#### 1、SIP协议介绍

许多网络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。SIP（会话初始协议）和这些协议一样，允许Internet端点（用户代理）来寻找参与者并且建立一个可共享的会话。为了能够精确的定位会话参与者，SIP创建基础的network hosts（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支持名字映射和重定向服务，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：

1. 用户定位： 检查终端用户的位置，用于通讯。

2. 用户有效性：检查用户参与会话的意愿程度。

3. 用户能力：检查媒体和媒体的参数。

4. 建立会话：”ringing”,建立会话参数在呼叫方和被叫方。

5. 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则那样管理。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、实施概览

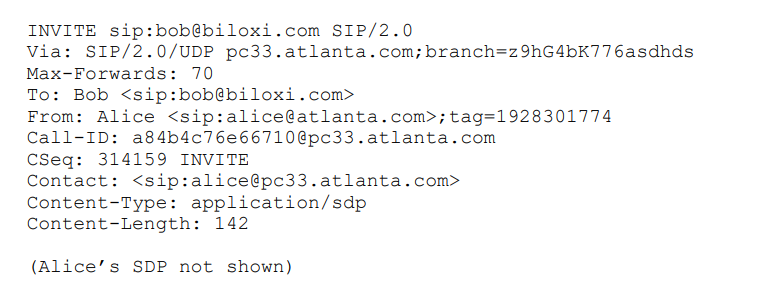
这节通过简单的示例介绍了SIP的基本实现。本节是通过自然的而非正则的示例来介绍的。

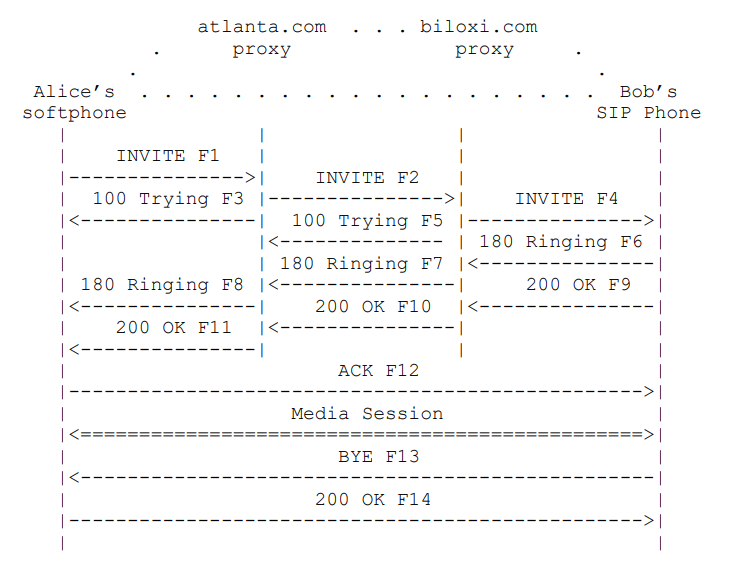
第一个例子说明了SIP的基本功能：定位一个断点，发出通讯请求，通过协商会话参数建立会话，拆卸刚才建立的会话。

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（比如说一个软件的电话），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：





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。

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）。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地址的值（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：

 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。Contact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。

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）。登记服务器记录这个映射，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，但是提供不同的服务。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）。

#### 4、协议的结构

SIP是一个分层的协议，意思是说SIP协议由一组无关的处理层次组成，这些层次之间只有松散的关系。协议分成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。当我们说一个元素”包含”某一个层，我们的意思是这个元素符合这个层定义的规则。不是SIP每一个元素都一定包含每一个层。此外，SIP定义的元素是逻辑上的元素，不是物理元素。一个物理的实现可以实现不同的逻辑元素，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)。

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要素都包含一个传输层。

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。当一个事务用户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。事务用户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，并且回滚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。

SIP元素，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TU。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，以及如何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，他们之间用于交流的媒体流组成。

#### 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：数据头。头是在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，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，并且有需要的话，在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 recurses 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。典型得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 sessino)。在定义中，一个被叫方可以被多次邀请，被不同得呼叫方邀请，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的事件，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，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 core ，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。

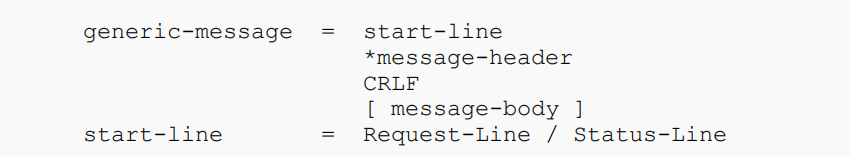
UAS Core：UAS核心。在transaction和transport层智商的UAS实现的功能集合。

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，请求还是应答消息都基于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field，一个可选的消息体message body组成。

起始行、每一个包头行，空行、都必须由回车换行组成（CRLF）。即使消息体没有，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）。

