPSec 也可以用于点到点的结构。

在很多结构下，IPSec 并不要求和 SIP 应用一起使用；IPSec 可能是最适合于部属在那种难于直接在 SIP 服务器上增加安全性的情况。具有预先共享密钥关系的 UA 和他们的第一个节点的 proxy 服务器很适合使用 IPSec。为 SIP 部属的 IPSec 要求一个 IPSec 描述了协议工具的 profile。这个 profile 在本文档中没有提供。

TLS 提供了通讯层的安全性，基于连接相关的协议（TCP）。可以通过在 Via 头域或者在一个 SIP URI 中列明”tls” (表示基于 TCP 的 TLS)指定通讯协议为 TLS。TLS 最适合没有事先定义的信任关系的点到点的结构。例如 Alice 信任她的本地 proxy 服务器，这个服务器在进行证书交换后信任 Bob 的本地 proxy 服务器，这个 Bob 的本地 proxy 服务器是 Bob 信任的，因此 Bob 可以和 Alice 安全的通讯。

TLS 必须和 SIP 应用紧紧联系在一起。注意在 SIP 中的通讯机制是点到点的，因此一个基于 TLS 发送请求到 proxy 服务器的 UA 并不能保证这个 TLS 会在端到端的应用。

当实现者在 SIP 应用中使用 TLS 的时候，实现者必须支持最小集合的 TLS_RSA_WITH_AES_128_CBC_SHA 密码套件[6]。并且为了向后兼容，proxy 服务器，重定向服务器和注册服务器应当支持 TLS_RSA_WITH3DES_EDE_CBC_SHA。实现者也可以支持其他密码套件。

26.2.2 SIPS URI 方案

SIPS URI 方案是 SIP URI（19 节）语法的一个附加，虽然这个方案串是”sips”不同于”sip”。SIPS 的语义和 SIP URI 的语义十分不同。SIPS 允许指定希望通过安全访问的资源。SIPS URI 可以当作一个特定用户的 address-of-record 使用－这个用户是已知的（根据他们的名片，在他盟请求的 From 头域，在 REGISTER 请求的 To 头域）。当在请求中使用 Request-URI，SIPS 方案指出请求经过的每一个节点，知道请求到达目的这个 Request-URI 指明的 SIP 元素，必须通过 TLS 进行加密；当请求抵达目标的域，他会根据目标域的本地安全策略和转发策略，很有可能最后一部也是用 TLS 到达 UAS。当用在请求的发起方（就像这种情况，当他们使用 SIPS URI 当作目标的 address-of-record 一样），SIPS 只是这个实体请求，到目的主机的所有路径都应当加密。

SIPS 方案适用于很多在 SIP 中应用的 SIP URI，比如附加域 Request-Uri,包含在 address-of-record,联系地址（Contact 的内容，包含 REGISTER 方法的头域等等），还有 Route 头域等等。在每个用法中，SIPS URI 方案允许这些存在的 URI 来指明需要安全访问的资源。这些由 SIPS URI 所替换的东西，有他们自己的安全属性（[4]中详细介绍）。

对 SIPS 的使用在细节上要求必须具备 TLS 互相的认证，并且要求支持密码套件 TLS_RSA_WITH_AES_128_CBC_SHA。在认证过程中接收到的信任书应当从客户端持有的信任书跟节点开始验证；对信任书验证失败应当导致请求的失败。

注意在 SIPS URI 方案中，通讯层是和 TLS 没有依赖关系的，并且因此”sips:alice@atlanta.com;transport=tcp”和”sips:alice@atlanta.com;transport=sctp”都是合法的（虽然注意到 UDP 不能用于 SIPS 的传送）。我们不建议使用类似”transport=tls”的方式，部分原因是因为这是用于请求的单个节点之间的通讯。这是从 RFC2543 的一个变化。

将 address-of-record 用 SIPS URI 发出的用户，如果在非可靠通讯协议上收到的请求，可以操作设备来拒绝这个请求。

26.2.3 HTTP Authentication

SIP 提供了认证机制，基于 HTTP 认证的身份认证机制，他们依赖于 401 倒 407 应答码和相关头域来提供拒绝不信任的信任书。对于 SIP 使用的 HTTP Digest 认证机制，并没有做重大的修改，它提供了 replay 攻击的保护和单向认证关系。

对 SIP 的 Digest 认证使用在 22 节有描述。

26.2.4 S/MIME

就像上边讲述的，在端到端的过程中加密整个 SIP 消息体，可以提供机密性的保护，但是并非所有的字段都能使用这个机制进行保护，因为中间的网络节点（比如 proxy 服务器），需要根据读取适当的头域然后决定这个消息应当转发倒哪里，并且如果这些中间节点由于安全原因被排出在外，那么 SIP 消息从本质上就是不能路由的。

不过，S/MIME 允许 SIP 的 UA 在 SIP 中加密 MIME 包体，在不影响消息头的情况下，在端到端的通讯中加密这些 MIME 包体。S/MIME 可以提供消息体的端到端的完整性和机密性，同样也提供了双向的认证机制。使用 S/MIME 也可以通过 SIP 消息隧道，为 SIP 头域提供一个完整性和机密性的方案。

对 SIP 的 S/MIME 使用在 23 节讲述。

26.3 安全机制的实现

26.3.1 对 SIP 实现者的要求

proxy 服务器，重定向服务器，和注册服务器必须实现 TLS，并且必须支持双向的和单向的认证关系。强烈建议 UA 可以初始化 TLS；UA 同样可以作为一个 TLS 服务器。proxy 服务器，重定向服务器和注册服务器应当有一个站点信任书，这个信任书的主题和他们的规范主机名相关。对于 TLS 的双向认证，UA 可以有他们自己的信任书，但是本文档中，没有规定他们的具体用法。所有的支持 TLS 的 SIP 元素必须具备在 TLS 协商中，验证信任书的机制；这个使得证书机关（可能是有名的类似 web 浏览器证书发行机构的发行机构）发布的一个或者多个根信任书成为必须。所有支持 TLS 的 SIP 元素必须同样支持 SIPS URI 方案。

Proxy 服务器，重定向服务器，注册服务器，和 UA 可以实现 IPSec 或者其他底层的安全协议。

当 UA 试图联系一个 proxy 服务器，重定向服务器或者主阿服务器，UAC 应当初始化一个 TLS 连接，在这个连接上发起 SIP 消息。在某些结构吓，UAS 可以同样在这些 TLS 接收请求

Proxy 服务器，重定向服务器，注册服务器，和 UA 必须实现 Digest 身份认证，包括所有的 22 节要求的要点。Proxy 服务器，重定向服务器，注册服务器应当配置成为至少有一个 Digest realm,并且对于给定服务器来说，必须支持至少有一个”realm”字符串和这个服务器的主机名或者 hostname 相关联。

