

使用 **OpenSER** 构建电话通信系统

Building Telephony Systems with OpenSER

第一章：**SIP** 介绍（**Introduction to SIP**）

会话初始化协议是互联网工程任务组（IETF）制定的协议标准，在多个 RFC（Request for Comments）文档中被进行了描述说明。RFC3261 是最近的一个 RFC，一般称它为 SIP 版本 2。SIP 是一个应用层的协议，用来建立，修改终止会话（sessions）或是多媒体通话（multimedia calls）。这些会话可以会议（conferences），e-learning，网络电话和一些相似的应用。它是同 HTTP 协议相类似的文本协议并且被设计用来发起，保持，关闭用户之间的交互会话。目前 SIP 已经是 VoIP 领域被广泛使用的协议之一了，市场上几乎每一台 IP 电话都会支持它。

本章结束的时候你将能够：

- ☐ 描述 SIP 是什么
- ☐ 描述 SIP 是干什么的
- ☐ 描述 SIP 的框架
- ☐ 解释 SIP 主要部件的意义
- ☐ 理解并比较主要 SIP 消息
- ☐ 描述 INVITE 和 REGISTER 请求消息头部的处理过程

在建立和关闭多媒体通话的过程中，SIP 协议支持五种要素。

- ☐ 用户定位（User location）
- ☐ 用户参数协商（User parameters negotiation）
- ☐ 用户可用性（User availability）
- ☐ 通话建立（Call establishment）

□ 通话管理 (Call management)

SIP 协议被设计成多媒体框架的一部分，而这种多媒体框架包括 RVSP，RTP，RTSP 还有 SDP 等其他协议。然而，SIP 却并不依靠其他这些协议工作。

SIP 基础 (SIP Basics)

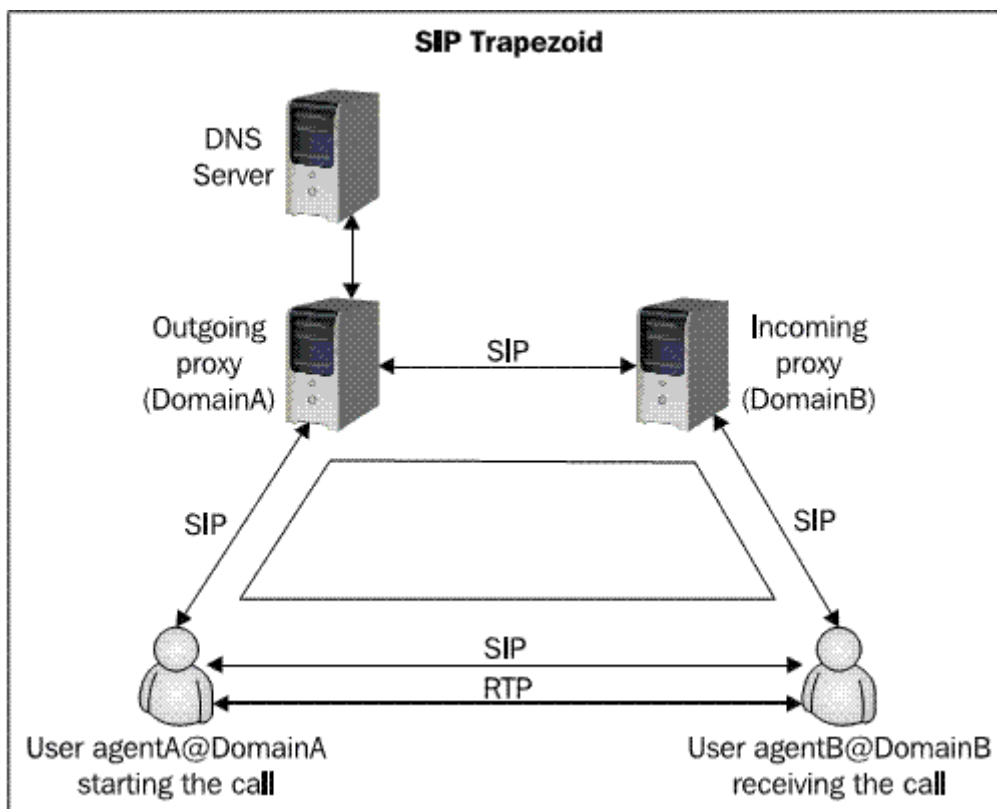
SIP 在工作的方式上与 HTTP 协议相类似。SIP 的地址像是 e-mail 的地址。SIP 代理中使用的一个比较有趣的特性就是“别名 (alias)”，也就是说你可以有多个 SIP 地址，譬如：

johndoe@sipA.com

[+554845678901@sipA.com](tel:+554845678901@sipA.com)

[45678901@sipA.com](tel:45678901@sipA.com)

在 SIP 的体系结构中，有多个用户代理和提供不同服务的服务器。SIP 使用点对点 (peer-to-peer) 的分布模型来和服务器进行消息的交互。服务器只进行消息 (signaling) 的处理，而用户代理的客户端和服务端既可以处理消息也可以处理媒体。下面的图描述了这样的一个体系：



在 SIP 模型中，用户代理，通常是一台 SIP 话机与它的 SIP 代理进行交互，从上图可以看到，外呼代理（outgoing proxy）将使用 INVITE 消息向外发出通话请求。

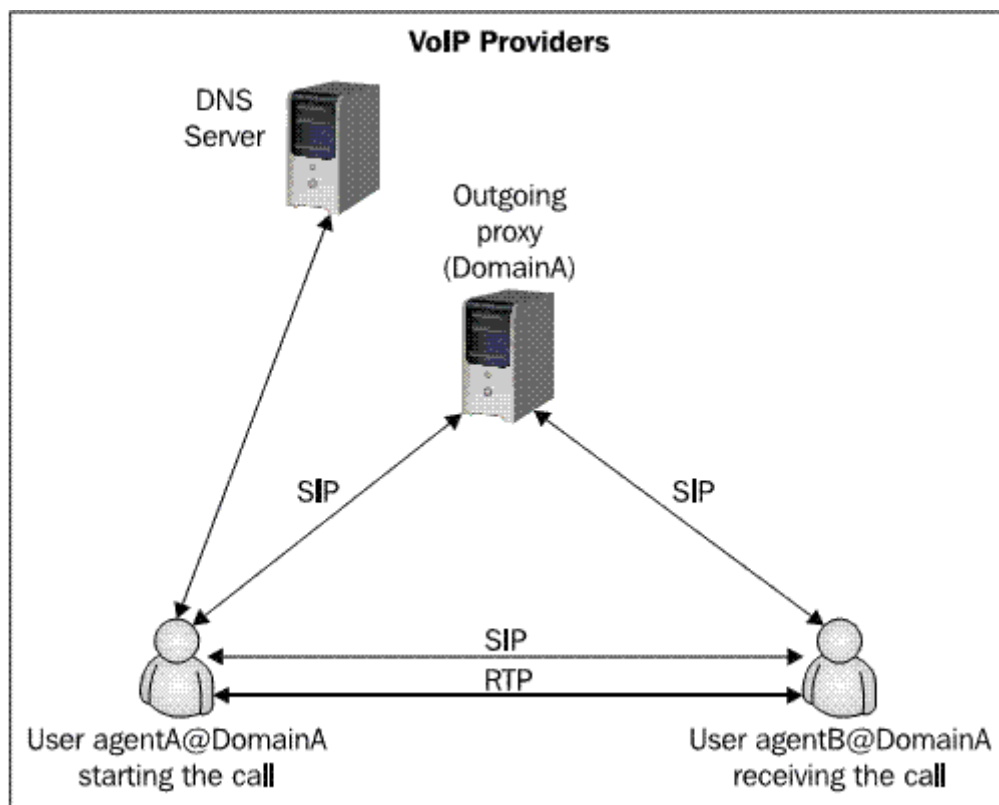
外呼代理将观察这通通话是否是被定向到外部的域名。然后它将向 DNS 服务器发出请求将目标域名解析为对应的 IP 地址。然后再将通话请求发送给 DomainB 对应的 SIP 代理。

呼入代理（incoming proxy）将在地址列表（location table）中查询 agentB 的 IP 地址。如果在地址列表这个地址与之前在注册过程中的 IP 地址对应那么呼入代理就可以定位这个地址了。现在就可以使用这个地址将通话请求发送到 agentB 了。

agentB 收到这个 SIP 消息后（INVITE），就拥有了可以与 agentA 建立 RTP 会话（通常是音频方面的会话）所需要的信息。使用 BYE 消息可以终止这个会话。

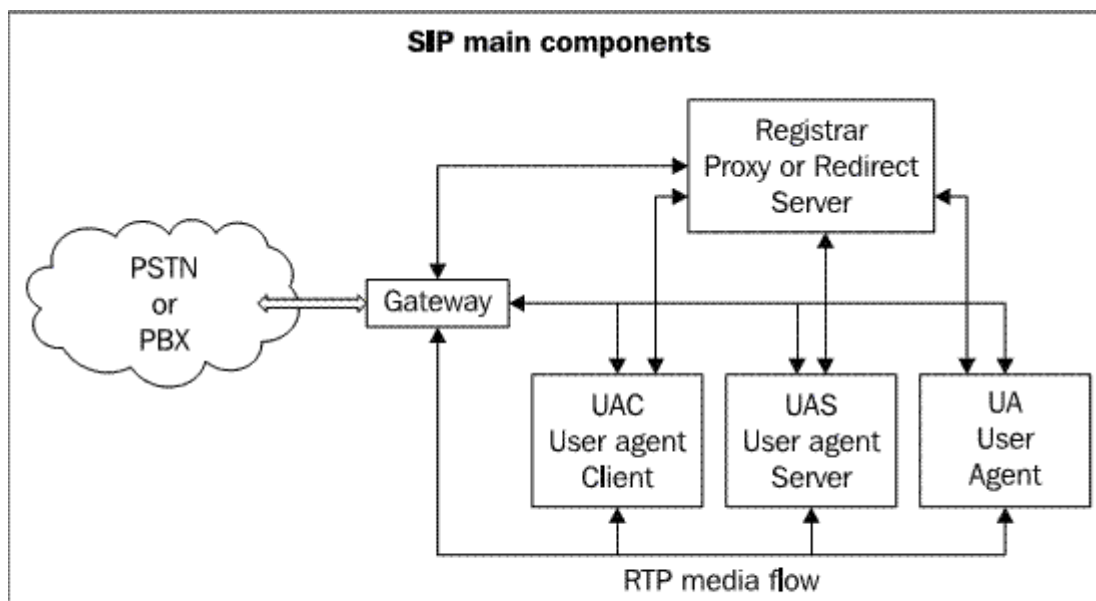
SIP 代理在 VoIP 提供者里的作用 / 上下文（SIP Proxy in the Context of a VoIP Provider）

通常 VoIP 服务的提供者并不会实现像上幅图那样的纯粹的 SIP 四边形结构，他们不会允许你向一个外部的域名发送通话请求，因为如果这样，那么将影响他们的收入（revenue stream）。取而代之的是一个接近三角形的 SIP 网络结构。（如下图所示）



SIP 工作原理（ SIP Operation Theory ）

在上图中，你可以看到 SIP 体系结构中的主要的构成部件。所有的 SIP 消息都会经过 SIP 代理服务器。另一方面，由 RTP 协议承载的媒体流则是从一端直接流向另一端。我们将在下面的列表中简要的对其中的一些构成部件进行解释。



- 用户代理客户端（UAC user agent client）——发起 SIP 消息的客户端或终端
- 用户代理服务端（UAS user agent server）——对接收到的从用户代理客户端发起的 SIP 消息进行相应的服务端。
- 用户代理（UA user agent）——SIP 终端（IP 电话，电话适配器（ATA），软电话（softphone））
- 代理服务器（Proxy Server）——从用户代理接收请求，并且如果指定的被请求的终端不在其域中时，会将请求发送给另外的 SIP 代理。
- 重定向服务器（Redirect Server）——接收请求，但是不直接发送给被叫用户，而是向主叫方发送目的地址的信息。
- 定位服务器（Location Server）——向代理服务器和重定向服务器提供被叫者的联系地址。

通常，物理上，代理服务器，重定向服务器和定位服务器存在于同一台电脑和软件中。

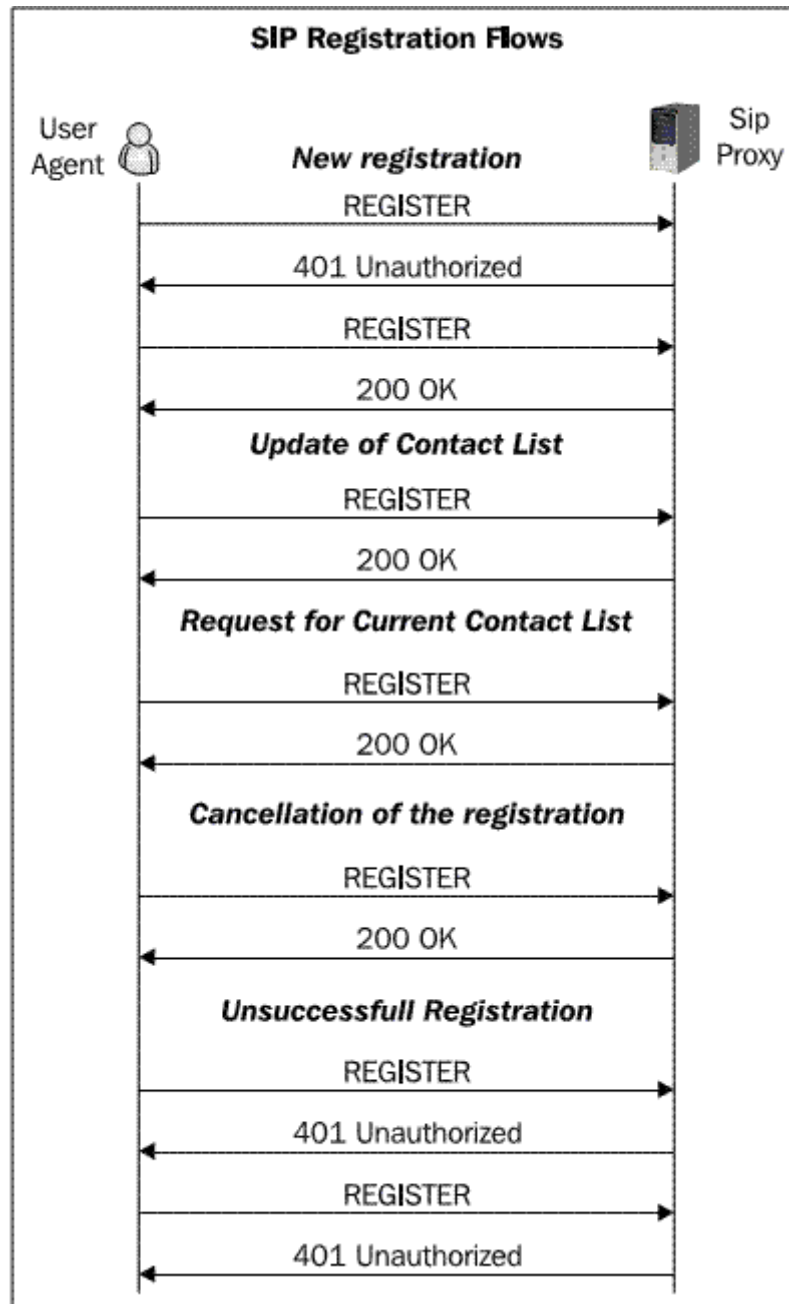
SIP 注册过程（SIP Registration Process）



SIP 协议中使用了一个构件叫做注册服务器。它不仅能够接收 REGISTER 消息请求，还能够将收到的消息包中的信息保存到管理对应域名的定位服务器上。SIP 协议具有发现能力；换句话说，就是如果一个用户要与另外一个用户开始会话，那么 SIP 协议必须要发现这个用户能够到达的主机存在。由于定位服务器可以收到请求消息并找到向什么地方发送，所以这个发现过程由定位服务器来完成。而这则是基于管理每个域的定位服务器维护着一个定位数据库的事实来实现的。注册服务器不仅可以接收客户端的 IP 地址，还能够接收其他类型的消息。比如，能够收到服务器上面的 CPL（Call Processing Language）脚本。