**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。

###### **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-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。

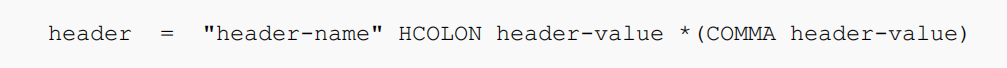
**4xx**：客户端错误－－请求包含错误的格式或者不能在这个服务器上完成。

**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。

###### 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：

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。

###### 7.4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于请求的方法（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(Message Body Type)

消息中的网络媒体类别必须在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”multipart” MIME类型可以在消息体中应用。在由多个部分组成的消息体发送的时候，如果接受方的包头域的Accept域中，不包含多部分”multipart”的标记，那么发送方必须发送一个非多部分session description。 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（TCP）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。UAC可以由一些外部的动作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的机制来产生应答。

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的处理有两个特点。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）。会话的详细描述在第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 URI(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。

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>

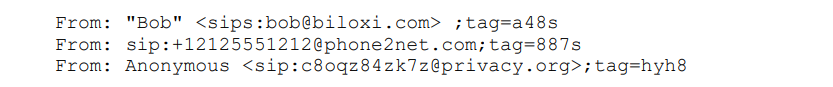


**8.1.1.3 From**

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(客户端)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：





**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）。（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头域是用来区分事务和排列事务的顺序的。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，就可以为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

我们就开始计算请求发送的目标。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，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）。

这样的过程产生了一系列有序的地址，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资源，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个额外的目的地用于发送。如果Request-URI包含一个SIPS URI,任何其他的目的地都必须用TLS来表达。除此之外，如果请求没有包含Route head头域，那么就没有对其他的目的地的限制了。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层处理，并且被transport层发送给上一层transaction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。

目前使用Allow



8.2.2 包头判断

如果UAS（SIP服务器）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core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（客户端）用来通知UAS（服务端）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（服务端）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服务器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（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（服务器）希望应用一部分SIP扩展，但是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（需要扩展支持）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。

在本节没有描述根据响应代码的不同的响应代码的其他的行为。当创建应答的步骤完成之后，UAS（服务器）将响应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(服务器)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(服务器)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(TU)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(重定向)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，后续客户端通过该地址进行访问。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(一小时)。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(客户端、服务器)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，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（客户端）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。

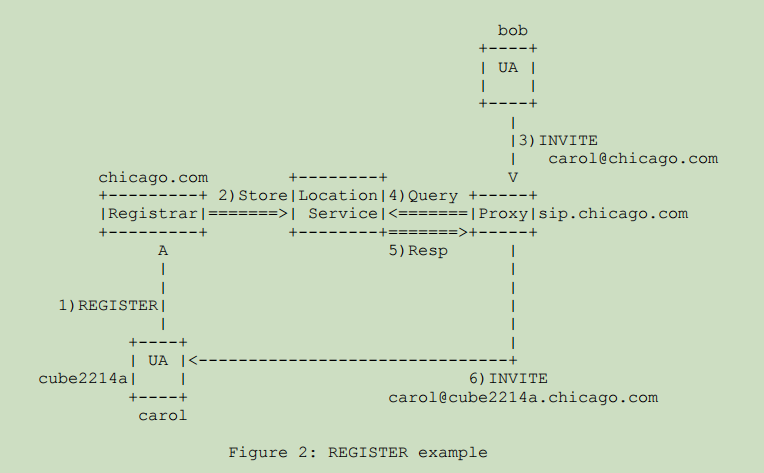
**Cseq**:Cseq值保证了REGISTER请求的正确顺序。一个UA为每一个具备相同的Call-ID的REGISTER请求顺序递增这个Cseq字段。

**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。(可选)

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）。

**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。

 下边的Contact头域参数在REGISTER请求中有特别的意义：



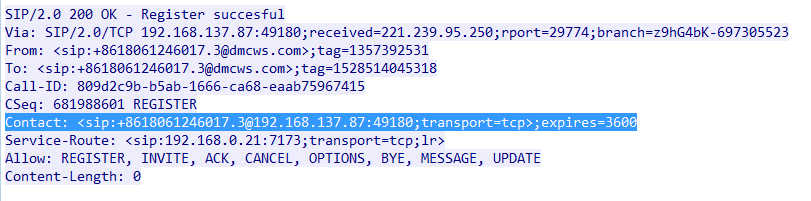
10.2.1 增加绑定

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，这说明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，一个登记服务器(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。如果符合，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。