UA 可以支持 MIME 包体的加密，并且通过 23 节描述的那样使用 S/MIME 传送信任书。如果 UA 具有一个或者多个根身份认证的信任书，用来鉴定 TLS 或者 IPSec 的信任书，它应当适当的可以用这些来鉴定 S/MIME 的信任书。UA 可以为 S/MIME 身份认证而具有特定的根信任书。

注意，随着 S/MIME 实现，将来会有安全扩展，来提高 S/MIME 的强度。

26.3.2 安全解决方案

这些安全机制的操作，可以在某种程度上和现存的 WEB 和 EMAIL 安全模式一致。在高一点的级别来看，UA 通过 Digest 用户名和口令把他们自己的身份向服务器（proxy 服务器，重定向服务器，注册服务器）认证；服务器把他们自己向 UA 单节点认证，或者向另外一个服务器进行单节点（one hop）认证（反之亦然），并且是通过 TLS 来传送服务器节点信任书。

在点对点的级别，UA 一般信任网络来进行对方身份的鉴别；不过,如果网络不能够鉴定对方身份，或者网络本身不被信任的情况下，也可以使用 S/MIME 来提供直接的身份认证。

接下来是一个例子，在这个例子中，不同的 UA 和服务器使用这些安全机制防止 26.1 节描述的攻击威胁。实现者和网络管理员可以遵循本节末尾给出的指示来防止攻击威胁，这些指示是作为实现例子提供的。

26.3.2.1 注册

当UA上线，并且注册到它自己的域上，它应当和它的注册服务器建立一个TLS连接（10 节描述了UA怎样找到它的注册服务器）。注册服务器应当提供一个信任书给UA，并且这个信任书的节点必须是这个UA想要注册的域相关的信任书节点；例如，如果UA向注册alice@atlanta.com这个address-of-record，这个信任书节点必须是一个atlanta.com域的主机（比如sip.atlanta.com）。如果它收到了TLS信任书消息，UA应当校验这个信任书，并且检查这个信任书的节点。如果信任书是非法的，作废的，或者它和持有者不符，UA必须不能发送REGISTER消息和进行注册处理。

当 UA 收到注册服务器提供的一个有效的信任书，UA 知道注册服务器并非一个攻击者（可能重定向、窃取口令、或者试图做类似攻击的攻击者）。

于是 UA 创建了一个 REGISTER 请求，并且 Request-URI 应当指向从注册服务器所接收到的信任书站点。当 UA 通过刚才建立的 TLS 连接发送 REGISTER 请求，注册服务器应当给出一个 401（Proxy Authentication Required）应答。在这个应答中，Proxy-Authenticate 头域的”realm”参数，应当和前边给出的信任书节点的域相同。当 UAC 收到这个拒绝，它应当提示给用户要求信任书，或者根据应答中的”realm”参数，从现有的密钥组中查找对应的信任书。这个信任书的用户名应当和 REGISTER 请求的 To 头域的 URI 的”userinfo”部分相关。当在一个合适的 Proxy-Authorization 头域中插入和这个信任书，REGISTER 应当重新发送到注册服务器。

由于注册服务器要求 UA 认证它自己，对于攻击者来说，伪造一个用户的 address-of-record 的 REGISTER 请求是很困难的。同样注意到由于 REGISTER 是通过机密的 TLS 连接发送的，攻击者不能通过截取 REGISTER 来记录信任书来进行重放攻击。

当注册请求被注册服务器接收，UA 应当继续保持 TLS 连接，这样使得注册服务器可以既当作 proxy 服务器，这个 proxy 服务器可以作为管理这个域的 proxy 服务器。刚完成注册的 TLS 连接会继续保留用于接收 UA 后续发起的请求。

由于 UA 已经通过在 TLS 连接的对方的服务器的认证，所有在这个连接上的请求都是经过 proxy 服务器的（由于保留了 TLS 连接，也就是说，刚才的注册服务器更换了角色，变成一个 proxy 服务器）——攻击者不能伪造好像是刚才从这个 proxy 服务器发送的请求。

26.3.2.2 在域之间的请求

现在我们说，Alice的UA希望和远端管理的域的一个用户，这个用户叫做”bob@biloxi.com”,初始化一个会话。我们讲会说本地管理的域（atlanta.com）有一个本地外发proxy。

对于一个管理域的 Proxy 服务器处理那发请求，可以同样作为本地的外发 proxy；基于简单的原则，我们假定这就是 atlanta.com（否则这时候 UA 将要初始化一个新的 TLS 连接到一个独立的服务器）。假定这时候，这个客户端已经完成了注册（参见前边的步骤），当它发出 INVITE 请求邀请另外一个用户的时候，它将重用这个 TLS 连接到本地 proxy 服务器（刚才的注册服务器）。这个 UA 在 INVITE 请求中，将会重用刚才 cache 的信任书，这样可以避免不必要的要求用户输入信任书。

当本地的发外服务器验证了这个 UA 在 INVITE 请求中的信任书，它应当检查 Request-URI 来决定这个消息应当如何路由（参见[4]）。如果这个 Request-URI 的”domainname”部分和一个本地域相关联（atlanta.com）而不是和 biloxi.com 相关，那么 proxy 服务器会向本地服务查询来决定怎样最好的访问到被呼叫的用户。

“alice@atlanta.com”正在尝试联系呼叫”alex@atlanta.com”,本地proxy将会转发这个请求到Alex和它的注册服务器所建立的TLS连接上。由于Alex将会通过它的已经通过认证的连接上收到这个请求，它就确定这个Alice的请求是通过了本地管理域的proxy的身份验证的。

不过，在这个例子中，Request-URI 指向的是一个远程域。在 atlanta.com 的本地外发服务器应当因此而建立一个和在 biloxi.com 的远程 proxy 服务器的 TLS 连接。由于这个 TLS 连接的两端都是服务器，并且都有服务器的信任书，那么应当使用双向的 TLS 身份认证。连接的双方应当验证和检查对方的信任书，把

在信任书中的域名同 SIP 消息中的头域做比较。例如，atlanta.com 这个 proxy 服务器，在这步，应当检查从对方接收到的关于 biloxi.com 域的信任书。当检查完毕，并且 TLS 协商完成，就建立了基于两个 proxy 的安全通道，atlanta.com proxy 于是可以把 INVITE 请求转发给 biloxi.com 了。

在 biloxi.com 的 proxy 服务器应当检查 atlanta.com 的信任书，并且比较信任书提供的域名和 INVITE 请求的 From 头域的”domainname”部分。这个 biloxi proxy 可以执行严格的安全机制，拒绝那些他们被转发的域和本 proxy 所管理的域不匹配的请求（这个不太明白）。