在一台话机能够接收一通通话之前，它需要在定位数据库中有注册信息。在这个数据库中我们要拥有所有电话的各自的相关的 IP 地址。在我们的例子中，你将看到 SIP 用户 8590@voffice.com.br 注册到 200.180.1.1 上面的过程。

RFC3665 定义实现了一个最小的功能集合，这是使得 SIP 进行 IP 网络交互时的最好实践。下面的图就是根据 RFC3665 中的描述所画出的注册事务处理流程。



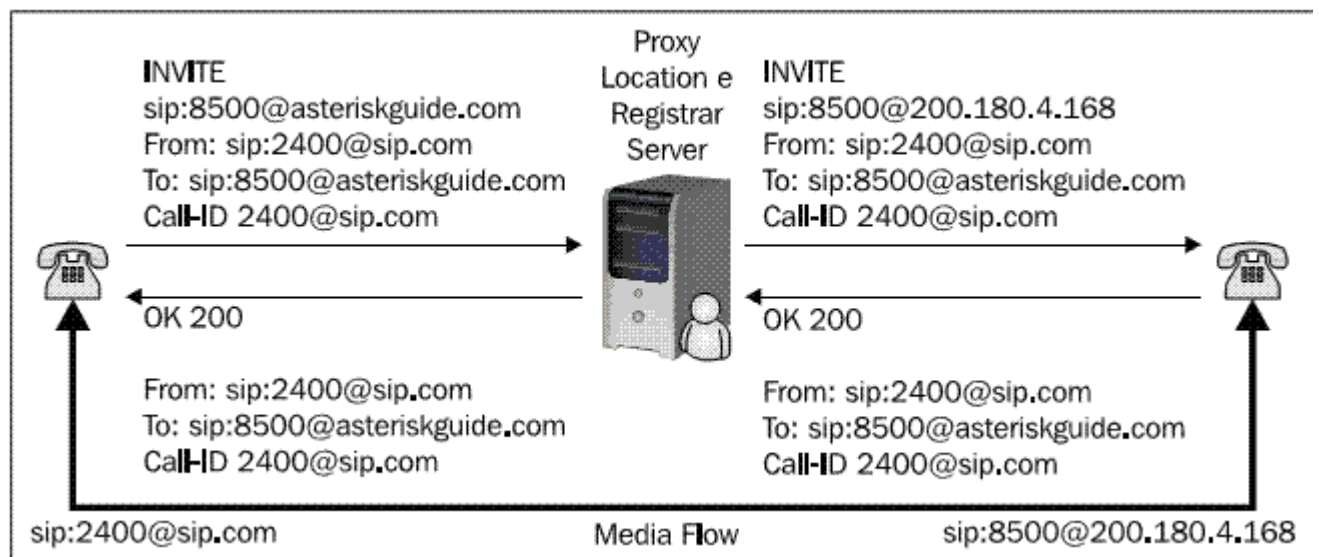
按照 rfc3665 中所说，与注册一个用户代理的过程相关的有五个基本的流程，如下所述：

1. 一个新的成功的注册（A successful new registration）——用户代理在发送 Register 请求后，将收到认证过程的挑战。我们将在阐述验证过程的章节中看到这个过程的细节。
2. 联系列表的更新（An update of the contact list）——由于不再是新的注册，消息中已经包含了摘要（digest），那么不会返回 401 消息。为了改变联系列表，用户代理仅仅需要发送一条在 CONTACT 头中带有新的联系信息的注册信息即可。

3. 请求获得当前的联系列表——在这种情况下，用户代理将把发送消息中的 CONTACT 头置空，表明用户希望向服务器询问当前的联系列表。在回复的 200OK 消息中，SIP 服务器将把当前的联系列表放在其 CONTACT 的头中。
4. 取消注册（Cancellation of a registration）——用户代理在发送的消息中将 EXPIRES 头置成 0，并且将 CONTACT 头设置为“*”表示将此过程应用到所有存在的联系信息。
5. 不成功的注册（Unsuccessful Registration）——用户代理客户端（UAC）发送一条 Register 请求消息，收到一条“401 Unauthorized”消息，事实上，这个过程同成功注册过程相同。但是接下来，它进行哈希运算尝试进行认证。而服务器检测到的是一个无效的密码，继续发送“401 Unauthorized”消息。这个过程一直重复直到重复次数超过在 UAC 设置的最大值。

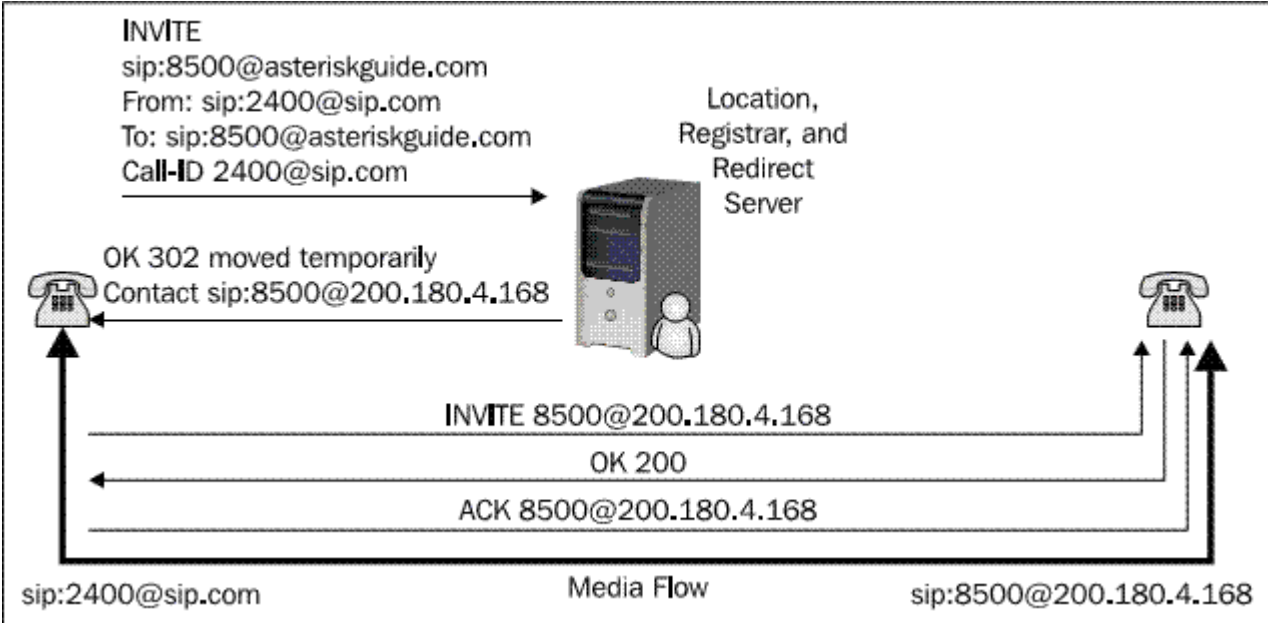
作为 SIP 代理运转的服务器（Server Operating as a SIP Proxy）

在 SIP 代理模式下，所有的 IP 消息都要经过 SIP 代理。这种行为在向诸如计费（billing）的过程中帮助很大，而且迄今为止，这也是一种最普遍的选择。但是它的缺点就是在会话建立过程中的所有的 SIP 交互中，服务器造成的额外开销也是客观的。要记住的是，即使服务器作为 SIP 代理在工作时，RTP 包也总是直接从一端传送到另一端，而不会经过服务器。



作为 SIP 重定向运转的服务器（Server Operating as a SIP Redirect）

SIP 代理可以运行在 SIP 重定向模式。在这种模式下，SIP 服务器的处理量是相当巨大的，因为它不需要保持事务处理的状态。在对 INVITE 消息进行初始化后，仅仅向 UAC 回复一条“302 Moved Temporarily”消息就可以离开 SIP 对话（dialog）了。在这种模式下的 SIP 代理，即使只是利用非常少的资源也可以每小时传送上百万的通话。当你需要的规模很大并且不需要对通话计费的情况下这种模式通常会被使用。



基本消息（ Basic Messages ）

在 SIP 环境中会被发送的基本的消息有：

SIP basic messages		
Method	Description	RFC
ACK	Acknowledge an INVITE	RFC3261
BYE	Terminate an existing session	RFC3261
CANCEL	Cancel a pending request	RFC3261
INFO	Mid-call signaling information	RFC2976
INVITE	Session establishment	RFC3261
MESSAGE	Instant message transport	RFC3428
NOTIFY	Send information after subscribe	RFC3265
OPTIONS	Query the capabilities of the UA	RFC3261
PRACK	Acknowledge a provisional response	RFC3262
PUBLISH	Upload status information to the server	RFC3903
REGISTER	Register the user and update the location table	RFC3261
REFER	Ask another UA to act upon URI	RFC3515
SUBSCRIBE	Establish a session to receive future updates	RFC3265
UPDATE	Update session state information	RFC3311

大多数时间内，你会使用到 REGISTER，INVITE，BYE 还有 CANCEL。而另外一些消

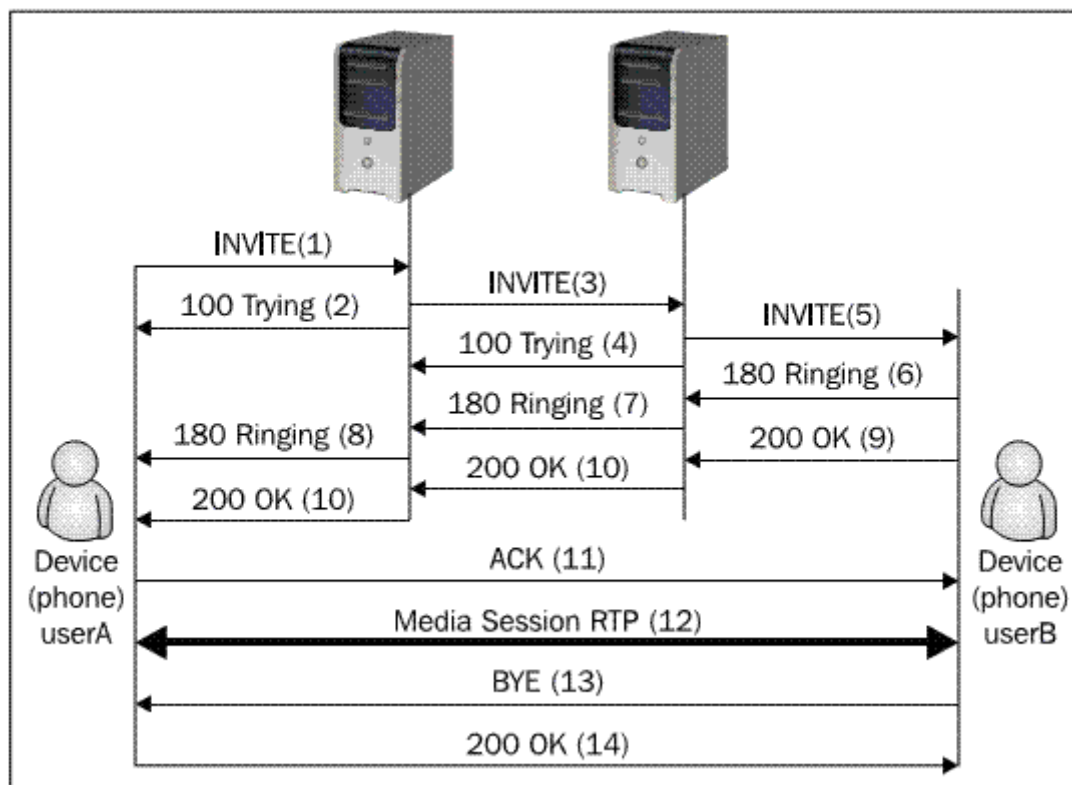
息会被用在其他的特性当中。举例来说，INFO 被用在 DTMF relay 和通话中消息信息（mid-call signaling information）。PUBLISH，NOTIFY 和 SUBSCRIBE 则用来支持列席系统（presence systems）。REFER 用来进行通话转接（call transfer）而 MESSAGE 则应用于一些聊天应用程序中。更新的消息也会随着协议标准化进程而随之出现。

像 HTTP 协议一样，这些消息的响应也会以文本形式出现。其中一些最终的响应消息被列在下图当中：

Response Codes			
Description	Code	Examples	
Informational or provisional response	1XX	100 Trying 180 Ringing 181 Call Is Being Forwarded	182 Queued
Success	2XX	200 OK 202 Accepted	
Redirect	3XX	300 Multiple Choices 301 Moved Permanently 302 Moved Temporarily	303 See Other 305 Use Proxy 380 Alternative Service
Client Errors	4XX	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 409 Conflict 410 Gone	411 Length Required 413 Request Entity Too Large 414 Request-URI Too Large 415 Unsupported Media Type 420 Bad Extension 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
Server Errors	5XX	500 Internal Server Error 501 Not Implemented 502 Bad Gateway	503 Service Unavailable 504 Gateway Time-out 505 SIP Version not supported
Global Errors	6XX	600 Busy Everywhere 603 Decline	604 Does not exist anywhere 606 Not Acceptable

SIP 对话流程（SIP Dialog Flow）

这一节将通过一个简单的例子来介绍一些基本的 SIP 操作。先让我们来诊视下图展示的两个用户代理之间的消息顺序。你可以看到伴随这 RFC3665 描述的会话建立过程还有几个其它的流程。



我们在这些消息上标上了序号。在这个例子中用户 A 使用 IP 电话向网络上的另外一台 IP 电话发出通话请求。为了完成通话，使用了两个 SIP 代理。

用户 A 使用称为 SIP URI 的 SIP 标识向用户发出通话。URI 就像是一个电子邮件地址，比如 sip:userA@sip.com。一种可靠安全的 SIP URI 也可以被使用，譬如 sips:userA@sip.com。使用 SIPS 建立的通话将会在主叫和被叫之间使用安全可靠的传输机制（TLS—Transport Layer Security）

事务（transaction）的建立始于用户 A 向用户 B 发送 INVITE 请求消息。INVITE 请求中包含一些特定头域。这些头域被称之为属性，为消息提供了额外的一些信息。包括唯一的标识，目的地，还有关于会话（session）的信息。

INVITE from A->B

```
INVITE sip:userB@sipB.com SIP/2.0
Via: SIP/2.0/UDP moon.sipA.com;branch=z9hG4bK776asdhds
Max-Forwards: 70
To: userB <sip:userB@sipB.com>
From: userA <sip:userA@sipA.com>;tag=1234567890
Call-ID: a84b4c76e66710@moon.sipA.com
CSeq: 314159 INVITE
Contact: <sip:userB@sun.sipB.com>
Content-Type: application/sdp
Content-Length: 142
(SDP not shown)
```

第一行是消息的方法名（the method name）。接下来是列出的头域。这个例子包含了所需要的头的最小集合。我们将在下面简要的描述这些头域。

- **VIA**：它包含了用户 A 等待发出请求对应响应的所在地址。还包含了一个叫做“branch”的参数，这个参数用来唯一的标识这个事务（transaction）。VIA 头将最近从“SIP 跳”（SIP hop）定义为 IP，传输，和指定事务参数（The VIA header defines the last SIP hop as IP，transport，and transaction-specific parameters）。VIA 专门用来路由响应消息。请求经过的每一个代理都会增加一个 VIA 头。而对于响应消息而言，相对于再次向定位服务器或是 DNS 服务器进行定位请求，使用 VIA 头进行路由将更加容易。
- **TO**：它包含了名字（显示名(display name)）和最初选择的目的地 SIP URI（这里是 sip: userB@sip.com）。TO 头域不被用来路由消息包。
- **FROM**：它包含了名字和表明主叫 ID（caller ID）的 SIP URI（这里是 sip: [userA@sip.com](#)）。这个头域有一个 tag 参数，而这个参数包含了被 IP 电话添加进 URI 的一个随机字符串。是被用来进行辨识唯一性的。tag 参数被用在 TO 和 FROM 头域中。作为一种普遍的机制用来标识对话（dialog），对话是 CALL-ID 和两个 tag 的结合，而这两个 tag 分别来自参与对话的双方。Tags 在并行派生（parallel forking）中作用显著。
- **CALL-ID**：它包含了一个作为这通电话全局性的唯一的标识，而这个唯一标识是有一个随机字符串，来自 IP 电话的主机名或是 IP 地址结合而成的。TO，FROM 的 tag 和 CALL-ID 的结合完整的定义了一个端到端的 SIP 关系，这种关系就是我们所知道的 SIP 对话（SIP dialog）