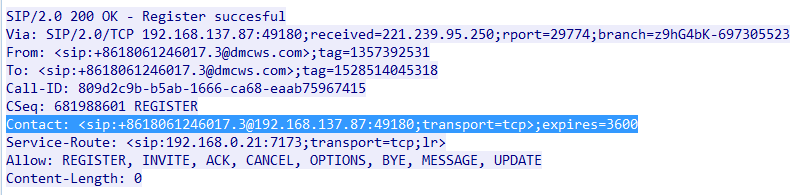
对于每一个地址，注册服务器在当前绑定列表中用URI比较规则进行搜索。如果绑定不存在，它会暂时性的添加进去。如果绑定存在，注册服务器检查Call-ID值。如果Call-ID值在这个已经存在的绑定中和本次请求的Call-ID值不一样(表示是新的设备)，如果绑定的有效期为0就删除这个绑定，否则就刷新这个绑定。如果Call-ID值一样(同一个设备)，那么注册服务器比较Cseq值，如果请求中的这个值比现存的绑定值中的Cseq高，那么必须更新或者删除这个绑定（如果有效期为0就删除，否则就刷新）。如果这个Cseq值比现存的Cseq值小(是之前的请求)，那么必须终止更新并且请求失败。

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。

(register 的原子性)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。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：

###### 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。

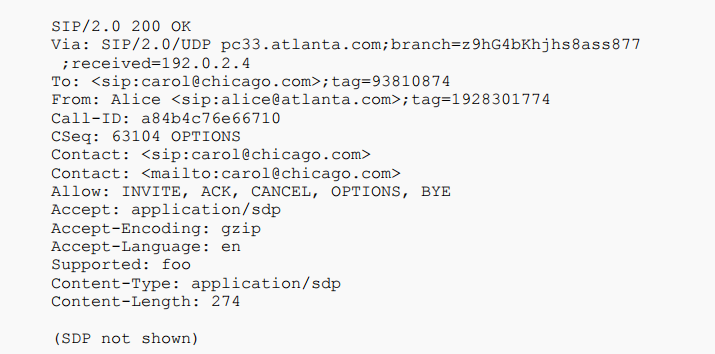
这个OPTIONS的使用有一定的限制，因为对于proxy处理OPTIONS和INVITE请求的不同。一个分支的INVITE可以有多个200（OK）的应答返回，但是一个分支的OPTIONS只能有单个200（OK）应答返回。因为这是由于proxy处理OPTIONS请求是当作非INVITE的处理。参见16.7节有详细的说明。

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）



#### 12 对话

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客户端行为

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客户端行为

12.2.1.1 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，在对话中的请求中去除这种强制性。

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。

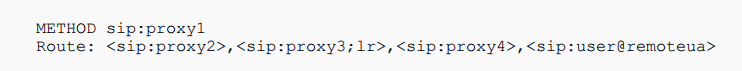
UAC使用remote target和路由集合来构造请求中的Request-URI和Route头域。如果路由集合是空的，那么UAC必须把remote target URI放到Request-URI中，并且UAC不能添加Route头域到请求中。

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：

那么请求应该有下列的Request-URI和Route头域

 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服务端行为

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。

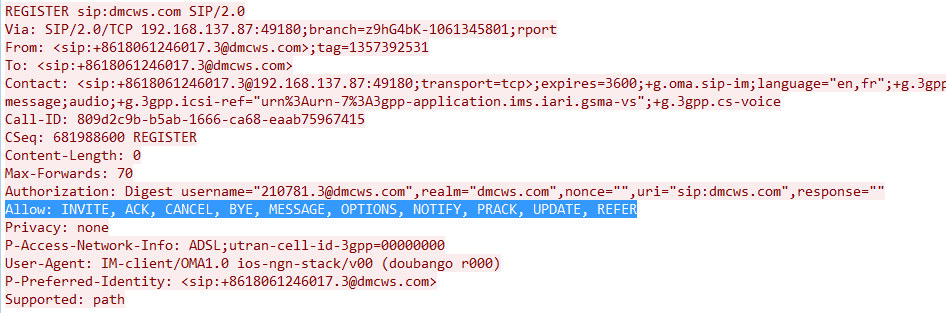
本节介绍了INVITE请求建立会话的详细过程。支持INVITE的UA(客户端、服务器)也一定同时支持ACK,CANCEL和BYE。

###### 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方法（比如audio，vidio，game），这些方法的参数（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方式，这些方法的参数。这个请求/应答的交换是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临时应答中存在。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SDP。

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。请求是在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，只适用于在包头域Content-Disposition中的值是”session”的情况。因此，有可能INVITE和ACK请求中都包含一个包体信息（比如，INVITE包含一个照片（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，但是在包体中的Content-Type是application/sdp的时候，等同于Content-Disposition”session”，其他的情况Content-Type就是”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(事务用户)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参数，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(Route set)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客户端行为

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服务器在发送第一个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