这些安全机制可以用来防止 SIP 和 SMTP ‘open relays’一样经常被用于产生垃圾邮件一样的信息。

这个政策，只是保证了请求声明的来源确实是它自己；他并不允许 biloxi.com 确知如何 atlanta.com 认证的 Alice。只有当 biloxi.com 由其他方法知道 atlanta.com 的身份认证机制，他才可能确知 Alice 如何证明她的身份的。biloxi.com 可以接着使用更严格的方法，禁止来自未确定和 biloxi.com 相同的认证策略的域的请求（就是说所有 biloxi.com 接受的请求，都必须来自 biloxi.com 所知道身份认证方式的域）。

当 INVITE 请求被 biloxi.com 核准，proxy 服务器应当鉴别现存的 TLS 通道，如果存在现存的 TLS 通道，并且是和这个请求中的被叫用户（在这个例子中是 bob@biloxi.com）相关联的。那么这个 INVITE 请求应当通过这个 TLS 通道发送给 Bob。由于通过这个 TLS 连接收到的请求，这个 TLS 连接是刚才已经在 biloxi proxy 上通过了身份认证，Bob 于是知道 From 头域没有被篡改，并且 atlanta.com 已经认证了 Alice，所以就没有必要犹豫是否信任 Alice 的身份。

在他们转发请求钱，两个 proxy 服务器应当在请求中增加 Record-Route 头域，这样所有在这个对话中的后续的请求讲过通过这两个 proxy 服务器。proxy 服务器因此可以在对话生存周期中，继续提供安全服务。如果 proxy 服务器并不在 Record-Route 头域增加他们自己，以后的消息将会直接端到端的在 Alice 和 Bob 中发送，而没有任何安全措施（除非两个端点使用了某种独立的端到端的安全措施，比如 S/MIME）。在这个考虑上，SIP 梯形模式可以提供一个精美的结构来在 proxy 服务器节点之间进行磋商，以提供一个 Alice 和 Bob 之间的有道理的安全通道。

例如，一个攻击这个结构的攻击者将会不能伪造 BYE 请求并且把他插入 Bob 和 Alice 的信令流，因为攻击者无从探听会话的参数，这也是由于通讯的完整性机制保证了 Alice 和 Bob 之间的通讯是机密的完整的。

26.3.2.3 点对点请求

另外，考虑这样一个情况，UA声称carol@chicago.com没有一个本地外发proxy。当Carol希望发送INVITE到bob@biloxi.com,她的UA应当直接初始化一个TLS连接到biloxi proxy（使用附件[4]中描述的机制来检查怎样到达指定的Request-URI）。当她的UA收到一个biloxi proxy发送的信任书，她应当在通过TLS连接发送她的INVITE请求前，正常校验这个信任书。不过，Carol并没有义务相biloxi proxy提供她自己的身份，但是她在INVITE请求的”message/sip”包体中，有一个CMS-detached(分离的)签名。如果Carol在biloxi.com realm有其他信任书的话，就不一定提供这个签名，但是她在biloxi.com上没有正式关系，所以她没有信任书，也就必须提供这个签名。biloxi proxy可以有严格的机制直接把这个请求踢掉，甚至不用麻烦被叫方来验证这个请求，因为在From头域的”domainname”部分，并没有biloxi.com－它可以被当作这个用户是未认证的。

biloxi proxy对于Bob用户有一个政策，就是所有未认证的请求都应当转发到一个适当的地址，对于bob@biloxic.om,就是<sipo:bob@192.0.2.4>。Carol在和biloxi proxy建立的TLS连接上，收到这个转发请求的应答，于是它信任这个转发地址的真实性。

Carol 应当和目标地址建立一个 TCP 连接，并且发送一个新的 INVITE 请求，在这个请求中，Request-URI 包含刚才接收到的联系地址（需要重新计算包体中的签名，因为请求重新构造了）。Bob 从不安全的界面上接收到这个 INVITE 请求，但是这个 UA 是特意预留一个不安全的界面，在这个情况下，认可请求中的 From 头域并且随后把 INVITE 包体中的签名和一个本地 cache 的信任书进行匹配。Bob 用类似的方法应答,把他自己相 Carol 进行认证，这样一个安全的对话就开始了。

某些情况下，在一个域的 NAT 或者防火墙会阻止 UA 之间直接建立 TCP 连接。在这个情况下，如果本地策略许可，proxy 服务器可以隐含的做 UA 之间的请求转发，并且这个转发是基于没有信任关系的（比如不用现存的 TLS 连接，而是通过 TCP 明码的请求转发）

26.3.2.4 DoS 防护

基于这些安全解决方案，为了使得拒绝服务(DoS)攻击造成的影响最小，实现者应当注意如下的指引：

当 SIP proxy 服务器所在的主机是基于公共 internte 做路由的，他应当部署在一个具有防护操作策略的管理域中（比如 block 源路由，过滤 ping 包等等）。TLS 和 IPSec 都可以在管理域的边界的防护主机，一起参与安全系统的构家，来提高安全性。这些防护主机可以防护拒绝服务攻击，确保在管理域中的 SIP 主机不会被大量的消息阻塞。

不管使用什么样的安全措施，对 proxy 服务器的洪水消息攻击可以耗尽 proxy 服务器的资源，并且阻止发送到目的地的正确的请求。对于 proxy 服务器来说，每一个 SIP 事务都会好用一定的资源，对于有状态的服务器来说这个消耗会更大。因此，有状态的 proxy 比无状态的 proxy 更容易受到洪水攻击的影响。

UA 和 proxy 服务器应当用一个 401（Unauthorized）或者 407（Proxy Authentication Required）拒绝有问题的请求，并且对这些有问题的请求不使用正常的应答重发机制，并且对这些未认证的请求把自己当作无状态的服务器使用。如果攻击者使用假的头域值(例如 Via)指向一个第三方的被攻击的服务器，那么对于 401(Unauthorized)或者 407（Proxy Authentication Required）应答的重发可能正中攻击者的下怀。

总得来说，基于 TLS 签名的 proxy 服务器之间的双向认证机制，降低了中间节点潜在的风险，并且减少了可以用作拒绝服务攻击的伪造的请求或者应答。这也同样提高了攻击者利用无知的 SIP 节点进行放大攻击的难度。

26.4 限制

虽然有这些安全机制，在正确应用的时候，可以阻止很多攻击，但是对于网络管理者和开发者来说，也必须明白他们也会有很多限制的地方。

26.4.1 HTTP Digest

在 SIP 中对于 HTTP Digest 一个限制是 Digest 的完整性机制在 SIP 下运作的不是很完美。尤其是，他们提供了对消息中的 Request-URI 和方法的保护，但是并没有对 UA 希望提供加密的所有头域进行了保护。