- CSEQ: CSEQ 或者称之为命令序列 (command sequence) 包含了一个整数和一个方法名。CSEQ 数对于每一个在 SIP 对话中的新请求都会递增, 是一个传统的序列数。
- CONTACT: 它包含一个代表直接路由可以联系到用户 A 的 SIP URI, 通常是有用户名和 FQDN (fully qualified domain name)。有时候域名没有被注册, 所以, IP 地址也是允许使用的。VIA 头告诉其他的组件向什么地方发送响应消息, 而 CONTACT 则告诉其他组件向什么地方发送将来的请求消息。
- MAX-FORWARDS: 它被用来限制请求在到达最终目的地的路径中被允许的最大跳数 (hops)。由一个整数构成, 而这个整数在每一跳中将会递减。
- CONTENT-TYPE: 它包含了对内容消息的描述。
- CONTENT-LENGTH: 它用来告知内容消息的字节数。

会话的一些细节, 像媒体类型和编码方式并不是使用 SIP 进行描述的。而是使用叫做会话描述协议 (SDP RFC2327) 来进行描述。SDP 消息由 SIP 消息承载, 就像是一封电子邮件的附件一样。

过程如下:

话机开始并不知道用户 B 和负责域 B 的服务器的位置。因此, 它向负责 sipA 域的服务器发送 INVITE 消息请求。发送地址在用户 A 的话机中进行设置或通过 DHCP 发现。服务器 sipA.com 也就是我们知道的域 sipA.com 的 SIP 代理服务器。

1. 在这个例子中, 代理服务器收到 INVITE 请求消息并发送“100 trying”响应消息给用户 A, 表明代理服务器已经收到了 INVITE 消息并正在转发这个请求。SIP 的响应消息使用一个三个数字组成的数字码和一条描述语句说明响应的类型。并拥有和 INVITE 请求一样的 TO, FROM, CALL-ID 和 CSEQ 等头域, 以及 VIA 和其“branch”参数。这就使得用户 A 的话机同发出的 INVITE 请求联系在一起。

2. 代理 A 定位代理 B 的方法是向 DNS 服务器 (SRV 记录) 进行查询以找到负责 sipB 的 SIP 域的服务器地址并将 INVITE 请求转发给它。在向代理 B (译者注: 这里作者写的是 proxyA, 但是应该是 B) 发送 INVITE 消息前, 代理 A 将其自己的地址通过 VIA 头添加进 INVITE, 这就使得用户 A 的话机同 INVITE 请求的响应消息联系在一起。

3. 代理 B 收到 INVITE 请求，返回“100 Trying”消息响应，表明其正在处理这个请求。

4. 代理 B 查询自己的位置数据库以找到用户 B 的地址，然后将自己的地址也通过 VIA 头域添加进 INVITE 消息发送给用户 B 的 IP 地址。

5. 用户 B 的话机收到 INVITE 消息后开始振铃。话机为了要表明这种情况（振铃）发送回“180 Ringing”响应消息。

6. 这个消息以相反的方向路由通过那两个代理服务器。每一个代理利用 VIA 头域来决定向哪里发送响应消息并从顶部将其自己的 VIA 头去除。结果就是，180 Ringing 消息不需要任何的 DNS 查询，不需要定位服务的响应，也不需要任何的状态处理就能够返回到用户那里。这样的话，每一个代理服务器都能够看到由 INVITE 开始的所有消息。

7. 当用户 A 的话机收到“180 Ringing”响应消息后开始“回铃”，表明另一端的用户正在振铃。一些话机是通过显示一些信息进行表示的。

8. 在这个例子中，用户 B 对对方发起的通话进行了响应。当用户 B 响应时，话机发送“200 OK”响应消息以表明通话被接起。“200 OK”的消息体中包含了会话的描述信息，这些信息包括指定了编码方式，端口号，以及从属于会话的所有事情。作这项工作的就是 SDP 协议。结果就是，在从 A 到 B（INVITE）和从 B 到 A（200 OK）的两个阶段，双方交换了一些信息，以一种简单的“请求/响应”的模式协商了在这通通话中所需的资源和所需要的能力要求。如果用户 B 不想得到这通通话或是此刻处于忙线中，200 Ok 将不会发出，取代它的是描述这种状况（这里是 486 Busy Here）的消息。