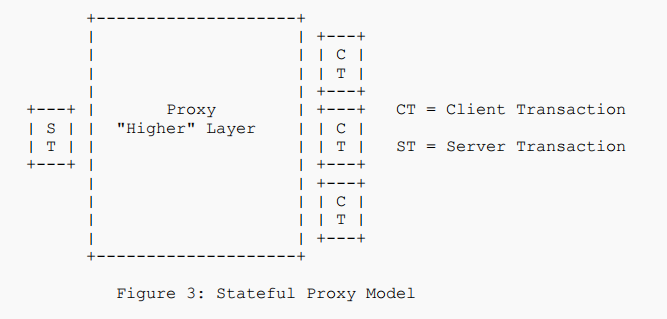
作为有状态的proxy，它必须是一个纯粹的SIP事务处理引擎。它在这里的定义遵循17节讲述的服务端和客户端的事务处理规定。有状态的proxy有一个服务端事务，这个事务与一个或者多个客户端事务相关，这些客户端事务是由高层proxy处理元素产生的（图3），这些高层proxy处理元素就是proxy处理核心。一个输入的请求是通过一个服务端事务来处理的。请求由服务端事务交给proxy处理核心进行处理。proxy处理核心检查请求应当路由到哪里，选择一个或者多个下一个节点。每一个发往下一个节点的外发请求都由客户端事务进行处理。proxy处理核心从客户端事务中得到请求的应答并且把他们的应答交给服务端事务进行发送。

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)
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。

1. 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）。

1. 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。

1. 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。

1. 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。

1. 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

小虎 2006-05-25 00:08

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：

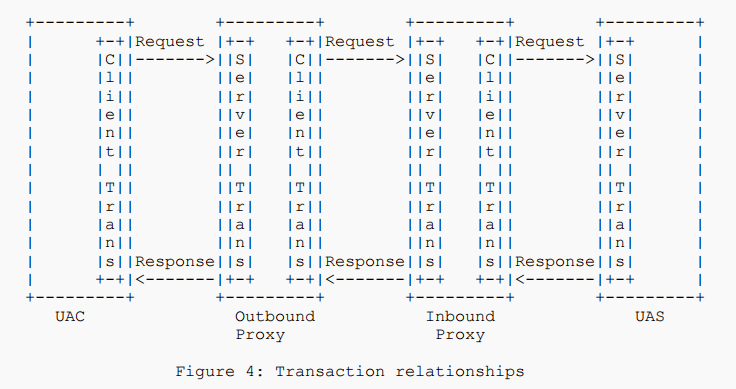
BYE sip:callee@u2.rightpriatespace.com SIP/2.0

#### 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肚子负责挨个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：



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。

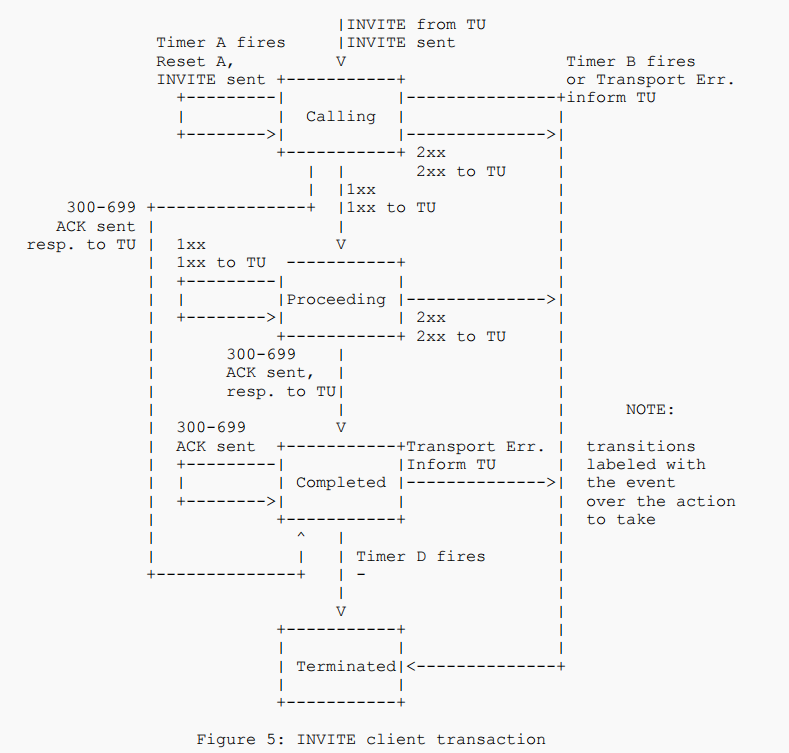
缺省的T1是500ms。T1是一个RTT的估计时间，是在客户端和服务端的一个事务处理的估计时间。节点可以（不推荐）使用更小的T1值，比如私有网络，并不接到INTERNET的网络可以设置小一点。T1也可以设置成为大一点的值，并且我们建议如果当我们知道RTT值比较大的时候（比如高延时的网络）应当设置T1成为大一点的值。不管T1如何取值，本节要求的重传机制要求的指数延时是必须使用的。