RFC2617 所提供的回放攻击保护对于 SIP 来说也有一些限制。例如，next-nonce 机制，并不支持通过管道传送的请求。防止回放攻击应当使用 nonce-count 机制。

另一个 HTTP Digest 限制是 realm 的范围。当用户希望把他们自己的身份向一个资源（这个资源和他们有着预先存在的关系）进行认证的时候，Digest 是有用的，它就像一个服务提供者，用户是一个客户端（这是十分常见的场景，因此 Digest 提供了一个十分有用的功能）。与之对应的，TLS 的范围是基于域之间的，或者 multirealm 的，因为信任书通常是全局可验证的，所以 UA 可以在不需要预先存在关系的服务器上进行身份验证。

26.4.2 S/MIME

对于 S/MIME 的一个最大的限制是对终端用户来说，缺少广泛的公共密钥机构。如果使用的是自签名（selfsigned）信任书（或者信任书不能被对话的对方所验证），23.2 节描述的基于 SIP 的密钥交换机制就容易遭受中间人攻击（man-in-the-middle），在这个中间人攻击中，攻击者可以悄悄检查和修改 S/MIME 包体。攻击者需要截取对话中，双方的第一个密钥交换，在请求和应答中移去现有的 CMS-detached 签名，并且插入另外的包含攻击者提供的信任书（但是看起来像是正确地 address-of-record 的信任书）的 CMS-detached 签名。通讯的双方都回以为他们和对方交换了密钥，实际上他们互相有的只是攻击者的公钥。

必须明确注意到攻击者只能攻击双方的第一次的密钥交换－在后续的情况，密钥的变更对于 UA 来说就是不可见的了。同样使得攻击者难以窃听对话双方的以后的对话中（比如一天内的对话，周内的，或者一年的对话）。

SSH 在第一次密钥交换的时候，也同样容易受到这个中间人攻击；但是，虽然众所周知 SSH 不完美，但是它确实提高了连接的安全性。对于 SIP 来说，使用密钥

的指纹可以有些帮助，就像在 SSH 中一样。例如，如果 SIP 的双方建立了一个语音通讯会话，每一个都可以读取对方传送的密钥指纹，并且可以根据这个指纹和原始指纹做比较。这使得中间人更加难以在语音中模拟通话双方的密钥指纹（实际上这个在 Clipper 基于芯片的保密电话中使用）。

在 S/MIME 机制下，如果 UA 在他们的密钥组中持有对方 address-of-record 的信任书，允许 UA 不用导言来发送加密的请求。不过，存在这个样的情况，如果某个设备注册了这个 address-of-record，并且没有持有持有设备当前用户先前使用的信任书。并且它将因此不能正常处理加密的请求，于是可能会导致某些原本可以避免的错误信号。这很像加密的请求被分支的情况。

S/MIME 相关的密钥通常用于和特定用户（一个 address-of-record）关联起来使用,而不是和一个设备（UA）一起使用。当用户在设备之间移动，很难把私钥安全的在 UA 之间传递；一个设备如何获得这些密钥不在本文讨论。

另外，使用 S/MIME 更常见的困难是，它可以导致很大的消息，特别是当采用 23.4 节的 SIP 隧道戒指的时候。基于这个原因，当使用 S/MIME 隧道机制的时候，一定要使用 TCP 通讯协议。

26.4.3 TLS

TLS 最大的问题是，它不能基于 UDP；TLS 要求面向连接的底层通讯协议，对于本文来说，就是 TCP。

对于 TLS 来说，在本地外发 proxy 服务器和/或者主车服务器上和大量 UA，维持很多并发 TLS 长连接，是一件费力的事情。这导致某些容量问题，特别是在使用加强密钥套件的时候更容易出现容量问题。维持冗余的 TLS 长连接，特别是当 UA 独立负责他们的连接，会很耗资源。

TLS 只允许 SIP 实体到他们临近的认证服务器的连接；TLS 提供了严格的节点到节点的安全性（hop-by-hop）。无论 TLS,还是其他本文档规定的安全机制，当不能直接建立 TCP 连接的情况下，都允许客户通过验证自己到 proxy 服务器来到达目的地。

26.4.4 SIPS URI

实际上在请求经过的每一段都使用 TLS，要求了最终的 UAS 必须能够通过 TLS 到达（也许是通过 SIPS URI 作为一个联系地址注册的）。通常这个目的地是用 SIPS 描述的。在很多结构下，使用 TLS 保护一部分请求路径，最后一部确实依赖于其他的安全机制到 UAS。因此 SIPS 不能保证 TLS 是真正的端到端的使用。注意，这是由于许多 UA 不接受进来的 TLS 连接，甚至那些支持 TLS 的 UA 也可能要求要维持一个永久的 TLS 连接（就像前边的 TLS 限制节讲述的一样），目的是为了作为 UAS 从 TLS 上接收请求。

位置服务，对于 SIPS Request-URI 来说，是不要求提供 SIPS 绑定的。虽然位置服务通常由用户注册（10.2.1 节描述）组成，但是对于一个 address-of-record 来说，可以由不同的协议和界面都可以提供联系地址，并且这些工具都可以用来映射 SIPS URI 到适当的 SIP URI。当对绑定信息查询的时候，位置服务返回它的联系地址，而不关心这个是否是从一个 SIPS 的 Request-URI 上收到的请求。如果是转发服务器访问位置服务，那就是说取决于处理转发的 Contact 头域的 SIP 实体来决定联系地址的属性。

如果要求请求经过的全部路径都使用 TLS 通讯，那么对目的域来说稍稍有点麻烦。如果实现困难，也允许在请求传送节点中的基于密码认证的 proxy 服务器，不兼容或者选择一个折中方案来略过 SIPS 的转发机制（在 16.6 节定义的通常转发规则）。但是，恶意的中间节点就有可能把一个基于 SIPS URI 的请求，重新降级成为 SIP URI。

另外，中间节点也可以正常的把一个基于 SIP 的请求更改成基于 SIPS URI 的请求。请求的接受方发现请求的 Request-URI 是基于 SIPS URI 方案的并不能假设原始请求的 Request-URI 也是基于 SIPS 通过中间节点传送的（从客户端往后）。

为了解决这个问题，我们建议请求的收件人,在请求的 Request-URI 包含一个 SIP 或者 SIPS URI 的情况下，检查 To 头域值，看看是否包含一个 SIPS URI（虽然注意到 Request-URI 可以和 To 头域 URI 有相同的 URI 方案，但是他们不相等并不意味者是一个安全隐患）。虽然客户端可以在一个请求中把 Request-URI 和 To 头域做成不一样的，但是当两个字段的 SIPS 不一样的话，那就是可能的安全隐患，并且请求因此应当被接受方拒绝。接受方也可以检查 Via 头域链来双重检查是否 TLS 在整个请求路径中被使用，一直到最近的本地管理域。源 UAC 也可以用 S/MIME 来帮助确保原始格式的 To 头域被端到端的发送。