Response 200 Ok

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP sipB.com
;branch=z9hG4bKnashds8;received=192.0.2.3
Via: SIP/2.0/UDP moon.sipA.com
;branch=z9hG4bK77ef4c2312983.1;received=192.0.2.2
Via: SIP/2.0/UDP phoneA.sipA.com
;branch=z9hG4bK776asdhds ;received=192.0.2.1
To: userB <sip:userB@sipB.com>;tag=a6c85cf
From: userA <sip:userA@sipA.com>;tag=1928301774
Call-ID: a84b4c76e66710@phoneA.sipA.com
CSeq: 314159 INVITE
Contact: <sip:userB@192.0.2.4>
Content-Type: application/sdp
Content-Length: 131
```

第一行是响应码和描述信息（OK）。接下来是头域行。VIA，TO，FROM，CALL-ID 和 CSEQ 是从 INVITE 请求中拷贝的。有三个 VIA 头，一个是用户 A 添加的，另一个是代理 A 添加的，最后一个则是代理 B 添加的。用户 B 的 SIP 话机在对话的双方加入了一个 TAG 参数，这个参数在这通电话的以后的请求和响应消息中都将出现。

CONTACT 头域中包含了 URI 信息，这个 URI 信息是用户 B 能够直接被联系到他们自己的 IP 话机的地址。

CONTENT-TYPE 和 CONTENT-LENGTH 头域先给出了关于 SDP 头的一些信息。而 SDP 头则包含了用来建立 RTP 会话的媒体相关的参数。

1. 在这个例子中，“200 Ok”消息通过两个代理服务器被送回给用户 A，之后用华 A 的话机停止“回铃”表明通话被接起。

2. 最后用户 A 向用户 B 的话机发送 ACK 消息确认收到了“200 Ok”消息。在这里，ACK 避开了两个代理服务器直接发送给用户 B。ACK 是 SIP 中唯一不需要进行响应的消息请求。两端在 INVITE 的过程中从 CONTACT 消息中了解双方的地址信息。也结束了 INVITE/200 OK/ACK 的过程，这个过程也就是我们所熟知的 SIP 三次握手。

3. 这个时候两个用户之间开始进行会话，他们以用 SDP 协议协商好的方式来发送媒体包。通常这些包是端对端进行传送的。在会话中，通话方可以通过发送一个新的 INVITE

请求来改变会话的一些特性。这叫做 re-invite。如果 re-invite 不被接受，那么“488 Not Acceptable Here”响应就会被发出，但是会话不会因此而失败。

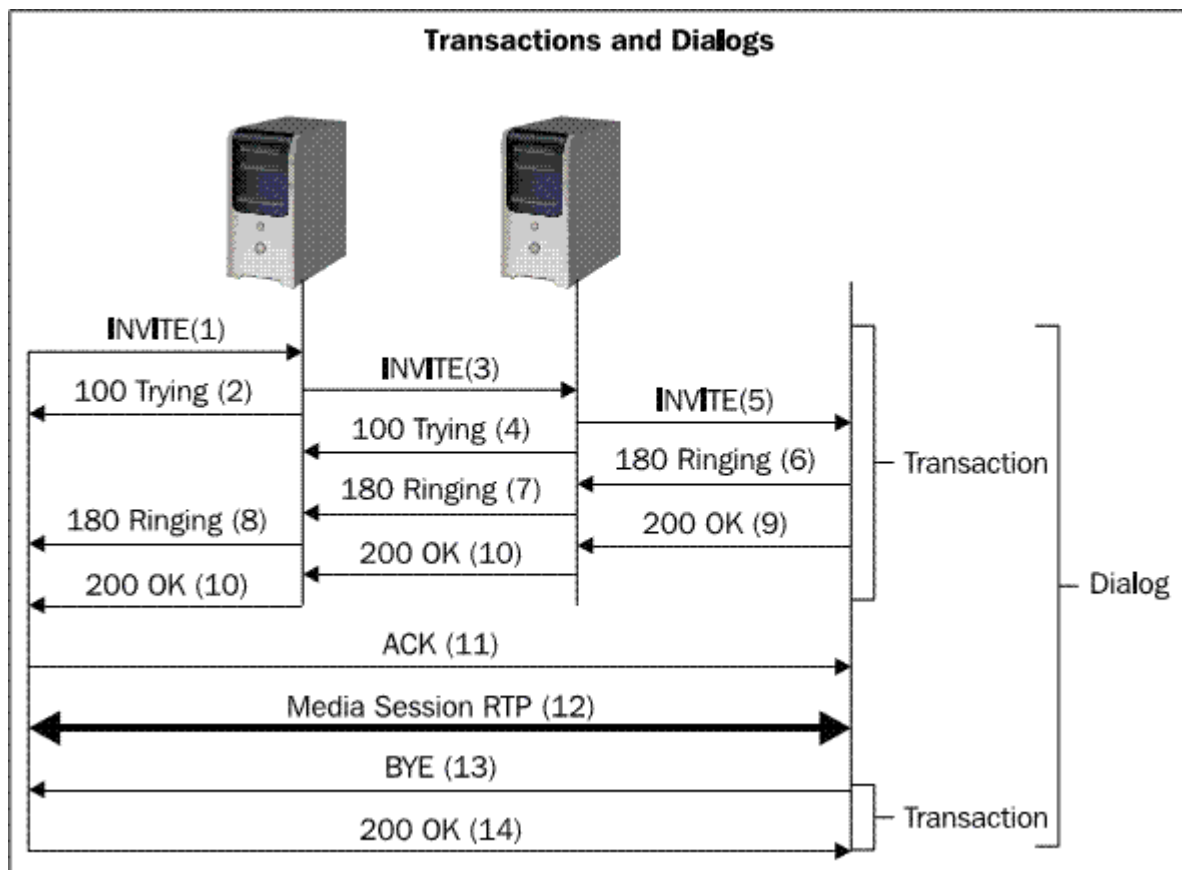
4. 要结束会话的时候，用户 B 产生 BYE 消息来中断通话。这个消息绕过两个代理服务器直接路由回用户 A 的软电话上。

5. 用户 A 发出“200 OK”响应消息以确认收到了 BYE 消息请求，从而结束会话。这里，不会发出 ACK。ACK 只在 INVITE 请求过程中出现。

有些情况下，在整个会话过程中，对于代理服务器来说，能够待在消息传输的中间位置来观察两端的所有消息交互是很重要的。如果代理服务器想在 INVITE 请求初始化完成后还待在此路径中，可以在请求消息中添加 RECORD-ROUTE 头。用户 B 的话机得到了这个消息，之后在其消息中也会带有这个头，并且会将消息发送回代理。记录路由 (Record Routing) 在大多数的方案中都会被使用。

REGISTER 请求是代理 B 用来定位用户 B 的方法。当话机初始化的时候或是在通常的时间间隔中，软电话 B 向在域 sipB 中的一个服务器 (SIP REGISTRAR) 发送 REGISTER 请求。注册服务器 (REGISTER) 将 URI 与一个 IP 地址联系在一起，这种绑定被存储在定位服务器上面的数据库里。通常，注册服务器，定位服务器，和代理服务器在同一台物理机器上，并使用相同的软件。OpenSER 就能够扮演这三种角色。一个 URI 只能够在一个特定的时间内由一个单独的机器注册。

SIP 事务和对话 (SIP Transactions and Dialogs)



理解“事务”（transaction）和“对话”（dialog）的区别是非常重要的。事务发生在一个用户代理客户端和另一个用户代理服务端之间，包含从第一请求到最后一个响应的所有消息。中间的阶段性的响应消息可以是以 1 开头的三位数字码（比如 180 Ringing），最终的响应消息则是以 2 开头的三位数字码（比如 200 OK）。事务包括的范围由 SIP 消息中 VIA 头形成“堆栈”来决定。因此，用户代理在初始化 invite 后才不需要依靠 DNS 或位置表进行消息路由。

对话（Dialog）通常是从 INVITE 事务开始，由 BYE 事务结束。一个对话由 CALL-ID 头唯一标识。TO tag, FROM tag 和 CALL-ID 的结合来完整的定义。

按照 rfc3665 的描述，有 11 个基本的会话建立流程。其列出的并不一定是完整的，但是覆盖了最好的例子。前两个流程在这一章节中进行了阐述——“成功建立会话 Successful Session Establishment”和“通过两个代理建立会话 Session Establishment Through Two Proxies”。其中的一些流程的描述你将在其他阐述呼叫前传（call forwarding）（譬如：“不接听导致没能成功建立 Unsuccessful with no Answer”和“忙音导致建立失败 Unsuccessful Busy”）的章节中看到。

RTP 协议（The RTP Protocol）

译者注：应为 Real Time Transport Protocol

实时传输协议是负责诸如音频和视频等数据的实时传输的。它标准化于 RFC3550。使用 UDP 协议进行传输承载。为了能够被实时传输，音频或视频必须经过一定的编码打包。最基本的，该协议允许使用如下的一些特性对进出的数据包媒体传输的时间和内容要求进行指定：

- 序号
- 时间戳
- 无重传机制的包的前转
- 源识别
- 内容识别
- 同步

与 RTP 相伴的协议叫做 RTCP (Real Time Control Protocol)，被用作对 RTP 包进行监控。它可以度量延迟和抖动。

编码 (Codecs)

RTP 协议描述的内容通常会由一种编码方式进行编码。每一种编码方式都有一种指定的用途。一些有压缩算法而另外一些则不需要。比较普遍的是使用不需要压缩的 G.711 编码方式。一个信道 64Kbps 的带宽要求需要一个高速的网络，通常在局域网 (LANs) 中比较常见。但是在广域网 (WAN) 中对于一个单独的声音信道来说，64Kbps 的带宽购买起来比较昂贵。这时候诸如 G.729 和 GSM 等可以将声音包压缩至 8Kbps 的编码方法则可以节省很多的带宽。由 Global IP sound 公司发明的 iLBC 编码方式则可以掩盖网络中由丢包造成的影响。在丢包率带到 7% 的情况下，使用 iLBC 仍然能够维持一个比较好的声音质量。所以对于你的 VoIP 提供商支持的编码方式你必须进行明智的选择。

DTMF — Relay

在一些情况下，RTP 协议被用来承载诸如 DTMF 的信号信息。RFC2833 中描述了一种方法，这种方法就是将 DTMF 作为 RTP 协议中的命名事件 (named events) 进行传输。在用户代理客户端和用户代理服务端之间使用同一种方法进行 DTMF 传输是非常重要的。

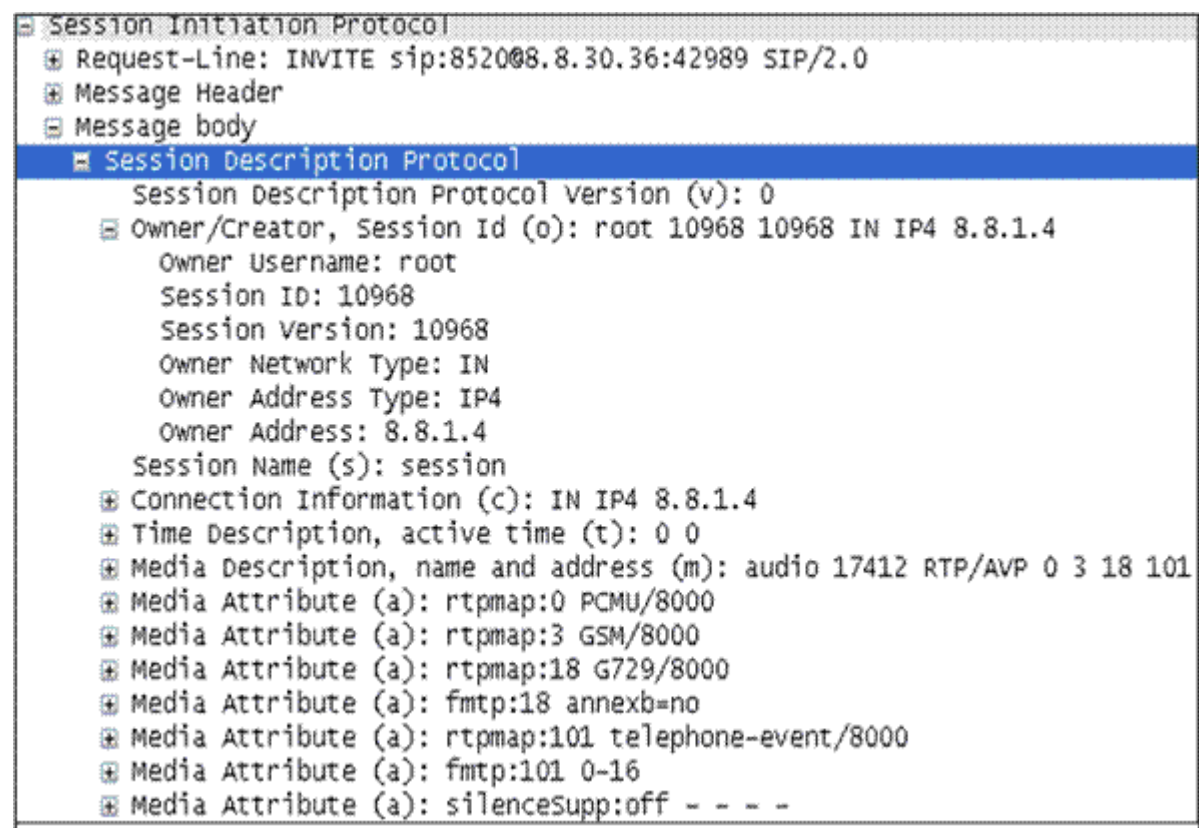
实时控制协议 (**Real Time Control Protocol-RTCP**)

RTCP 可以对接受的质量进行反馈。它为 RTP 媒体流提供带外控制信息。诸如抖动 (Jitter)，往返时延 (RTT—Round Trip Time)，传输延迟 (latency) 和丢包等的數據可以使用 RTCP 进行搜集。RTCP 通常用来对声音质量进行报告。

会话描述协议 (**Session Description Protocol-SDP**)

SDP 协议在 RFC4566 中被进行了详细的描述。它是在用户代理之间进行会话参数协商之用的。媒体的细节，传输的地址还有其他与媒体相关的一些信息都使用 SDP 协议在用户代理之间进行交互。通常，INVITE 消息中包含了 SDP“供给消息”，而 200 OK 则包含了“回答消息”。这些消息会在下面的图中进行展示。在下图中，你可以观察到 GSM 编码方式被“供给”，但是另外一台话机却并不支持该编码，那么然后它将使用它本身支持的编码方式进行“回答”，在这个例子中它支持的是 G.711 ulaw (PCMU) 和 G.729 编码方式。会话的“rtpmap:101”就是在 RFC2833 中描述的 DTMF—relay 信息。

INVITE (SDP Offer)



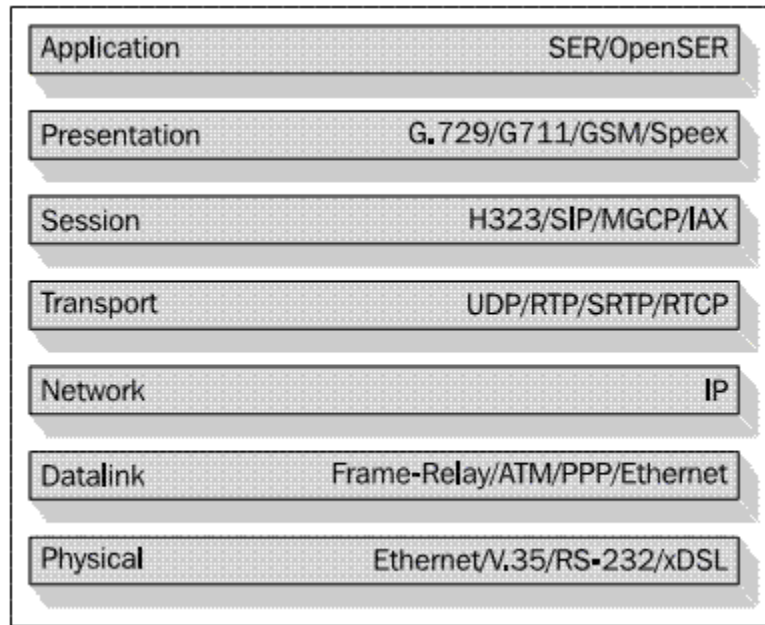
```
Session Initiation Protocol
+ Request-Line: INVITE sip:8520@8.8.30.36:42989 SIP/2.0
+ Message Header
+ Message body
+ Session Description Protocol
  Session Description Protocol version (v): 0
  Owner/Creator, Session Id (o): root 10968 10968 IN IP4 8.8.1.4
    Owner Username: root
    Session ID: 10968
    Session Version: 10968
    Owner Network Type: IN
    Owner Address Type: IP4
    Owner Address: 8.8.1.4
  Session Name (s): session
  Connection Information (c): IN IP4 8.8.1.4
  Time Description, active time (t): 0 0
  Media Description, name and address (m): audio 17412 RTP/AVP 0 3 18 101
  Media Attribute (a): rtpmap:0 PCMU/8000
  Media Attribute (a): rtpmap:3 GSM/8000
  Media Attribute (a): rtpmap:18 G729/8000
  Media Attribute (a): fmtp:18 annexb=no
  Media Attribute (a): rtpmap:101 telephone-event/8000
  Media Attribute (a): fmtp:101 0-16
  Media Attribute (a): silencesupp:off - - - -
```

200 OK (SDP Answer)

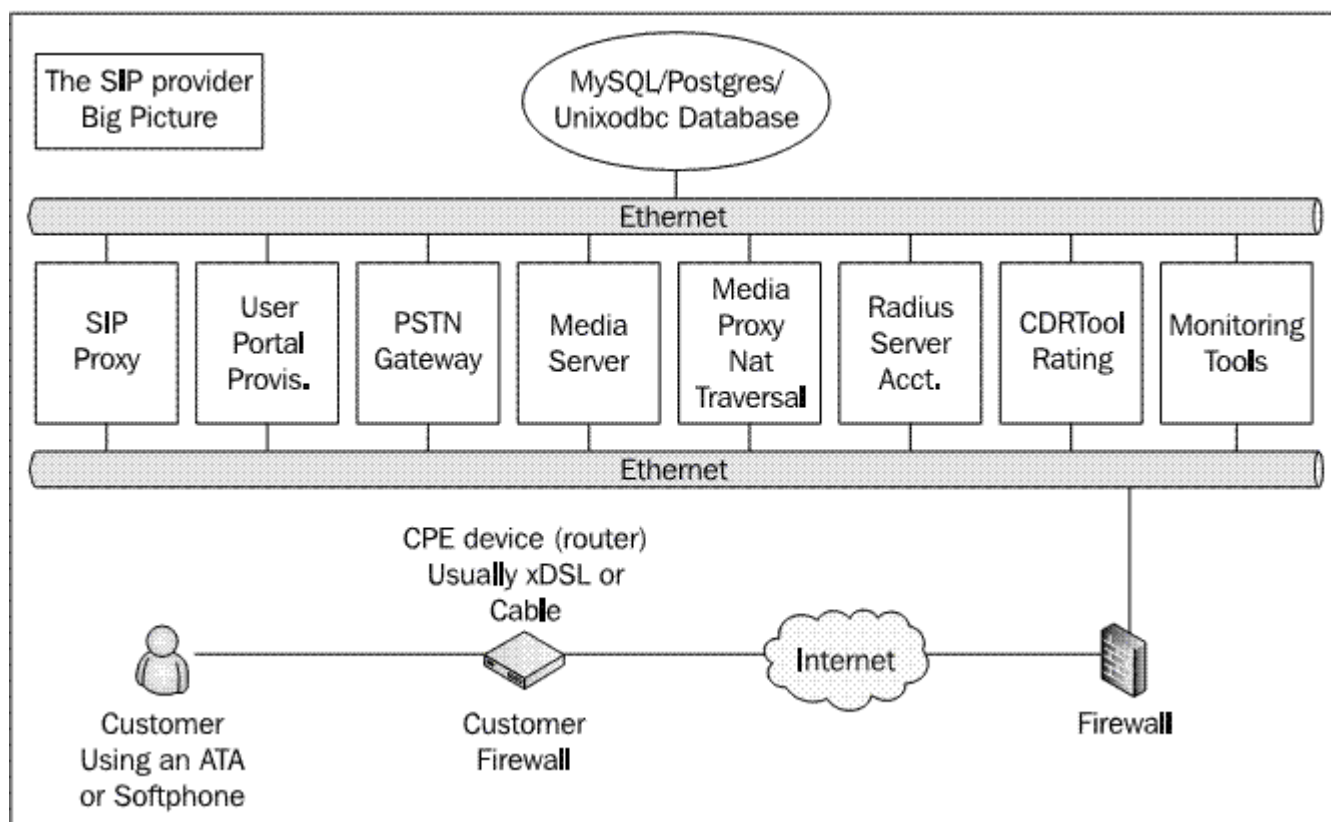
```
[-] Session Initiation Protocol
  [+ Status-Line: SIP/2.0 200 OK
  [+ Message Header
  [- Message body
  [- Session Description Protocol
    Session Description Protocol Version (v): 0
    [- Owner/Creator, Session Id (o): root 11218 11218 IN IP4 8.8.1.4
      Owner Username: root
      Session ID: 11218
      Session Version: 11218
      Owner Network Type: IN
      Owner Address Type: IP4
      Owner Address: 8.8.1.4
      Session Name (s): session
    [+ Connection Information (c): IN IP4 8.8.1.4
    [+ Time Description, active time (t): 0 0
    [+ Media Description, name and address (m): audio 17428 RTP/AVP 0 18 101
    [+ Media Attribute (a): rtpmap:0 PCMU/8000
    [+ Media Attribute (a): rtpmap:18 G729/8000
    [+ Media Attribute (a): fmtp:18 annexb=no
    [+ Media Attribute (a): rtpmap:101 telephone-event/8000
    [+ Media Attribute (a): fmtp:101 0-16
    [+ Media Attribute (a): silenceSupp:off - - -
```

SIP 协议与 OSI 七层模型 (The SIP Protocol and the OSI Model)

理解声音相关协议的每一个协议对于 OSI 模型是属于哪一层也是相当重要的。



VoIP 服务提供商的整体框图 (The VoIP Provider "Big Picture")



在我们开始深入的挖掘 SIP 代理之前，了解 VoIP 提供商的解决方案中的所有部件是非

常重要的。服务提供者通常由多个服务器（servers）和多个服务（services）组成。这里说的服务可以根据规模的大小来决定是被安装在一台单独的服务器上还是安装在多台机器上面。

本书将在前几个章节中按照这张图从左到右来描述每一个部件。所有的章节中都将使用这张图来帮助你了解你所处的位置何在。

SIP 代理（SIP Proxy）

SIP 代理是我们解决方案中的核心部件。用来负责用户的注册和维护位置数据库（映射 IP 和 SIP 地址）。所有的 SIP 路由和消息都会被 SIP 代理处理，它也负责一些用户端级别的服务譬如呼叫前转，白/黑名单，快速拨号等。这个部件从不处理媒体（RTP 包），所有与媒体相关的包都从用户代理客户端，服务器和 PSDN 网关直接路由。

用户，管理和供给入口（User，Administration，and Provisioning Portal）

用户管理和供给入口是一个重要的部件。在入口当中，用户订阅服务并且应该能够购买信用量，修改密码和验证他/她的账号。另一方面，管理者应该能够删除用户，改变用户信用等级，承认，删除权限。对于管理员来说，“供给（Provisioning）”过程使得用户代理如 IP 电话，模拟话机适配器还有软电话的自动安装过程更加容易。

PSDN 网关（PSDN 网关）

为了能和公共的交换电话网络交互，PSTN 网关是需要的。通常，这个网关是使用 E1 或 T1 中继线的 PSTN 的接口。在这个领域中使用最广泛的是来自 Cisco，AudioCodes 和 Quintum 公司的网关。Asterisk 也占据了一定的市场份额，因为它的每个端口的价格要比竞争对手们便宜 75%。如何评估一个网关的好坏，要检查它对 SIP 协议扩展的支持程度，譬如对 RFC3515（REFER），RFC3891（Replaces）还有 RFC3892（Referred by）。这些协议使得我们能够在 SIP 代理背后进行呼叫转移；如果网关中不支持他们，进行呼叫转移是不太可能的事。

媒体服务器（Media Server）

因为 SIP 代理从不处理媒体，所以如 IVRs，语音邮箱，电话会议等和媒体相关的服务要能够在媒体服务中得到实现。由 iptel 开发的 SEMS SIP Express 媒体服务器具有一些很

好的特性，如 conference，voicemail 和 announcements 等。我们要再一次提到 Asterisk，因为它也能够用来提供这些服务。

穿透 NAT 的媒体代理或 RTP 代理（**Media Proxy or RTP**

Proxy for Nat Traversal）

任何 SIP 服务提供者必须要为他的客户提供 NAT 穿透的解决方案。媒体代理就是一座帮助处在对称式防火墙（symmetric firewall）之后的用户能够访问 SIP 服务提供者进行 RTP 连接的桥梁。如果没有了这些代理，服务提供者可能会流失 35% 的用户。你可以使用这些部件来实现一个通用的 NAT 穿透技术。媒体代理还可以帮助你进行记帐纠错，比如，因为某种原因，没有收到 BYE 消息而导致了 SIP 对话没有结束，结果记帐发生了错误等情况。

RADIUS 记账（RADIUS Accounting）

拥有一台安装了 RADIUS 的服务器是对通话进行记账的基本条件。SIP 服务提供者对账单记录是最关心的。OpenSER 可以被配置将一些记账信息发送到一台 RADIUS 服务器（比如 Radiator 或 FreeRADIUS）上。SIP 通话账单也可以被记录到数据库中。但是，这样将产生两条记录，而这两条记录需要手工进行核对。

用 **CDRTool** 计费（**CDRTool Rating**）

RADIUS 服务器可以记录关于通话时长的一些信息，但是却不能记录每通通话的资费信息。将资费信息应用到通话上是需要技巧的。这里我们使用 AG 项目组（cdrtool.agprojects.com）开发的被称为 CDRTool 的 GPL 工具。它负责将资费应用到通话上面。

监控工具（**Monitoring Tools**）

最后我们需要一些监控，故障检修和测试的工具来帮助我们定位并解决在 SIP 服务器上发生的一些问题。首当其冲的当然是协议分析工具，在余下的章节中我们将能够看到如何使用 ngrep，ethereal 和 tethereal。OpenSER 有一个被称为 SIP trace 的模块，我们也将使用到。

哪儿能够找到更多信息

SIP 协议最好的参考资料就是 RFC3261。阅读 RFC 确实有点沉闷（不过如果你有点失眠这确实是一种好办法）。你可以在 <http://www.ietf.org/rfc/rfc3261.txt> 上找到它。也可以在哥伦比亚大学的网站上找到好的 SIP 教程如：http://www.cs.columbia.edu/~coms6181/slides/11/sip_long.pdf。也可以在 <http://www.cs.columbia.edu/sip/> 上找到许多关于 SIP 的信息。