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。

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



给非2xx终结应答的ACK请求应当是：



###### 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。

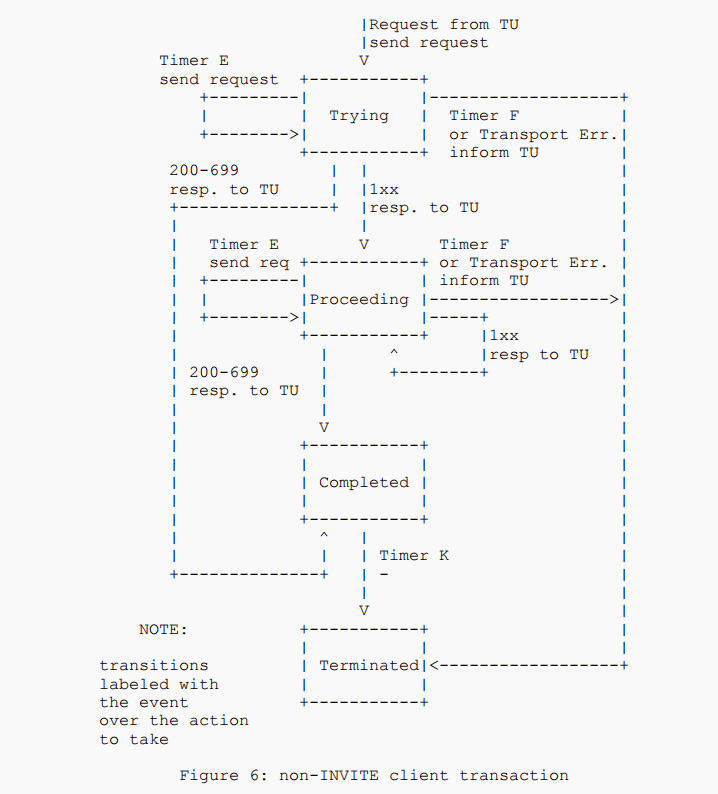
###### 17.1.3 客户端事务匹配应答

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

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。

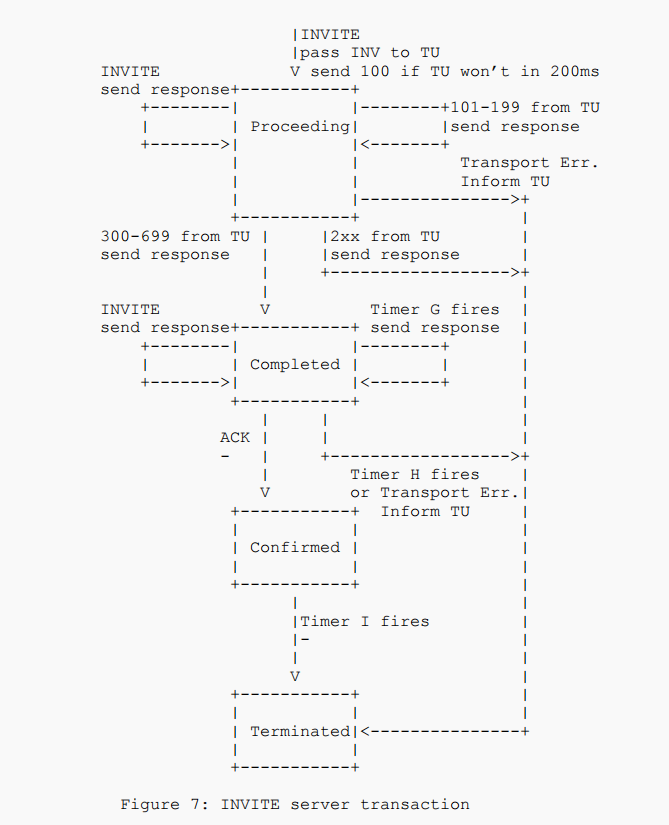
当在”Proceeding”状态的时候，如果TU交给服务端事务一个300到699的应答，那么应答必须交给通讯层进行发送，并且状态机必须进入”Completed”状态。对于非可靠传输的情况，必须设置定时器G=T1秒，对于可靠传输的情况，不设置定时器G（=0的情况就是不设置）