如果 UAS 有原因来相信 Request-URI 的 URI 方案在通讯中被非法修改，UA 应当提示用户这个潜在的安全隐患。

作为更深远的考量，为了防止底层的攻击，如果 SIP 实体只接受 SIPS 请求，那么可以拒绝在非安全端口的连接。

终端用户应当完全清楚 SIPS 和 SIP URI 的区别，他们可以在应答中手工更改这个 URI 方案。这可以增加或者降低安全性。例如，如果一个攻击者攻陷了一个 DNS 的 cache，插入了一个假的记录集，这个记录集有效删去了一个 proxy 服务器的所有 SIPS 记录，接着经过这个 proxy 服务器的 SIPS 请求就会失效。这时候，一个用户，看见这个 SIPS address-of-record 总是失败，他可以手工改掉 URI 从 SIPS 改成 SIP，并且重试。当然，这也有一些安全机制防止这类事情发生（如果目标 UA 真是有神经病拒绝所有非 SIPS 请求的话）。但是这个限制毫无价值。往好了想，用户也可以把 SIP URI 变成 SIPS URI。

26.5 Privacy(隐私)

SIP 消息经常包含发送者的敏感信息－不只是他们将说些什么，也包括了他们和谁在通讯，以及他们通讯了多久，以及从那里到哪里的通讯等等。很多应用和他们的用户都要求这类隐私信息对于没有必要知道的方面来说都必须保持隐秘。

注意，也有隐私信息也有少数直接泄漏的方式。如果用户或者服务，位于一个容易猜到的地址，比如通过用户的名字或者机构的联系（这是构成 Address-of-record 的最经常的情形），传统保证隐私性的方式是通过一个未列出的"电话号码表"来实现的。在呼叫方发起的会话邀请中，用户位置服务可以提供精确的被叫方的位置从而破坏了隐私；在实现上因此应当更严格，基于每用户的原则，根据不同的呼叫方给出不同的位置信息和可用性的信息。这是一个 SIP 需要解决的完整的问题。

在某些情况下，用户可能希望在通过身份验证时，在头域中隐藏个人信息。这可以应用于不仅是 From 和相关表现请求发起方信息的头域，也应用于 To 头域－他可能不能由快速拨号的 nick-name 转换成为最终的目的地信息，或者一个为扩展的一组目标的标志，当请求被路由的时候，每一个都可以从 Request-URI 上被移去，但是如果和 To 头域初始值相等的时候，不能更改 To 头域。因此，他可能由于隐私的原因创建和 Request-URI 不同的 To 头域。

27 IANA 认证

SIP 应用中的所有的方法名字，头域名字，状态码，和 option tags，都是通过 RFC 中关于 IANA 认证节的说明在 IANA 注册的。

本规范指示IANA创建了 4 个新的子注册项目在：

http://www.iana.org/assignments/sip-parameters: Option

Tags,Warning Codes(warn-codes),Methods和Response Codes,头域的子项也在那里。

27.1 Option Tags

这个规范在 http://www.iana.org/assignments/sip-parameters 建立了 Option Tags 注册项.

Option tags 用于类似 Require,Supported,Proxy-Require,Unsupoprted 这类的头域，用来支持 SIP 的扩展兼容性机制（19.2 节）。Option tag 自身时一个字符串，和特定的 SIP 选项相关（也就是和特定的 SIP 扩展相关）。它为 SIP 终端确定这个选项。Option tags 当被标准的 RFC 轨迹扩展，它就在 IANA 注册了。RFC 的 IANA 认证节必须包含如下内容，这些内容与 RFC 出版号码一起在 IANA 注册表登记。

o option tag 的名字。名字可以是任意长度的，但是应当不要超过 20 个字母。丙子必须是 alphanum(25 节)字符。

o 描述扩展的描述信息

27.2 Warn-Codes

本规范在 http://www.iana.org/assignments/sip-parameters 建立了 Warn-Codes 注册项。并且初始发布的 warn-codes 值在 20.43 节列出。附加的 warn-codes 通过 RFC 出版物注册发表。

对于 warn-codes 表的描述信息如下：

当一个会话描述协议（SDP）（RFC 2327[1]）出现问题导致事务失败的时候，Warning codes 在 SIP 应答信息中提供了对状态码的补充信息。
“warn-code”包含了 3 位数字。第一个数字”3”表示 warning 是 SIP 的警告信息。除非以后另有描述，只有 3xx 警告信息可以被注册。

300 到 329 的警告信息为会话描述保留字错误保留，330 到 339 为会话要求基本网络服务错误保留，370 到 379 是为会话描述要求的 QoS 参数数量错误保留，390 到 399 是无法归类的杂项警告信息。

27.3 头域名

本规范在 http://www.iana.org/assignments/sip-parameters 建立了头域的注册项。

为了注册一个新的头域名，下列信息需要在 RFC 出版物中提供，

o 头域注册的 RFC 号码
o 注册的头域名
o 如果有缩写，头域的缩写版本。

部分常用的头域可以缩写成 1 个字母的简写形式（7.3.3 节）。简写形式只能在 SIP 工作组复查的时候被登记在 RFC 出版物上。

27.4 方法和应答码

本规范在 http://www.iana.org/assignments/sip-parameters 建立了 Method 和 Response-code 的注册项。并且下边列出了他们的初始值。初始的方法表如下：

INVITE [RFC3261]
ACK [RFC3261]
BYE [RFC3261]
CANCEL [RFC3261]
REGISTER [RFC3261]
OPTIONS [RFC3261]
INFO [RFC2976]

应答码的初始值在 21 节列出，分为信息部分(Informational)，成功(Success)，转发(Redirection)，客户端错误(Client-Error)，服务端错误(Server-Error)，全局错误(Global-Failure)几个部分。这个表格有如下格式：

类型（Type，就是 Informational）, Number, Default Reason Phrase [RFC3261]

如下信息需要提供给 RFC 出版机构来注册新的应答码或者方法：

o 应答码和方法需要注册的 RFC 号码
o 应答码号码或者方法名
o 缺省应答码的原因说明

27.5 “message/sip” MIME 类型

本文档注册了”message/sip”MIME 媒体类型，目的是允许 SIP 可以通过 SIP 包体隧道，首要用于端到端的安全保障。这个媒体类型由如下部分组成：

媒体类型名字： message
媒体子类型： sip
要求的参数： none

可选的参数： version/Encoding scheme/Security considerations

version: 这个打包的消息的 SIP-version 号码（比如”2.0”）。如果没有提供，version 的缺省值就是”2.0”。

Encoding scheme: SIP 消息包含一个 8 位的头，在二进制的 MIME 数据对象后边可选。同样的，SIP 消息必须把这个当作一个二进制值。在通常情况下，SIP 消息通过二进制兼容的通讯协议传送，不需要特别的编码方式。