一个非常好的教程在 iptel 的网站上：http://www.iptel.org/files/sip_tutorial.pdf。

下面是一个叫做 SIP 应用者（SIP implementors）的邮件列表，你可以在上面就 SIP 的知识进行提问：<https://lists.cs.columbia.edu/mailman/listinfo/sip-implementors>。

概要（**Summary**）

这一章你已经学到了什么是 SIP 协议以及 SIP 协议的功能。也知道了诸如 SIP 代理，SIP 注册服务器，用户代理客户端，用户代理服务端，PSTN 网关等 SIP 协议中的部件。还看到了 SIP 的体系结构，主要的消息和处理过程。以及去什么地方可以找到更多相关的信息。

第二章：SIP 快速路由器（The SIP Express Router）

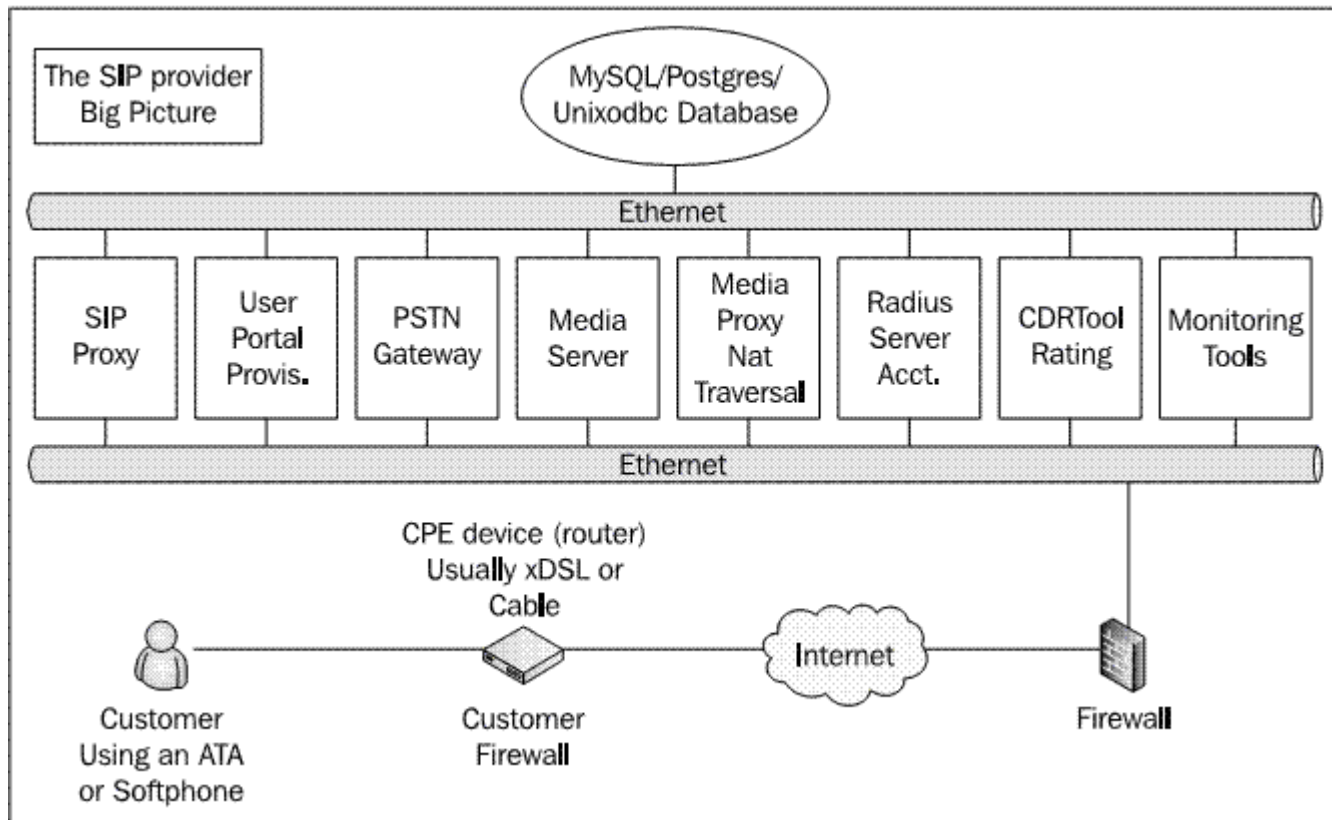
上一章节中我们讨论了 VoIP 提供商的“整体框图”（the big picture）。通常，一个 VoIP 提供商由几个部件构成。这些部件根据规模的大小或驻留在同一台机器上或分布于多个机器上面。其中的一个部件就是 SIP 代理，在我们的例子中代理服务器运行的是 OpenSER 软件。就像它的名字一样，描述 SER 的最好的东西就是 SIP 路由器（a SIP Router）。它能够对 SIP 的头域进行操作并能够以极高的速度对 SIP 包进行路由。第三方的模块给了 SER 极高的灵活性来完成一些原本没有的功能，诸如 NAT 穿透，IMS，负载均衡等其他功能。在这一章，我们将向你展示 SIP 快速路由器的能力和框架。

在本章的末尾，你将能够：

- 解释 SIP 快速路由器（SER）到底是什么
- 在两个开源项目 SER 和 OpenSER 中作出选择
- 描述对它们的使用方案
- 辨别出 openser.cfg 文件中的不同的区段
- 描述 SIP 消息的处理过程
- 辨别松散路由和严格路由
- 辨别 SIP 和 SDP

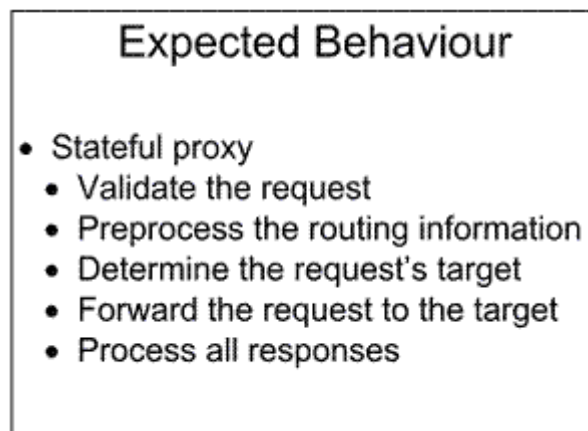
我们在哪儿？（Where Are We?）

VoIP 提供商的解决有很多部件。为了能够对各个部件的联系保持一个整体的把握，我们将在每一个章节中展示下面这张图片。在这一章中，我们的主要的讨论围绕 SIP 代理部件来进行。



SIP 快速路由器是什么？（ What is the SIP Express Router? ）

SIP 快速路由器是一套兼容 IETF RFC3261 sip 协议的开源的 SIP 代理服务器。它的目的是兼容并包尽可能多的应用。



只需要起一个单独的服务，SER 就可以以它“短小精悍”的特点最快速的对请求进行前转，并能够处理成千上万的用户。它被大量的 VoIP 的提供商所使用，

也被使用在处理能力相对较弱的嵌入式 IP PBX 上面。它和其他一些设备的互操作性也使得它成为实际的标准。

用哪个软件，**SER** 还是 **OpenSER**？（**What software to use, SER or OpenSER?**）

SER 最开始是由德国柏林的 FhG Fokus 研究院所开发，发布的时候是遵从 GPL 许可的。它的核心开发人员是 Andrei Pelinescu-Onciul, Bogdan-Andrei Iancu, Daniel Constantin Mierla, Jan Janak,

还有 Jiri Kuthan。之后，其他的一些人也为此作出了贡献，他们是 Juha Heinanen(RADIUS, ENUM, DOMAIN, URI), Greg Fausak (POSTGRES), Maxim Sobolev

(NATHELPER), Adrian Georgescu (MEDIAPROXY), Elena Ramona Modroiu

(XLOG, DIAMETER, AVPOPS, SPEEDDIAL), Miklos Tirpak (Permissions),

等其他人。

OpenSER 是 SER 项目的衍生品。2004 年 FhG Fokus 进行了 SER 项目的副产品

的开发创立了 iptel.org。2005 IPtel 的商业变种被卖给了 TEKELEC。核心开发团队一分为二。他们中的三位成员去了 iptel.org (Andrei Pelinescu-Onciul, Jan Janak, and Jiri Kuthan)，另外两名成员则离开 FhG，创建了一家叫做 Voice-System 的公司，他们也是 2005 年开始的 OpenSER 项目的主要维护人员。

这本书始于 2005 后半年，基于 SER 项目。那个时候，我对使用 SER 进行 NAT 穿透的解决方案很感兴趣。Asterisk 的可伸缩性对于（host??）SIP 提供商不是足够的好，所以我转而投向 SER 的研究中。它的文档真的很难懂，于是乎我开始撰写自己的文档来对 SIP 提供商的管理者们进行培训。

在电子书完成后，我发现 SER 项目已经停止维护了。大部分的代码还停留在 2003 年。经过一点研究我找到了 OpenSER 项目。它似乎更有活力，它有更加新的模块，更频繁发行的版本。我于是在非常短的时间内将所有的东西转向了 OpenSER。

我不想陷入 SER VS OpenSER 的争论中。这样的争论是毫无意义的。现在的事实就是，这本书是为 OpenSER 而写的。

OpenSER 为第三方应用程序提供了一个灵活的可插入模型。应用程序可以被很容易的创建并插入服务器中。这种可插入模型给予了一些新的模块的开发，譬如 RADIUS, DIAMETER, ENUM,

PRESENCE 还有 SMS。更新的模块每个月都会被添加进来。你可以在 <http://www.openser.org/docs/modules/1.2.x> 上查看 OpenSER 1.2.x 支持的模块。

它的高效和健壮使得 OpenSER 能够被用来为数百万的用户提供服务。在最近的 2007 3 月 14 号的性能报告中，OpenSER 1.2.x 能够处理相当于 400 万用户的注册请求。TM（事务模块）能够每小时处理 2800 万通通话。完整的报告可以在 <http://www.openser.org/docs/openser-performance-tests/> 上面找到。

OpenSER 不仅仅被服务提供商所使用。它还可以构造 SIP 应用。目前有一些 SIP 防火墙（SIP firewall），会话边缘处理器（Session Border Controller），和负载均衡的代码都是从 OpenSER 项目中借鉴的。LINKSYS 选择 OpenSER 作为它的一款 PBX 的平台，可能就是因为它的资源耗用少但性能高的特性吧。

OpenSER 灵活，移植性好并且可以扩展。用 ANSI C 开发的它能够被轻易的移植到任何平台上。使用 C 语言能够很容易的创建出新的模块用于扩展。近来，编程中的一些新的层次被添加了进来。使用呼叫处理语言（Call Processing Language）简化路由脚本，使用 Perl 实时的对请求进行处理都成为可能。WeSIP 是一种应用程序的编程接口，它允许你使用 Java 和 Servlets 创建 SIP 应用服务对 OpenSER 服务器进行扩展。可以在 www.wesip.com 上查看 WeSIP。

使用方案（Usage Scenarios）

OpenSER 主要用来作为 SIP 代理和注册服务器。但是，它也可以被用于其他的一些应用当中，比如代理分发器（Proxy dispatches），Jabber 网关（Jabber Gateway），与媒体网关和 RTP 代理合作来进行 NAT 穿透等。支持 IPv4 和 IPv6 并且能够支持多域。OpenSER 可以被应用在 Linux，Solaris，还有 FreeBSD 等平台。

OpenSER 本身的创建是为了当作 SIP 代理服务器使用的。然而，利用它的新模块，如今，OpenSER 能够被用在如下的一些方案中：

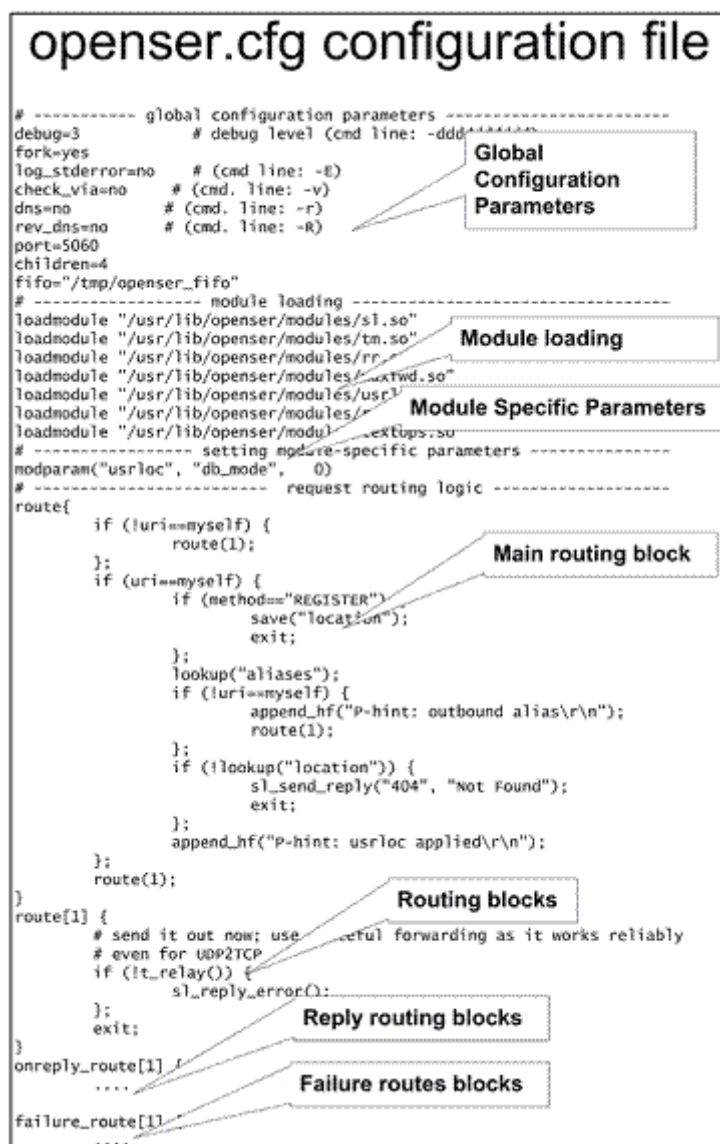
Modules	Functionality
DISPATCHER,PATH	Load balancing
MEDIAPROXY,RTPPROXY,NATHELPER	Nat Traversal

PRESENCE IMC XMPP	Presence Server Instant Messaging
----------------------	--------------------------------------

让我们看看 OpenSER 的大多的使用场景吧。在所有这些场景中，OpenSER 就像胶水一样，将所有的 SIP 部件粘在一起。

- VoIP 服务提供商
- 即时消息服务提供商
- SIP 负载均衡
- 嵌入式 IP PBX
- NAT 穿透
- SIP.EDU

OpenSER 框架 (OpenSER Architecture)



核心和模块（**Core and Modules**）

OpenSER 建造在一套核心之上，这套核心负责基本的功能实现以及对 SIP 消息进行处理。模块则负责 OpenSER 的大半的功能实现。模块和在脚本中使用的命令和参数一起将他们的功能性暴露在 OpenSER 当中。在一个叫做 openser.cfg 的文件中我们对 OpenSER 进行配置。这个配置文件控制着哪个模块被加载以及他们对应的参数。所有的 SIP 流程也都在此文件中定义的一些流程块中被控制。Openser.cfg 是 OpenSER 的主要的配置文件。

Openser.cfg 文件中的各个区段（**Sections of the File**

openser.cfg）

Openser.cfg 文件有七个区段：

- 全局定义 (Global definitions)：文件的这一部分包含了 OpenSER 的几个工作参数，包括 SIP 服务的监听 ip 端口对和 debug 等级。
- 模块 (Modules)：包含了外部库的列表，这些外部库是核心所没有的但却是能够展现其功能的。模块的加载使用 loadmodule。
- 模块配置 (Modules configuratio)：模块有一些参数是需要被合适的设置的。这些参数可以使用 modparam (modulename, parametername,parametervale) 进行配置。
- 主路由块 (Main routing block)：主路由块是进行 SIP 消息处理的开始之处。它控制着所有收到的消息的处理。
- 次要路由块 (Secondary routing blocks)：管理员可以使用 route () 命令来定义新的路由块。这些路由块就像是 OpenSER 脚本中的子程序一样。
- 处理响应路由块 (Reply routing blocks)：响应路由块白被用来处理响应消息，通常是 200 ok。
- 处理出错路由块 (Failure routing blocks)：处理出错路由块用来处理一些出错情况如线路繁忙 (busy) 或是超时 (timeout)。

注：这个文件的细节将在 4，5，6，7，8，9 章节中详细进行描述。

会话，对话和事务 (**Sessions, Dialogs, and Transactions**)

理解一些在 OpenSER 处理过程中使用的 SIP 概念是很重要的：

- SIP 事务 (SIP transaction)：包括一条 sip 消息或任何重发的和对他们的直接响应消息 (如，REGISTER 和 200 OK)
- SIP 对话 (SIP dialog)：两个 SIP 实体之间存在一段时间的关系。如，两个 UAC 之间由 INVITE 消息到 BYE 消息这段时间建立的对话)
- SIP 会话 (SIP session)：在两个 SIP 实体之间的一通媒体流 (音频/视频/文本)

openser.cfg 消息处理

openser.cfg 是为了处理收到的 sip 消息而执行的一段脚本。例如：如果用户 A 想要和用户 B 进行通话，它就要向 B 发送 INVITE 消息。这个消息在主路由块中被处理。这个处理过程一直要延续到它找到 `t_relay ()`（前转）或是 `s1_send_reply`（发送出错信息）或是最终在块的末尾使用 `exit ()` 命令丢弃该消息。billing

SIP 代理——期望的行为（SIP Proxy—Expected Behavior）

按照 RFC3261 中描述的 SIP 代理的基本的处理过程是非常重要的。如果不能很好的理解，将很难去配置代理服务器。

每一个代理在将请求消息发送到下一个部件时会进行路由抉择，并对请求消息作些修改。响应消息将沿着请求消息走的路线原路路由回同样的一组代理。

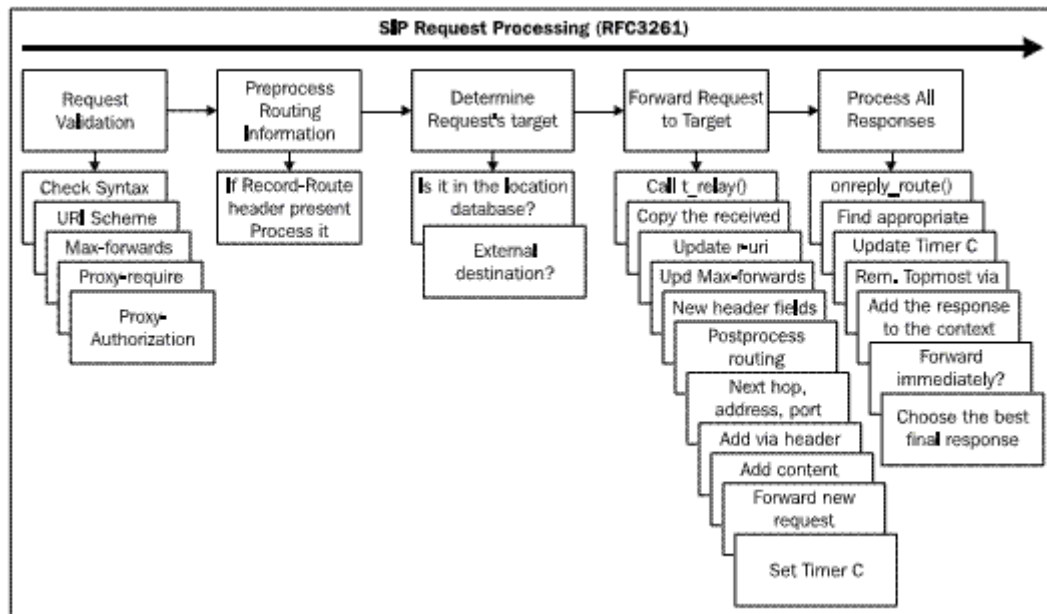
代理服务器既可以运作在有状态模式下也可以以无状态的模式运行。当 SIP 代理服务器只是被当作一个简单的 SIP 包前转器（forwarder）工作时，它只是按照请求消息的要求将消息包前转到一个单独的部件上。无状态模式工作的代理会丢弃它所前转的消息的任何信息。而这个特性限制则限制了对错误的处理和对费用的记录。

如果 OpenSER 知道 200 Ok 是和一个特定的 INVITE 相对应，那么我们就说它此时工作在有状态模式下。这意味和你现在可以在 `onreply_route()` 块中来对响应消息进行管理。而无状态下的消息处理过程不会有上下文的处理方式。无状态处理过程通常被用在类似负载均衡的应用中；在脚本中使用 `forward ()` 命令来处理。

当你需要更加复杂的资源如计费，呼叫前转，voicemail 等时，有状态处理方式是你所需要的。每个事务都将在内存中被维护，并且出错信息，响应信息和重传信息都将和这些事务有所联系。有状态事务由 TM（transaction module）处理，通常使用 `t_relay ()` 命令。

一个经常会被误解的概念是：所谓处理过程的有状态指的是事务而不是对话。因此，一条 INVITE 请求消息的有状态处理过程的结束是到收到 200 OK 响应为止，而不是收到 BYE。

状态操作（Stateful Operation）



这只是对有状态操作的简单描述。你会在 RFC3261 中找到更加完整和详尽的描述。openser.cfg 和上图有些相似的地方。有些过程是手工完成的，如检查 Max-forwards 头域，而其他一些过程则在一条命令中即可完成。更好的说明就是，当你调用 `t_relay()` 时，所有的就像描述中的前转请求处理过程都将自动完成。

当处在有状态模式下是，代理只是一个简单的 SIP 事务处理器而且下面的这些处理步骤都是需要的：

- 验证请求
- 预处理路由信息
- 决定请求的目的地
- 前转请求到达目标
- 处理所有响应消息

有状态代理为每一个得到的请求创建一个新的服务器事务（server transaction）。该请求的任何重传都会被该服务器事务所处理。

举个例子：对于每一个经过我们 SIP 代理服务器的请求，我们都会：

第一步：请求验证

- 检查消息大小，避免缓存溢出
- 检查 Max-forwards 头域以检测出是否发生回环（loops）

第二步：路由信息的预处理

- 如果有 record-route 头，处理之

第三步：决定请求的目的地

- 目的地在定位数据库中么？（对于注册用户来说）
- 可以路由到目的地么？（网关目的地）
- 能够到达外部的域么？（外部地址）

第四步：请求前转

- 调用 t_relay（）函数，OpenSER 将在有状态的模式下处理所有的工作任务。

第五步：响应消息的处理过程

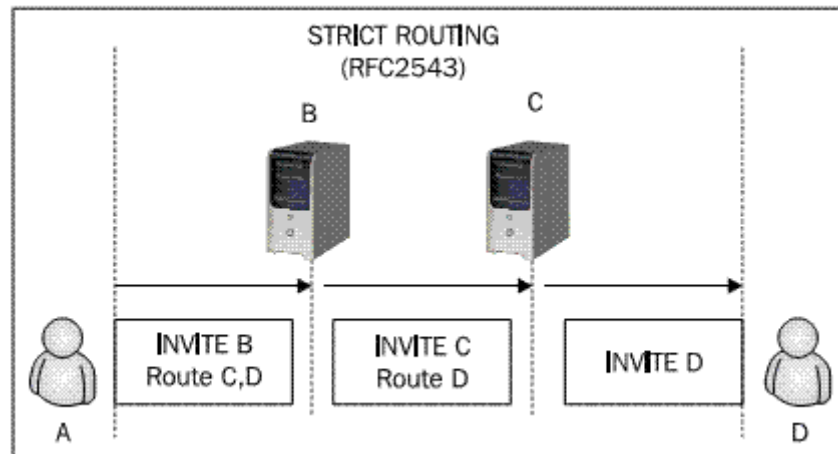
- 通常这个过程会被 OpenSER 自动完成。有时候你可以使用 onreply_route[] 区段对一些响应作出处理。譬如：在“忙则呼叫前转”（call forward on busy）的情形中，我们可以使用 486 响应消息将通话转向 voicemail 服务器。

严格路由和松散路由之间的区别（**Differences between Strict Routing and Loose Routing**

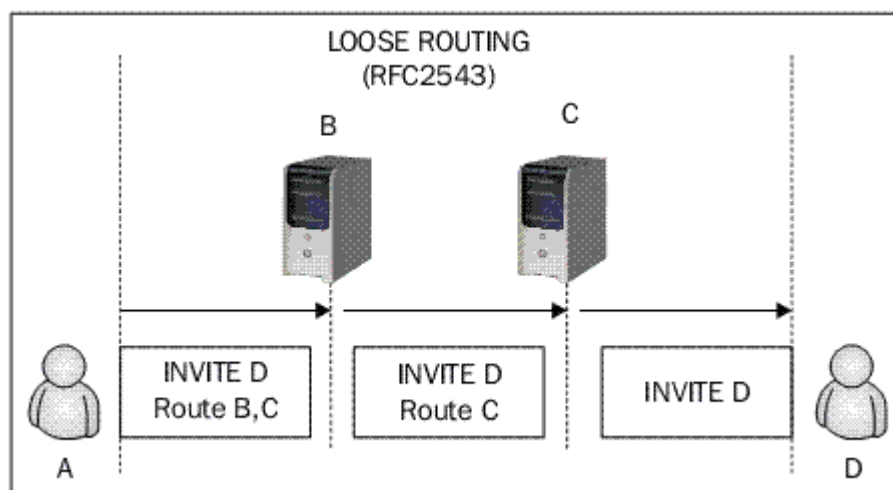
)

在路由 SIP 消息的过程中，有松散和严格两种不同的方法。松散路由是 SIP 版本 2 中的新方法。当使用松散路由时，R-URI 从来都不会被改变，这种方法之前的旧方法保持向后兼容。（严格路由 RFC2543）

严格路由的问题是开始对话前的初始化请求消息时指定所有的代理集合的过程。这种处理过程抛弃了包含在得到的 R-URI 中的信息。带有带外代理（outbound-proxy）的 UA 的行为是不确定的。如果其中的一个部件发生错误那么整个系统都将会出错。



解决办法就是松散路由。这种方法将目标从路由信息中分离。允许每一个目的地来路由消息包，并且有机制能够保证和严格路由向后兼容。参数；lr的使用就是对松散路由的支持象征。



当 SIP 服务器收到一个消息，它可以决定它自己是否处在中间位置。也就是说，如果 SIP 服务器不愿意待在中间位置，那么它就将信息传递给用户代理的 UA 让他们能够能够连接起来。然后消息就将在两个用户代理间进行处理。

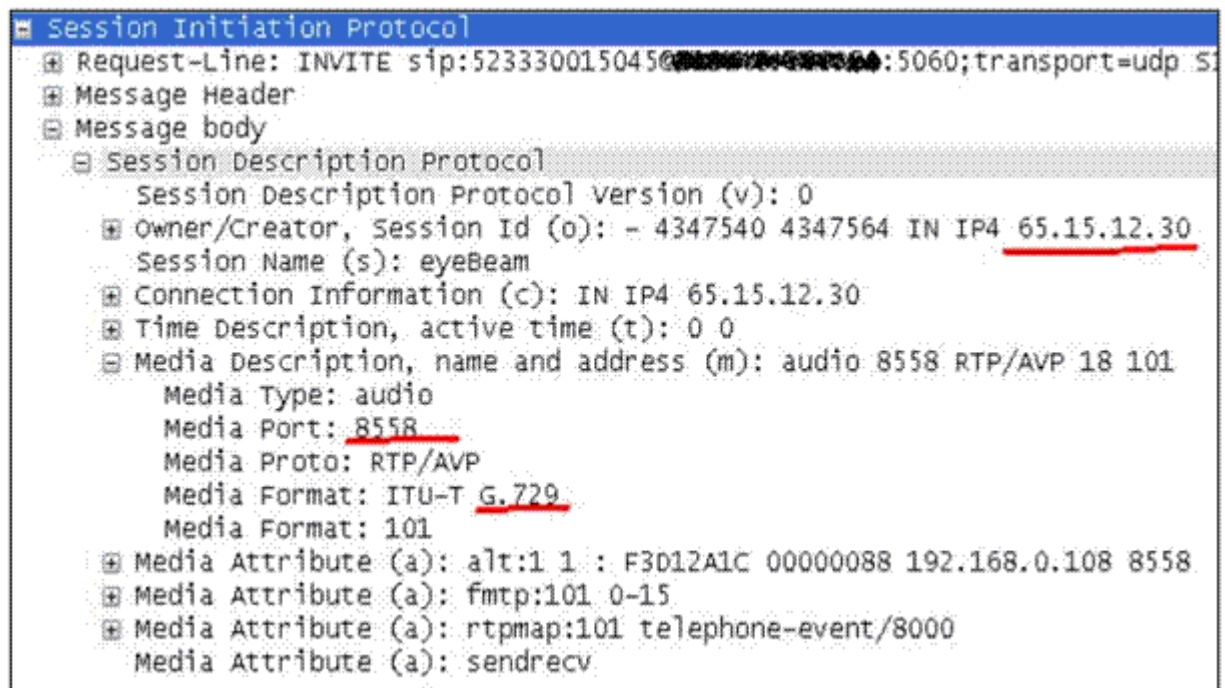
而如果它想要待在中间，那么就应该使用函数 `record_route()` 插入 ROUTE 头。

理解 SIP 和 RTP (Understanding SIP and RTP)

在理解接下来的子区段之前，你应该要懂得一些关于 SIP 和 RTP 的知识。首先，SIP 是一个使用 INVITE, BYE, 和 CANCEL 方法来控制通话的信号协议。在 INVITE 请求消息中的关于会话（音频/视频/文本）消息包含在 SIP 协议中，使用的是叫做 SDP（Session Description Protocol）的协议来包含的这些信息。包含在 SDP 中的信息描述了两个用户代理间的一个或是更多的媒体流的配置情况。

代理服务器从不参与到媒体流中，因此他对于媒体是什么都不用了解的。换句话说，无论 UA 和网关指定的媒体是什么格式，它都支持。但是有时候，B2BUA（back to back user agent）如媒体代理，可以安装在同一个服务器上来处理 RTP 音频（也就是 NAT 穿透机制）。SDP 协议是供给/应答模型（Offer/Answer model）。SDP 供给信息嵌入在 INVITE 请求消息中而应答信息在在 200 OK 的响应消息中。

举例：摘自 Ethereal：



上图中的包是一个 INVITE 请求消息。该请求消息中嵌入了描述会话信息的 SDP 包。我们可以看到这是 eyeBeam 软电话产生的 INVITE 消息。它提供使用 G.729 编码格式，并使用 UDP 的 8558 端口（为了安全起见，我隐藏了 IP 地址）。属性 rtpmap:101 telephone-event/8000 描述了使用的 DTMF 前转方法（RFC2833）。另一个设备，在这个例子中是一个网关，在 200 OK 的回复中对“供给”进行了应答。