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。

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：

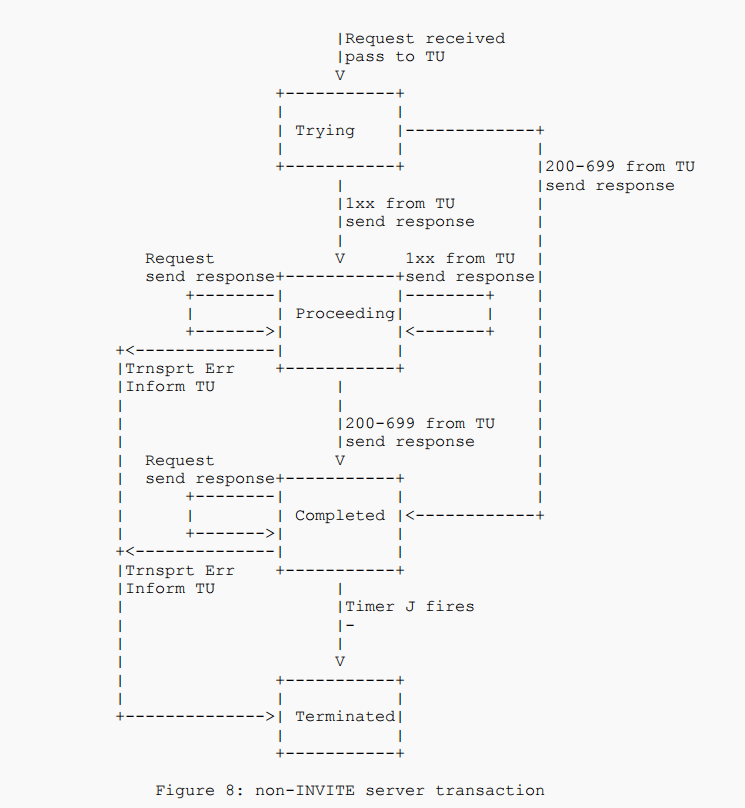
1. 请求中的最上的Via头域的branch参数和创建本事务的请求的最上的Via头域的branch参数一样，并且：
2. 请求的最上的Via头域的sent－by参数和创建本事务的请求的最上的Via头域的send-by参数一样，并且：
3. 请求的方法和创建本事务的方法一样。这有一个例外，就是ACK,ACK对应的创建本事务的请求方法是INVITE。

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。

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 。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 URI中有@，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