Security considerations: 本参数用于同 23.4 给出的 S/MIME 安全机制一致，请参见下边的例子和说明。

27.6 新 Content-Disposition 参数注册

本文档也注册了 4 个新的 Content-Disposition 头”disposition-types”: altert,icon,session 和 render。作者要求这些值在 IANA 中为 Content-Disposition 登记。

对于”disposition-types”的描述，包括例子和说明，在 20.11 中说明。

在 IANA 注册中的简单描述是：

alter	包体是一个客户定义的铃声用于提醒用户。
icon	包体是一个显示给用户的 icon
render	包体应当显示给用户
session	包体描述了一个通讯会话，例如 RFC2327 SDP 包体

28 同 RFC 2543 的改变

这个 RFC 修订了 RFC 2543。它主要和 RFC2543 向后兼容。这里描述的变更主要修订了 RFC2543 中发现的错误，并且提供了在 RFC2543 中没有提及的相关情节信息。在本文中，协议以更清晰的层次结构描述。

我们把和 RFC2543 的永久改变，分成按功能行为划分这些区别。比如在互操作上不兼容，或者修正了某些情况下的错误操作，以及和 RFC2543 功能行为不同但是不是一个潜在的兼容性问题。以及有数不清楚的澄清，在本文中没有提及。

28.1 主要的功能改变

o 在 UAC 还没有给出应答之前，UAC 希望终止一个会话，它发送 CANCEL 来终止。如果原始 INVITE 依旧给出 2xx 应答，那么 UAC 接着发送 BYE。BYE 只能发给现存的呼叫对话（leg）中（在这个 RFC 中成为对话(dialog)），但是在 RFC2543 中可以在任何时间发送 BYE。

o SIP 的 BNF 改成 RFC2234 兼容的了。

o SIP URL 的 BNF 更改的更加通用，在 user 部分，允许一个很大的字符集。此外，比较规则也简化为首先大小写不敏感，接着详细比较各个参数。最主要的变化是带参数的 URI 和不带这个参数（但是缺省值相等）的 URI 判定是不相等的。

o 删除了隐藏的 Via。这要求发出绝对的信任，由于它依赖于下一个节点来执行这个不明确的操作。作为替代，Via 隐藏可以通过在 proxy 服务器上的本地实现选择完成，于是在这里不在说明。

o 在 RFC2543,CANCEL 和 INVITE 请求是混和的。他们现在分开了。当用户发出一个 INVITE 请求接着 CANCEL 掉，INVITE 事务将正常终结。UAS 应当对原始 INVITE 请求给出一个 487 应答。

o 类似的，CANCEL 和 BYE 事务也是混和的；RFC2543 允许 UAS 在收到 BYE 的时候，不给 INVTE 请求发送应答。本文档不允许这种情况出现。原始的 INVITE 请求需要一个应答。

o 在 RFC2543 中,UA 需要支持只有 UDP 的情况。在这个 RFC 中，UA 应当支持 UDP 和 TCP。

o 在 RFC2543 中，forking 分支 proxy 在收到多个拒绝应答的时候，只向下行元素发出一个拒绝身份认证应答，在这个 RFC 中，proxy 应当搜集所有的拒绝并且把他们放在应答中发送。

o 在 Digest 信任书中，URI 需要被引号引起来；这个在 RFC2617 和 RFC2069 是不一致的。

o SDP 处理被分为独立的规范[13]，并且更全面的描述了正式的通过 SIP 包体隧道进行的 offer/answer 交换过程。作为 SIP 实现基线，SIP 允许在 INVITE/200 或者 200/ACK 中存在；RFC2543 间接指出了在 INVITE,200 和 ACK 中的单个事务使用这个方法，但是这个在 RFC2543 中没有很好的定义。在 SIP 扩展中允许使用更复杂的 SDP。

o 增加了对 URI 中和在 Via 头域中的 IPV6 的全面支持。Via 头域允许方括号和冒号的参数，要求对 Via 头域的 IPV6 的支持。这些字符在先前是不允许的。在理论上，这颗一导致和旧规范实现上的相互操作的问题。不过，我们观察到大部分实现在这个参数上，接受非控制 ASCII 字符。

o DNS SRV 处理步骤现在使用了单独的规范[4]。这个步骤使用了 SRV 和 NAPTR 资源记录，并且不在合并从 SRV 记录的数据（RFC2543）。

o 循环检测变成可选的了，替代为强制使用 Max-Forwards 参数。这个循环检测在 RFC2543 由严重 bug，可能会在正常情况下把”螺旋”报告成为一个错误。在这里，可选的循环检测步骤是更加全面的和正确的。

o 由于他们现在在对话中是作为识别对话的基本要素存在，所以，对 tag 的使用是强制的（他们在 RFC2543 中是可选的）。

o 增加了 Supported 头域，允许客户端向服务器表示它支持什么样的扩展，服务器可以根据这个字段决定给出包含什么样扩展的应答，并且把他们需要的扩展支持放在 Require 字段里边。

o 几个头域的扩展参数的 BNF 描述在 RFC2543 中忽略了，这里增加了。

o 处理 Route 和 Record-Route 的构在在 RFC2543 中是很不规范的，并且也不正确。它在本规范中更改成为规范正确的（并且大大简化了）。这是很大的改变。对于没有使用”预先加载路由的”部属情况下（在预先加载路由的时候，初始的请求

有一个 Route 路由集合，这个集合不是由 Record-Route 构造的），依旧提供了向后兼容性。在预先加载路由的情况下，新旧机制是不能互相操作的。

o 在 RFC2543 中，消息中的行是可以由 CR/LF/或者 CRLF 终端的。本规范只允许 CRLF。

o 在 CANCLE 和 ACK 请求中的 Route 的使用，在 RFC2543 中没有精确定义，在本文档中定义了；如果请求有一个 Route 头域，他的为一个给请求的非 2xx 应答的 CANCEL 或者 ACK 应当包含相同的 Route 头域。为 2xx 应答的 ACK 请求使用 2xx 应答的 Record-Route 构造 Route 头域值。

o RFC 2543 允许一个 UDP 包中多个请求。这个使用方法在本文档中删去了。

o 对于 Expires 头域和参数中的绝对时间的使用方式被删去了。当各个节点之间时间不同步的时候，他会带来户操作的问题。本文档使用相对时间。

o Via 头域中的 branch 参数现在是强制每一个元素都使用。它现在作为事务的唯一标志。这避免了 RFC2543 容易出错的事务鉴别规则的复杂性。我们在这个参数值使用一个乱数 cookie 来检查上一个节点产生的这个参数是否全局唯一的，如果不是，那么旧采用旧的比较规则。因此，保证了互操作性。