```
⊕ Status-Line: SIP/2.0 200 OK
⊕ Message Header
⊕ Message body
  ⊕ Session Description Protocol
    Session Description Protocol version (v): 0
    ⊕ Owner/Creator, Session Id (o): IPCOM-NEXTONE 1234 865833 IN IP4 [REDACTED]
      Session Name (s): sip call
    ⊕ Connection Information (c): IN IP4 [REDACTED]
    ⊕ Time Description, active time (t): 0 0
    ⊕ Media Description, name and address (m): audio 46190 RTP/AVP 18
      Media Type: audio
      Media Port: 46190
      Media Proto: RTP/AVP
      Media Format: ITU-T G.729
    ⊕ Media Attribute (a): cpar: a=T38FaxUdpEC:t38UDPRedundancy
    ⊕ Media Attribute (a): cpar: a=T38FaxMaxDatagram:176
    ⊕ Media Attribute (a): cpar: a=T38FaxMaxBuffer:336
    ⊕ Media Attribute (a): cpar: a=T38FaxRateManagement:transferredTCF
    ⊕ Media Attribute (a): cpar: a=T38MaxBitRate:14400
    ⊕ Media Attribute (a): cpar: a=T38FaxVersion:0
    ⊕ Media Attribute (a): cdsc:1 image udpt1 t38
    ⊕ Media Attribute (a): sqn:0
    ⊕ Media Attribute (a): ptime:30
```

概要（Summary）

这一章中，我们学到了什么是 OpenSER 以及它的主要特性。现在，你可以识别出 openser.cfg 配置文件和它的一些配置块，如全局定义（global definitions），加载模块（load modules），模块的参数，主路由块，路由块，响应路由块和出错处理路由块。每一个代理接收的请求都按照 openser.cfg 脚本中的配置来进行处理。脚本的内容的组织几乎和 SIP 有状态代理处理过程相一致。通常 OpenSER 作为松散路由器来运行（SIP 版本 2）。最后我们介绍了 SIP 和 SDP 的概念。

第三章：OpenSER 安装（OpenSER Installation）

安装是所有工作的开始。能够利用源代码正确的安装 OpenSER 是非常重要的。使用

Debian 包或 apt-get 工具安装的速度能够快很多。但是利用源代码安装可以更加灵活，能

够允许你对需要什么样的模块的编译进行选择。你不可能用 Debian 包安装对 RADIUS

accounting 的支持。这就是我们为什么不使用任何所谓的捷径来进行安装的原因。另外，

我强烈建议安装在 Debian 平台上。

如果你选择安装在其他的平台上，那么你必须要对初始化脚本和其他包进行修改和处

理。

在本章节结束的时候，你将能够：

- ☐ 安装 Linux，为 OpenSER 的安装做准备
- ☐ 下载 OpenSER 源代码和一些依赖
- ☐ 编译并安装 OpenSER 及其对 MySQL 和 RADIUS 的支持
- ☐ 启动停止运行 OpenSER
- ☐ 配置 Linux 系统使 OpenSER 在系统启动时运行

所需硬件（Hardware Requirements）

对于 OpenSER 来说没有什么所谓的最低硬件配置问题。它可以运行在普通的 PC 上

面。我们最大的惊喜来源于始于 1.2 版进行的性能测试。一台拥有如下配置的 PC 能够支持

每小时 28,000,000 路的完整通话。测试服务器是一台普通的台式机，Intel Core2 CPU

6300 @1.86GHz，1GB 的内存，100Mbps 的以太网卡。不幸的是，到目前为止，还没有公

式可以对 OpenSER 进行测量。到底什么才是好的正确的硬件选择是依靠经验判断的。

所需软件（Software Requirements）

OpenSER 软件可以运行在各种不同的平台上，Linux，BSD，Solaris 等。有些包对于

Linux 和 Solaris 的一些变种也是需要的。这些包可以从 www.openser.org 上下载到。下面的包是编译 OpenSER 所必须的。

- ☐ Gcc
- ☐ bison 或 yacc（伯克利 yacc）
- ☐ flex
- ☐ GNU make
- ☐ GNU tar

MYSQL, POSTGRES, RADIUS 等一些其他的模块还需要一些附加的包才能进行编

实验 (Lab) ——安装 Linux 为 OpenSER 作准备

Linux 的分支 Debian Etch 可以在 http://cdimage.debian.org/debian-cd/4.0_r0/i386/ 下载到。

步骤 1: 插入光盘 CD, 使用 Debian Etch 4.0 的 CD 启动计算机。按下 ENTER 键开始

在屏幕显示上面的图示时，你也可以对启动和安装选项进行配置。有时候，你需要为

步骤 2: 选择语言。

在欧洲和亚洲的一些国家，选择键盘布局是比较常见的事情。

选择服务器的名字。这一步很重要，因为之后你需要使用这个名字来访问服务器

别。所以，请确定保证对这一屏的选择作出正确的回答。

关于分区我们可以写满一整章的内容。Linux 的熟练使用者们当然在这里会选择手动选

器去向一位 Linux 专家咨询最好的分区方法。

现在，只需要选择用来安装 Linux 的磁盘即可。

你得再一次的选择如何对系统进行分区。我们仍然坚持使用默认安装选项。高级用户

可以自己进行一些改变。

步骤 9：完成磁盘分区。

现在，才只是完成分区的步骤并将这些改变写入磁盘。如果你想要保持你的磁盘不

变，那么作上面这些。因为完成了分区，所有磁盘之前存在的内容都将会被删除。所

以，小心点为妙。我使用的是 VMWare 来对 OpenSER 做的实验；这样操作起来就比较自

步骤 8：选择将所有的文件

不要由，

。因为创建的是一个虚拟的机器，这样我可以在上面安全的来完成我的工作。

步骤 10：将这些改变写入磁盘

这个部分是比较“恐怖”的。一定要确定你想要删除磁盘上的所有内容。ok，在选择

YES 之前还是请多思考两遍吧。

| 警告： |

| 磁盘上所有的数据都将被销毁 |

步骤 11：设置时区。选择时区。拥有正确的时区是很重要的，主要是用来进行报告。如果你没有正确的选

择，那么最后 voicemail 信息的时间将是错误的。

步骤 12：给 openser 设置 root 用户密码。

为你的 root 用户选择一个密码。这是系统上最重要的密码。

步骤 13：再次输入密码进行确认。请再次输入密码进行确认。尽量使用一个很难被破解的密码（最少 8 个字符，字母，

数字

称开始。

步骤 15：为 openser 用户帐号输入用户名。

和一些其他的特殊字符，如“*”或“#”）。

步骤 14：为 openser 用户帐号输入全名。

有些系统需要你创建至少一个用户。让我们使用一个完整的用户名这个名字是我们用来登录系统的用户名。

步骤 16：为 openser 用户帐号输入密码并再次输入以进行确认。

选择 Yes 使用镜像。

输入密码并进行确认。记住，要使用难以破解的密码。

步骤 17：设置安装包管理器。在安装的过程中，我们将使用到由 Debian 发行的一些包。

步骤 18：选择一个镜像国。

这一屏是让你选择到底从什么地方下载安装包。

步骤 19：选择 ftp.debian.org 或你所倾向的镜像。选择最近的一个下载地能够加速这些包的下载。

步骤 20：将 HTTP 代理留空，或如果你有使用 HTTP 代理，那么填入合适的参数。

数以

流行度调查，那么选择 Yes，否则选择 No。

如果你使用诸如 Squid 或 Microsoft ISA Server 等的 HTTP 代理，那么请填入合适的参

使得你能够访问到互联网进行下载安装包。

步骤 21：如果你想要参与包的包的流行度调查将产生一些关于那些被下载最多的包的数据。

步骤 22：选择标准版系统。

Debian 有一些预定义的安装版如“桌面版”（Desktop）。以桌面版为例，它将为 Linux

安装 GNOME 或 KDE 的 GUI。我们的安装不需要 GUI，所以请选择标准版系统（Standard

syst

s 安装 GRUB 引导程序。

em）。之后我们将手动安装 Web Server，Mail Server，和 SQL 数据库。

步骤 23：选择 YeGRUB 是你的服务器的引导程序管理器。允许你进行双系统引导并在引导的过程中作

一些技巧性的操作。

步骤 24：完成安装。

完成安装，引导系统。

系

下载并安装 OpenSER 1.2 版（Downloading and Installing

系统将自动重启。

步骤 25：重启后安装 SSH。

`sudo apt-get install ssh OpenSER v1.2`）

容易，我们也会描述整个编译过程。这个过程更

加灵活，在这份材料中我们可能需要重新编译 OpenSER 数次来包含其他的一些模块。安

装过

`on flex make openssl libmysqlclient-dev`

`librt-devel mysql-server`

不是真正的依赖，但是我们在这里安装它 |

将使事情变得容易些。 |

`d /usr/src`

`wget http://www.openser.org/pub/openser/1.2.2/src/openser-1.2.2-tls_`

`src.t`

“exclude_modules?”行移除 mysql 和任何 radius 相关的模块。这样将是编译过程包

括 M

`d /usr/src/openser-1.2.2-tls/`

`i Makefile`

变前的文件内容如下：

`xclude_modules?= jabber cpl-c mysql pa postgres osp unixodbc \`

avp_radius auth_radius \

即使使用 Debian 包安装 OpenSER 更
程将一步一步的列举如下：

步骤 1：安装依赖。