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。

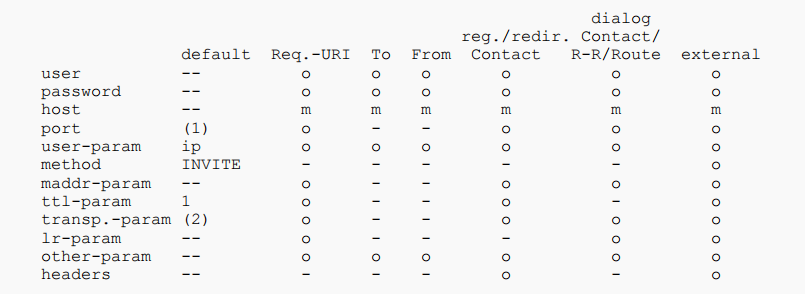
Headers：头域是从给定URI创造的请求的头域部分。

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）



(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。

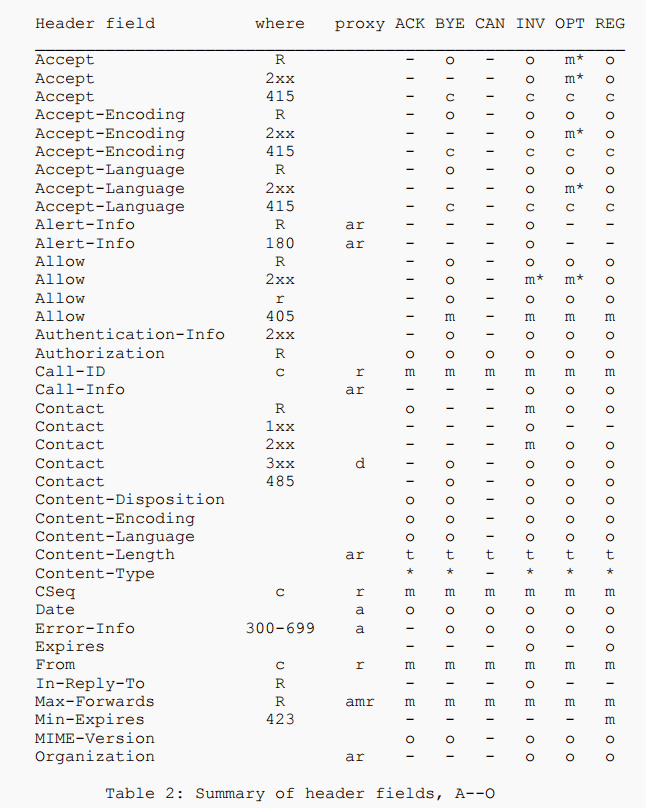
本规范也定义了常用的头域名的缩写，用于缩小消息的大小。

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。



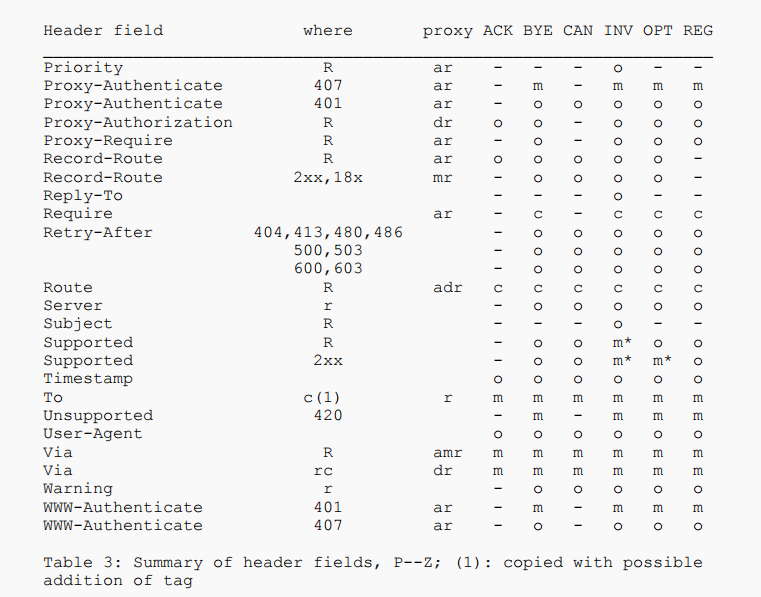


表3：头域概览，P-Z （1）和可能的附加tag一起拷贝。

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:  ;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 Permently

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 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：



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

小虎 2006-05-25 00:11

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。

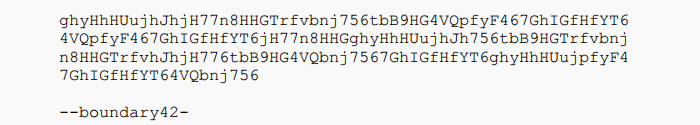
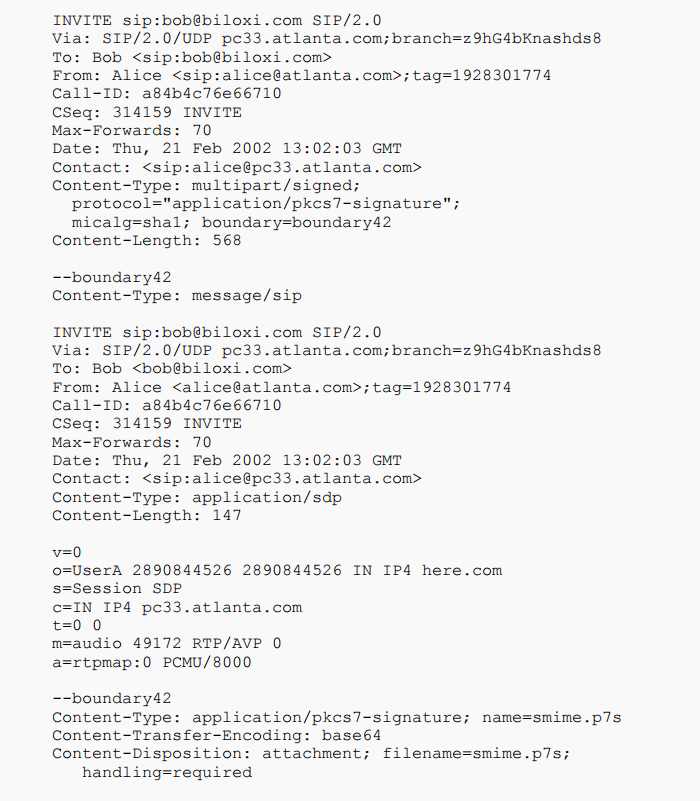
假设那个”message/sip”包体包含了最基本的对话要素（最小集合）（To， From， Call-ID， CSEq），并且一个签名的MIME包体可以提供优先的身份认证。在这个特别的最小集合上，如果接受方不认识用于签名MIME包体的信任状，并且不能被检验，那么在同一个信任状拥有者所初始化的会话中，这个信任状拥有者可以稍后发送一个在会话中的请求，包含这个签名来进行确认。如果签名MIME包体的接受方选择信任这个信任书（他们可以检验信任书，他们已经从信任列表中检查了，或者他们经常使用这个信任书），那么这个签名就和这个信任书的主题有着相同的身份一致性。