o 在 RFC2543，对 TCP 连接的关闭和 CANCEL 是相同的。当 TCP 连接在两个 proxy 之间的时候，这在实现上是几乎不可能的（也是错误的）。这个要求在这里被删去了，所以，TCP 连接状态和 SIP 处理之间没有直接关系。

o RFC2543 中，当 UA 到一个节点可以初始化一个新的事务，当对方正在进行事务处理的时候。在这里被精确定义了。只允许初始化非 INVITE 请求，不允许 INVITE 请求。

o PGP 被删去了。它没有很充分的定义，并且和更复杂的 PGP MIME 不兼容。替换成为 S/MIME。

o 增加了”sips” URI 方案用于端到端的 TLS 通讯。这个方案是和 RFC2543 不能年个兼容的。现有的节点如果接收到请求的 Request-URI 中包含 SIPS URI 方案，将拒绝这个请求。这实际上是一个特性；它保证了对 SIPS URI 只会在所有节点都是安全的情况下被传送。

o 在 TLS 上附加了安全特性，并且这里也更广泛更完整的描述了安全考虑。

o 在 RFC 2543 中，并不要求 proxy 转发 101 到 199 的临时应答到上行队列。这里被更改成为必须转发。这是非常重要的，因为很多后续的特性都依赖于传送所有的 101 到 199 的临时应答。

o RFC 2543 提及了很少的 503 应答。它用于标志 proxy 失败或者过载的情况。这要求某些特别的处理。尤其是，接收到 503 的接受方应当触发一个对于下一个节点在 DNS SRV 的重新查找。同样的，503 应答只由 proxy 服务器在特定情况下转发到上行队列。

o RFC2543 定义了，但是没有充分指出，对于 UA 认证一个服务器的机制。这个已经删去了。作为替代，允许使用 RFC2617 的双向认证步骤。

o 对于 UA 来说，除非它收到了初始请求的 ACK，不能发送 BYE 到一个呼叫。这个在 RFC2543 是允许的，但是可能导致潜在的空转可能。

o UA 或者 proxy 不能发送 CANCEL 到一个事务，直到它得到了一个临时应答。这个在 RFC2543 中是允许的，但是可能导致空转的可能。

o 在注册中的 action 参数被禁止使用了。它不足以提供任何有用的服务，并且会导致在 proxy 上的应用程序处理上的冲突。

o RFC2543 对于 multicast 有一堆特别的情况。例如某个应答被禁止了，定时器调整了，等等。multicast 现在受到更多限制，并且同 unicast 相反，协议操作不影响对 multicast 的使用。

o 基本的认证整个被删除了，不允许用基本的认证。

o proxy 不再立刻收到就转发 6xx 应答。他们立刻 CANCEL 相关等待的分支。这也避免了潜在的空转，使得 UAC 再 2xx 之后收到一个 6xx。在除了这个空转的情况，结果都必须是相同的－6xx 转发到上行队列。

o RFC2543 并不对请求的合并认为是一个错误。这就导致在 proxy 上分支的请求稍后会在一个节点合并。只有在 UA 上能进行合并操作，并且处理步骤定义的是除了第一个请求都统统拒绝。

28.2 小功能性的变更

o 为给用户展示可选的内容，增加了 Alert-Info,Error-Info,Call-Info 头域。

o 增加了 Content-Language,Content-Disposition 和 MIME-Verison 头域。

o 增加了一个”glare hanling”机制来处理双方同时都向对方提出一个 re-INVITE。它使用信的 491（Request Pending）错误码。

o 增加了 In-Reply_To 和 Reply-To 头域来支持稍后对未接呼叫/消息的返回。

o 在 SIP 通讯协议中增加了 TLS 和 SCTP。

o 描写了很多机制来处理呼叫中的发生的错误；现在做了统一的处理。BYE 用来发送会话终结。

o RFC2543 要求 INVITE 应答通过 TCP 重新传送，但是注意到实际上只对 2xx 应答是需要的。这就导致了人为的协议不足。通过定义了一个一致的事务层，这就不在需要了。只有给 INVITE 的 2xx 应答需要基于 TCP 重传。

o 客户端和服务端事务机器现在基于超时机制，而不是基于重传次数。这允许状态机能够更好的适应 TCP 和 UDP。

o Date 头域在 REGISTER 应答中使用，提供一个简单的自动配置 UA 的日期的机制。

o 允许注册服务器拒绝过小的超时时间的注册。并且为此定义了 423 应答码和 Min-Expires 头域。

29 标准索引

[1] Handley, M. and V. Jacobson, "SDP: Session Description Protocol", RFC 2327, April 1998.

[2] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[3] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[4] Rosenberg, J. and H. Schulzrinne, "SIP: Locating SIP Servers",RFC 3263, June 2002.
[5] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.
[6] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.
[7] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.
[8] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L.,Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
[9] Vaha-Sipila, A., "URLs for Telephone Calls", RFC 2806, April 2000.
[10] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.
[11] Freed, F. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.
[12] Eastlake, D., Crocker, S. and J. Schiller, "Randomness Recommendations for Security", RFC 1750, December 1994.
[13] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with SDP", RFC 3264, June 2002.
[14] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.
[15] Postel, J., "DoD Standard Transmission Control Protocol", RFC 761, January 1980.
[16] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer,H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson,"Stream Control Transmission Protocol", RFC 2960, October 2000.

[17] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S.,Leach, P., Luotonen, A. and L. Stewart, "HTTP authentication:Basic and Digest Access Authentication", RFC 2617, June 1999.

[18] Troost, R., Dorner, S. and K. Moore, "Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field", RFC 2183, August 1997.

[19] Zimmerer, E., Peterson, J., Vemuri, A., Ong, L., Audet, F.,Watson, M. and M. Zonoun, "MIME media types for ISUP and QSIG Objects", RFC 3204, December 2001.

[20] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[21] Alvestrand, H., "IETF Policy on Character Sets and Languages",BCP 18, RFC 2277, January 1998.

[22] Galvin, J., Murphy, S., Crocker, S. and N. Freed, "Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted", RFC 1847, October 1995.

[23] Housley, R., "Cryptographic Message Syntax", RFC 2630, June 1999.

[24] Ramsdell B., "S/MIME Version 3 Message Specification", RFC 2633, June 1999.

[25] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[26] Kent, S. and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.30

30 信息索引：

[27] R. Pandya, "Emerging mobile and personal communication systems," IEEE Communications Magazine, Vol. 33, pp. 44--52, June 1995.

[28] Schulzrinne, H., Casner, S., Frederick, R. and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", RFC 1889, January 1996.