```
apt-get install gcc binutils  
radiusclient-ng2 libradiusclient-  
| MySQL server  
|
```

步骤 2：下载源代码包并解压。

c

ar.gz

```
tar -xzf openser-1.2.2-tls_src.tar.gz
```

步骤 3：使用你最喜欢的 Linux 编辑器来编辑 Makefile

从

MySQL 和 RADIUS。

c

v

改

e

```
group_radius uri_radius xmpp \  
presence pua pua_mi pua_usrloc \  
mi_xmlrpc perl snmpstats
```

变后的文件内容如下：

```
exclude_modules?= jabber cpl-c pa postgres osp unixodbc \  
xmpp \  
presence pua pua_mi pua_usrloc \  
mi_xmlrpc perl snmpstats
```

步骤 4：编译安装核心和模块

实验（——在 Linux 引导时运行 OpenSER

中包含 openSER

```
cd /usr/src/openser-1.2.2-tls/packaging/debian
```

改

e

步

```
cd openser-1.2.2-tls
```

```
make prefix=/ all
```

```
make prefix=/ install
```

步骤 5：作出需要的调整：

```
mkdir /var/run/openser
```

Lab)

步骤 1：在 linux 引导

c

```
cp openser.default /etc/default/openser  
it.d/openser
```

.cfg 文件并移除 fork=no 行（即使有 c 风格的注释）

初始

nser.init 有必要的权限。

```
chmod 755 openser
```

步骤 4：编辑 /etc/default/openser.cfg，将内存参数改为 128MB，并将
RUN——OPENSER 改为 yes。

步骤 5：编辑初始化脚本确定守护进程指向正确的文件夹

```
vi /etc/init.d/openser
```

编辑之前的文件内容：

```
DAEMON=/usr/sbin/openser
```

编辑后的文件内容：

```
DAEMON=/sbin/openser
```

步骤 6：重启计算机观察 OpenSER 是否启动，确认刚才的改动是否成功。

```
ps -ef | grep openser
```

| 强烈推荐更改你的用户名和密码来使用/etc/init.d/openser 文件 |

| 运行 openser。 |

```
cp openser.init /etc/in
```

```
update-rc.d openser defaults 99
```

步骤 2：编辑/etc/openser/openser
化脚本将寻找 fork=no，即使被注释。

步骤 3：确保脚本 ope
cd /etc/init.d

OpenSER v1.2 目录结构

在安装完成后，OpenSER 将创建文件安放架构。了解这个架构对于定位系统存储在哪一个主文件夹中是很重要的。你需要这些消息来更新或删除软件。

配置文件（**etc/openser**）

```
openser-1:/etc/openser# ls -l
```

```
total 12
```

```
-rw-r--r-- 1 root root 1804 2007-09-10 14:02 dictionary.radius
```

```
-rw-r--r-- 1 root root 4077 2007-09-10 14:05 openser.cfg
```

```
-rw-r--r-- 1 root root 1203 2007-09-10 14:02 openserctrlccd
```

模块（**/lib/openser/modules**）

```
openser-1:/lib/openser/modules# ls
```

```
acc.so      domain.so   msilo.so    sms.so
```

```
alias_db.so  enum.so     mysql.so     speeddial.so
```

```
auth_db.so   exec.so     nathelper.so sst.so
```

```
auth_diameter.so flatstore.so options.so   statistics.so
```

```
auth_radius.so gflags.so   path.so      textops.so
```

```
auth.so      group_radius.so pdt.so       tm.so
```

```
avpops.so      group.so      permissions.so uac_redirect.so
avp_radius.so  imc.so       pike.so       uac.so
dbtext.so      lcr.so       registrar.so  uri_db.so
dialog.so      mangler.so   rr.so         uri.so
dispatcher.so  maxfwd.so   seas.so       usrloc.so
diversion.so   mediaproxy.so siptrace.so   xlog.so
domainpolicy.so mi_fifo.so   sl.socd /lib/openser/modules
```

二进制文件（ **/sbin** ）

```
openser-1:/sbin# ls -l op*
```

```
-rwxr-xr-x 1 root root 2172235 2007-09-10 14:02 openser
```

```
-rwxr-xr-x 1 root root 41862 2007-09-10 14:02 openserctl
```

```
-rwxr-xr-x 1 root root 38107 2007-09-10 14:02 openser_mysql.sh
```

```
-rwxr-xr-x 1 root root 13562 2007-09-10 14:02 openserunixcd /sbin
```

日志文件（ **Log Files** ）

初始化日志可以在 syslog 文件中查看到（ /var/log/syslog ）：