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：



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：



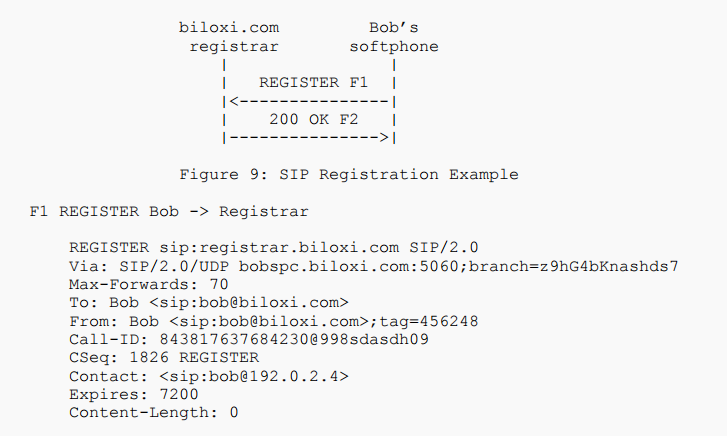


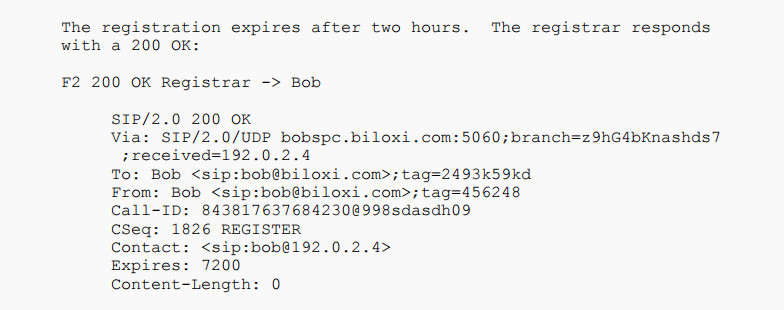
#### 24 例子

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

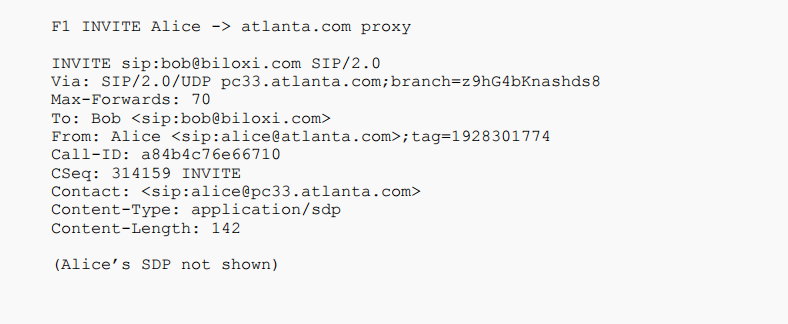
Bob在启动的时候进行注册。这个消息流在图9中展示。注意对于注册服务来说，通常需要认证，而且不像下边描述的这样简单。





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。



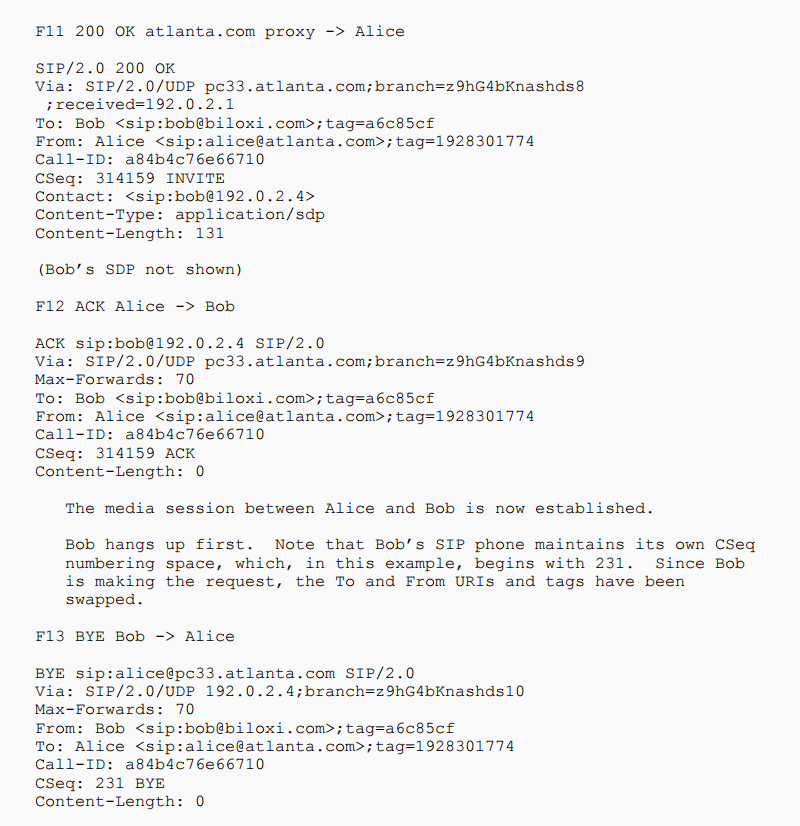












在Alice和Bob之间的媒体会话现在建立了。Bob首先挂机。注意Bob的SIP电话维持它自己的Cseq号码空间，在这里，是231开始的。由于Bob发起请求，那么To和From URI和tags交换了。



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。

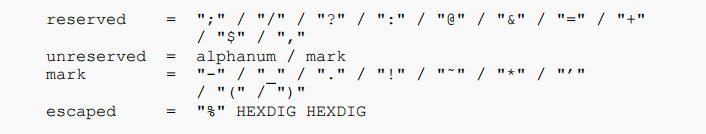
方括号的使用是在语法上可选的。在这里用于特定参数是可选的语义提示。

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：

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

小虎 2006-05-25 00:13

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 UDP0 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 UDP0 TCP/SCTP    17.2.1    ACK重传的等待时间

定时器J    64×T1 UDP0 TCP/SCTP    17.2.2    非INVITE请求重传的等待时间

定时器K    T4 UDP0 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.