[29] Schulzrinne, H., Rao, R. and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[30] Cuervo, F., Greene, N., Rayhan, A., Huitema, C., Rosen, B. and J. Segers, "Megaco Protocol Version 1.0", RFC 3015, November 2000.

[31] Handley, M., Schulzrinne, H., Schooler, E. and J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, March 1999.

[32] Hoffman, P., Masinter, L. and J. Zawinski, "The mailto URL scheme", RFC 2368, July 1998.

[33] E. M. Schooler, "A multicast user directory service for synchronous rendezvous," Master's Thesis CS-TR-96-18, Department of Computer Science, California Institute of Technology, Pasadena, California, Aug. 1996.

[34] Donovan, S., "The SIP INFO Method", RFC 2976, October 2000.

[35] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[36] Dawson, F. and T. Howes, "vCard MIME Directory Profile", RFC 2426, September 1998.

[37] Good, G., "The LDAP Data Interchange Format (LDIF) – Technical Specification", RFC 2849, June 2000.

[38] Palme, J., "Common Internet Message Headers", RFC 2076, February 1997.
[39] Franks, J., Hallam-Baker, P., Hostetler, J., Leach, P., Luotonen, A., Sink, E. and L. Stewart, "An Extension to HTTP: Digest Access Authentication", RFC 2069, January 1997.
[40] Johnston, A., Donovan, S., Sparks, R., Cunningham, C., Willis, D., Rosenberg, J., Summers, K. and H. Schulzrinne, "SIP Call Flow Examples", Work in Progress.
[41] E. M. Schooler, "Case study: multimedia conference control in a packet-switched teleconferencing system," Journal of Internetworking: Research and Experience, Vol. 4, pp. 99--120, June 1993. ISI reprint series ISI/RS-93-359.
[42] H. Schulzrinne, "Personal mobility for multimedia services inthe Internet," in European Workshop on Interactive Distributed Multimedia Systems and Services (IDMS), (Berlin, Germany), Mar. 1996.
[43] Floyd, S., "Congestion Control Principles", RFC 2914, September 2000.

定时器值的表格：

表 4：本规范使用的定时器意义机器缺省值：

定时器	值	章节	意义
T1	500 ms 缺省	17.1.1.1	预估的 RTT 时间
T2	4s	17.1.2.2	给非 INVITE 请求和 INVITE 应答

			的最大重传时间间隔
T4	5s	17.1.2.2	最大网络传送消息时间
定时器 A	初始值 T1	17.1.1.2	INVITE 请求重传间隔,UDP only
定时器 B	64×T1	17.1.1.2	INVITE 请求超时时间
定时器 C	>3 分钟	16.6/11	proxyINVITE 请求超时时间
定时器 D	>32 秒 UDP 0 TCP/SCTP	17.1.1.2	应答重发的等待时间
定时器 E	初始值 T1	17.1.2.2	非 INVITE 请求重传间隔，UDP only
定时器 F	64×T1	17.1.2.2	非 INVITE 请求事务超时时间
定时器 G	初始值 T1	17.2.1	INVITE 应答重传间隔
定时器 H	64×T1	17.2.1	等待 ACK 的时间
定时器 I	T4 UDP 0 TCP/SCTP	17.2.1	ACK 重传的等待时间
定时器 J	64×T1 UDP 0 TCP/SCTP	17.2.2	非 INVITE 请求重传的等待时间
定时器 K	T4 UDP 0 TCP/SCP	17.1.2.2	应答重传的等待时间

感谢书

我们感谢 IETF MMUSIC 和 SIP WGs 的意见和建议。Ofir Arkin,Brian Bidulock,Jim Buller,Neil Deason,Dave Devanathan,Keith Drage,Bill Fenner, Cedric Fluckiger, Yaron Goland, John Hearty, Bernie Hoeneisen, Jo Hornsby, Phil Hoffer, Christian Huitema, Hisham Khartabil, Jean Jervis, Gadi Karmi, Peter Kjellerstedt, Anders Kristensen, Joanthan Lennox, Gethin Liddell, Allison Mankin, William Marshall, Rohan Mahy, Keith Moore, Vern Paxson, Bob Penfield, Moshe J. Sambol, Chip Sharp, igor Slepchin, Eric Tremblay, Rick Workman 提供了详细的注释

Brian Rosen 提供了完整的 BNF

Jean Mahoney 提供了技术写作协助

这个工作是基于 inter alia [41,42]

作者地址

这里作者的地址是按照字母顺序的，首先是作者，接着是 RFC2543 原著。所有列出的作者都对本文有着巨大贡献：

Jonathan Rosenberg
dynamicsoft

72 Eagle Rock Ave
East Hanover, NJ 07936
USA
EMail: jdrosen@dynamicsoft.com

Henning Schulzrinne
Dept. of Computer Science
Columbia University
1214 Amsterdam Avenue
New York, NY 10027
USA
EMail: schulzrinne@cs.columbia.edu

Gonzalo Camarillo
Ericsson
Advanced Signalling Research Lab.
FIN-02420 Jorvas
Finland
EMail: Gonzalo.Camarillo@ericsson.com

Alan Johnston
WorldCom
100 South 4th Street
St. Louis, MO 63102
USA
EMail: alan.johnston@wcom.com

Jon Peterson
NeuStar, Inc

1800 Sutter Street, Suite 570
Concord, CA 94520
USA
EMail: jon.peterson@neustar.com

Robert Sparks
dynamicsoft, Inc.
5100 Tennyson Parkway
Suite 1200
Plano, Texas 75024
USA
EMail: rsparks@dynamicsoft.com

Mark Handley
International Computer Science Institute
1947 Center St, Suite 600
Berkeley, CA 94704
USA
EMail: mjh@icir.org

Eve Schooler
AT&T Labs-Research
75 Willow Road
Menlo Park, CA 94025
USA
EMail: schooler@research.att.com

版权声明

Copyright (C) The Internet Society (2002). All Rights Reserved.
This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.
The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.
This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING
TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING
BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION
HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF
MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.

译者：崮山路上走 9 遍 2004-9-23 于深圳完稿。
BLOG: sharp838.mblogger.cn
EMAIL: sharp838@21cn.com；guangweishi@gmail.com

所有的版权归于原作者。

感谢：黄小东，朱朱，洋洋，sophia。

---

[1] Implementations that send requests
containing multipart message bodies MUST send a session description as a non-multipart message body if the remote implementation requests
this through an Accept header field that does not contain multipart.

[2] 原文是说and transport.以前的想法，好像transport就是通讯协议，通讯层的意思。但是翻译到现在忽然想到当然是与通讯传输器相关了。因为通讯层可能是可以同时操作多个通讯的，所以每个通讯都可以叫做一个通讯收发员，也就是通讯实例。

Trackback:
http://blog.csdn.net/zysee/archive/2007/01/16/1484278.aspx