```
Sep 10 14:25:56 openser-1 openser: init_tcp: using epoll_it as the io
watch
```

method (auto detected)

Sep 10 14:25:56 openser-1 /sbin/openser[7791]: INFO: statistics manager

successfully initialized

Sep 10 14:25:56 openser-1 /sbin/openser[7791]: StateLess module -
initializing

Sep 10 14:25:56 openser-1 /sbin/openser[7791]: TM - initializing...

Sep 10 14:25:56 openser-1 /sbin/openser[7791]: Maxfwd module-
initializing

Sep 10 14:25:56 openser-1 /sbin/openser[7791]: INFO:ul_init_locks: locks
array

size 512

Sep 10 14:25:56 openser-1 /sbin/openser[7791]: TextOPS - initializing

Sep 10 14:25:56 openser-1 /sbin/openser[7791]: INFO: udp_init:
SO_RCVBUF is

initially 109568

Sep 10 14:25:56 openser-1 /sbin/openser[7791]: INFO: udp_init:
SO_RCVBUF is

finally 262142

Sep 10 14:25:56 openser-1 /sbin/openser[7791]: INFO: udp_init:
SO_RCVBUF is

initially 109568

Sep 10 14:25:56 openser-1 /sbin/openser[7791]: INFO: udp_init:
SO_RCVBUF is

finally 262142

Sep 10 14:25:56 openser-1 /sbin/openser[7792]: INFO:mi_fifo:mi_child_

init(1): extra fifo listener processes created

启动选项 (Startup Options)

OpenSER 可以使用初始化脚本或使用 `openserctl` 工具进行启动。如果你使用的是初始化脚本启动的 `openser`，那么你也只能使用初始化脚本来终止其运行。同样的道理适用于使用 `openserctl` 工具。

使用初始化脚本启动，终止和重启 OpenSER。

`/etc/init.d/openser start|stop|restart`

使用 `openserctl` 工具脚本启动，终止和重启 OpenSER。

`/etc/init.d/openserctl start|stop|restart`

OpenSER 的执行有几个启动选项。展现于下面的这些选项允许你改变守护进程（DAEMON）的配置。最有用的几个如下：

- “-c”用来查看配置文件
- “-D -E dddddd”用来查看模块加载（不要用于[production ??]，它只是绑定第一个接口）

还有许多其他的选项允许你调整你的配置。对于每一个选项，在你填入的配置文件中都能找到对应的核心参数。

Usage: `openser -l address [-p port] [-l address [-p port]...] [options]`

Options:

-f file Configuration file (default `//etc/openser/openser.cfg`)

-c Check configuration file for errors

-C Similar to '-c' but in addition checks the flags of

exported functions from included route blocks

-l address Listen on the specified address/interface (multiple -l

mean listening on more addresses). The address format

is [proto:]addr[:port], where proto=udp|tcp and

addr= host|ip_address|interface_name. E.g: -I localhost,

-I udp:127.0.0.1:5080, -I eth0:5062 The default

behavior is to listen on all the interfaces.

-n processes Number of child processes to fork per interface

(default: 8)

-r Use dns to check if is necessary to add a "received="

field to a via

-R Same as '-r' but use reverse dns;

(to use both use '-rR')

-v Turn on "via:" host checking when forwarding replies

-d Debugging mode (multiple -d increase the level)

-D Do not fork into daemon mode

-E Log to stderr

-T Disable tcp

-N processes Number of tcp child processes (default: equal to '-n')

-W method poll method

-V Version number

-h This help message

-b nr Maximum receive buffer size which will not be exceeded

by auto-probing procedure even if OS allows

-m nr Size of shared memory allocated in Megabytes

-w dir Change the working directory to "dir" (default "/")

-t dir Chroot to "dir"

-u uid Change uid

-g gid Change gid

-P file Create a pid file

-G file Create a pgid file

-x socket Create a unix domain socket

概要 (**Summary**)

在这一章中你已经学会如何安装 OpenSER 和为了 OpenSER 的安装如何准备 Linux 系统。我们已经下载和编译了 OpenSER 和 MySQL 模块。在安装完后，我们又介绍了如何使用 OpenSER 初始化文件在引导的时候启动 OpenSER。

第四章：OpenSER 标准配置（ OpenSER Standard Configuration ）

OpenSER 的标准配置文件安装在/etc/openser/openser.cfg。它是 OpenSER 的最简单的配置文件之一。而且它还是一个可以开始进行解释 OpenSER 功能的理想脚本。和一些基本的模块，参数和函数一样，还有一些区段是你应该熟悉了解的。

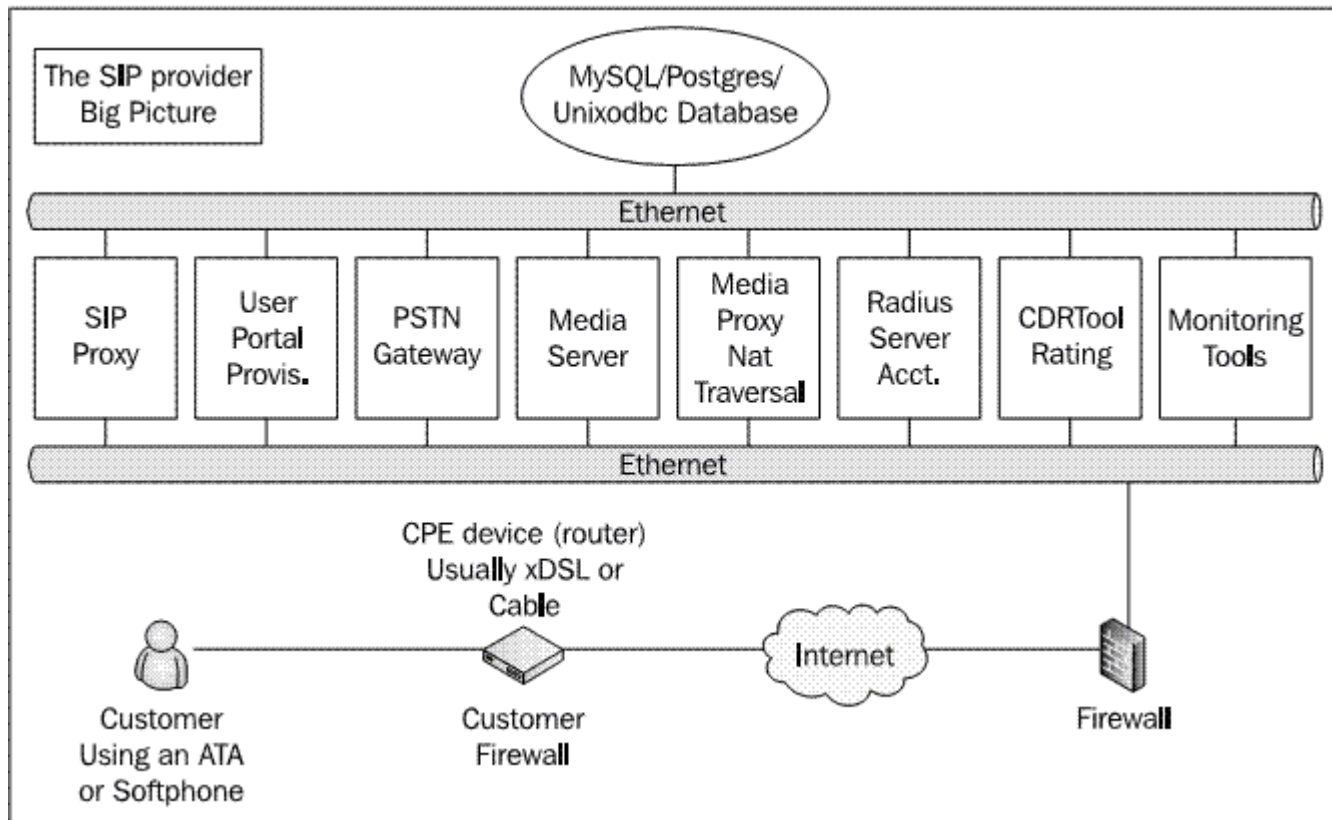
这个章节结束后，你将能够：

- 识别出 openser.cfg 配置文件的各个区段
- 识别出标准配置的限制
- 使用 ngrep 工具跟踪 SIP 事务
- 使用 XLOG 模块记录路由过程
- 使用 append_hf 命令标记 ngrep 工具跟踪的包

标准配置是一个好的出发点。它拥有最小化的功能集合，不支持验证，所以你可以不使用密码就可以连接你的 SIP 话机。不管怎样，你都可以从一部话机呼向另一部，我们将在之后测试之。

我们在哪？（ Where Are We？ ）

我们要再一次提到的是，VoIP 服务提供商的解决方案中有许多的部件。为了失去对整体的把握，我们会在大多数的章节中展示这幅图片。在这一章中，我们仍然会与标准配置的 SIP 代理部件打交道。



分析标准配置（ **Analyzing the Standard Configuration** ）

下面是 OpenSER 1.2.2 版本的标准配置的展示。在这一节，我们将开始描述标准配置中的每一行的命令和函数。

```
#

# $Id: openser.cfg 1676 2007-02-21 13:16:34Z bogdan_iancu $

#

#simple quick-start config script

#Please refer to the Core CookBook at http://www.openser.org/dokuwiki/doku.php

#for a explanation of possible statements, functions and parameters.

#
```

```
# ----- global configuration parameters -----

debug=3      # debug level (cmd line: -dddddddddd)

fork=yes

log_stderr=no  # (cmd line: -E)

children=4

port=5060

#uncomment the following lines for TLS support

#disable_tls = 0

#listen = tls:your_IP:5061

#tls_verify_server = 1

#tls_verify_client = 1

#tls_require_client_certificate = 0

#tls_method = TLSv1

#tls_certificate = "//etc/openser/tls/user/user-cert.pem"

#tls_private_key = "//etc/openser/tls/user/user-privkey.pem"

#tls_ca_list = "//etc/openser/tls/user/user-calist.pem"

# ----- module loading -----

#set module path

mpath="/lib/openser/modules/"

#Uncomment this if you want to use SQL database

#loadmodule "mysql.so"

loadmodule "sl.so"

loadmodule "tm.so"
```

```
loadmodule "rr.so"

loadmodule "maxfwd.so"

loadmodule "usrloc.so"

loadmodule "registrar.so"

loadmodule "textops.so"

loadmodule "mi_fifo.so"

# Uncomment this if you want digest authentication

# mysql.so must be loaded !

#loadmodule "auth.so"

#loadmodule "auth_db.so"

# ----- setting module-specific parameters -----

# -- mi_fifo params --

modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")

# -- usrloc params --

modparam("usrloc", "db_mode", 0)

# Uncomment this if you want to use SQL database

# for persistent storage and comment the previous line

#modparam("usrloc", "db_mode", 2)

# -- auth params --

# Uncomment if you are using auth module

#

#modparam("auth_db", "calculate_ha1", yes)

#
```

```

# If you set "calculate_ha1" parameter to yes (which true in this
config),

# uncomment also the following parameter)

#

#modparam("auth_db", "password_column", "password")

# -- rr params --

# add value to ;lr param to make some broken UAs happy
modparam("rr", "enable_full_lr", 1)

# ----- request routing logic -----

# main routing logic

route{

    # initial sanity checks -- messages with

    # max_forwards==0, or excessively long requests

    if (!mf_process_maxfwd_header("10")) {

        sl_send_reply("483", "Too Many Hops");

        exit;

    };

    if (msg:len >= 2048 ) {

        sl_send_reply("513", "Message too big");

        exit;

    };

    # we record-route all messages -- to make sure that

    # subsequent messages will go through our proxy; that's

```



```

# particularly good if upstream and downstream entities

# use different transport protocol

if (!method=="REGISTER")

    record_route();

# subsequent messages withing a dialog should take the

# path determined by record-routing

if (loose_route()) {

    # mark routing logic in request

    append_hf("P-hint: rr-enforced\r\n");

    route(1);

};

if (!uri==myself) {

    # mark routing logic in request

    append_hf("P-hint: outbound\r\n");

    # if you have some interdomain connections via TLS

    #if(uri=~"@tls_domain1.net") {

        #    t_relay("tls:domain1.net");

        #    exit;

    #} else if(uri=~"@tls_domain2.net") {

        #    t_relay("tls:domain2.net");

        #    exit;

    #}

    route(1);

```

```

};

# if the request is for other domain use UsrLoc

# (in case, it does not work, use the following command

# with proper names and addresses in it)

if (uri==myself) {

    if (method=="REGISTER") {

        # Uncomment this if you want to use digest
        authentication

        #if (!www_authorize("openser.org",
            "subscriber")) {

            #    www_challenge("openser.org", "0");

            #    exit;

            #};

            save("location");

            exit;

        };

        lookup("aliases");

        if (!uri==myself) {

            append_hf("P-hint: outbound alias\r\n");

            route(1);

        };

        # native SIP destinations are handled using our

        USRLOC DB

```

```

        if (!lookup("location")) {

            sl_send_reply("404", "Not Found");

            exit;

        };

        append_hf("P-hint: usrloc applied\r\n");

    };

    route(1);

}

route[1] {

    # send it out now; use stateful forwarding as it works

    # reliably even for UDP2TCP

    if (!t_relay()) {

        sl_reply_error();

    };

    exit;

}

```

这个标准的配置是最简单的可以使用的配置。我们将以它作为出发点，然后再在接下来的章节中循序渐进的包含新的命令和函数。使用这个配置，客户端能够进行注册（不需要认证）并且 UACs 能够互相之间进行沟通。注册服务器，定位服务器和代理服务器使用最小配置进行工作。下面我们将解释一些脚本中的引用。

```
debug=3 # debug level (cmd line: -ddddddddd)
```

设置日志级别（Set log level）：它是一个-3 到 4 之间的一个数。默认为 2。数越大，那么写道日志中的信息就越多。不过如果将其设为 4，那么系统的性能可能会变得很差。日志级别是：

- L_ALERT（-3）——这个级别只被用来报告需要立即响应的错误。

- `L_CRIT (-2)` ——这个级别只被用来报告造成危险状况的错误。
- `L_ERR (-1)` ——这个级别用来报告数据处理过程中不会造成系统故障的错误。
- `L_WARN (1)` ——这个级别用来写入一些警告消息。
- `L_NOTICE (2)` ——这个级别用来报告不寻常的状况。
- `L_INFO (3)` ——这个解别用来写入一些提示信息的消息。
- `L_DBG (4)` ——这个级别用来写入用来调试的消息。

`fork=yes`

`fork` 参数用来定义 OpenSER 进程是运行在前台还是后台。运行在后台则设置 `fork=yes`。有时候你会发现如果在前台运行对于定位脚本错误是非常有用的。如果 `fork` 被禁用的话，OpenSER 将不能够同时监听多个接口，TCP/TLS 支持功能将被自动禁用。在单进程模式下，只有一个 UDP 接口被接受。

`log_stderr=no # (cmd line: -E)`

如果被设为 `yes`，服务器将打印调试信息到标准错误输出。如果设为 `no`, `syslog` 会被使用。

`Children=4`

“`children`”核心参数告诉 OpenSER 每个创建处理接入请求的进程的接口有多少孩子进程。四个进程对于大多数系统来书都是很好的一个出发点。这个参数只适用于 UDP 接口，对于 TCP 没有影响。

`Port=5060`

如果在监听参数中没有被指明，那么这个参数就是默认使用的端口号。

`mpath="/lib/openser/modules/"`

设置模块搜索路径。这可以被用来简化模块的加载。

`loadmodule "sl.so"`

`loadmodule "tm.so"`

`loadmodule "rr.so"`

```
loadmodule "maxfwd.so"
```

```
loadmodule "usrloc.so"
```

```
loadmodule "registrar.so"
```

```
loadmodule "textops.so"
```

```
loadmodule "mi_fifo.so"
```

上面的这些行是 OpenSER 加载外部的模块。这个时候，只有需要的那些模块被加载了。其余附加的功能需要加载其他诸如 RADIUS 和 MYSQL 的模块。所有这些模块都有一个 README 文件来描述其功能。

```
modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
```

创建 FIFO 文件及其名称用来监听和读取外部命令。

```
modparam("usrloc", "db_mode", 0)
```

modparam 核心命令配置了相应的模块。上面的 usrloc 模块负责定位服务。当一个用户注册时，它用来保存定位信息，也就是我们知道的变量 db_mode 表明的相对于位置的 AOR (Address of Record)。为 0 表示是内存。所以如果你关闭了服务器，那么你将丢失所有的注册记录。表的位置靠的就是 db_mode 变量。将 db_mode 设置为 0 表明这个数据将被保存到数据库中。换句话说，如果 OpenSER 关闭，所有记录丢失。

```
modparam("rr", "enable_full_lr", 1)
```

上面的语句设置 rr (Record Routing) 模块的 enable_full_lr 变量为 1。它告诉 OpenSER 要支持那些不对 record_route 头域进行管理的 SIP 客户端。如果设为 1，;lr=on 将被使用而不是;lr。

```
Route {
```

这是 SIP 请求的路由逻辑的开始。“块 (block)”起始于“{”。在这个块中，SIP 请求将被处理。通过下面的语句你将看清大概：

```
# initial sanity checks -- messages with
```

```
# max_forwards==0, or excessively long requests
```

```
if (!mf_process_maxfwd_header("10")) {
```

```
    sl_send_reply("483","Too Many Hops");
```

```

        exit;

};

if (msg:len >= 2048 ) {

    sl_send_reply("513", "Message too big");

    exit;

};

```

当一个请求进入主路由块时，要进行一些检查。

首先检查的是前转（forward）的最大次数。为了避免回环（loop），我们使用 `mf_process_maxfwd_header()` 函数来检查包到底传递了多少 SIP 跳（SIP hops）。如果回环（loop）被发现，那么脚本则使用 `s1_send_reply()` 函数发送“483 too many hops”消息。

`Msg:len` 是 OpenSER 核心用来返回 SIP 请求长度（单位 byte）的函数。这是一个标准的用来检查消息大小限制方法。

```

if (!method=="REGISTER")

    record_route();

```

如果不是 REGISTER 方法，OpenSER 将进行 record-route。这条消息告诉 SIP 服务器要处在两个 UAC 请求的路径当中。

`record_route()` 函数只是简单的增加了一个新的 record-route 头域。

Subsequent messages within a dialog should take the

```

# Path determined by record-routing

if (loose_route()) {

    # mark routing logic in request

    append_hf("P-hint: rr-enforced\r\n");

    route(1);

};

```

loose_route()函数是用来试着看看请求有没有使用 record-route 头域进行路由的。这个函数标识的请求将使用最上面的 record-route 头域的内容进行路由。

如果请求是在同一个对话中，我们将进入 if，并前转该包。我们只是简单的前转该请求。我们将调用次级路由块 route(1)做这件事，在该块中 t_relay()函数将被调用。

按照 record-route 头域(rr-enforced)，append_hf 函数将添加一个表明请求被处理的提示。

```
if (!uri==myself) {

    # mark routing logic in request

    append_hf("P-hint: outbound\r\n");

    # if you have some interdomain connections via TLS

    #if(uri=~"@tls_domain1.net") {

        #    t_relay("tls:domain1.net");

        #    exit;

    #} else if(uri=~"@tls_domain2.net") {

        #    t_relay("tls:domain2.net");

        #    exit;

    #}

    route(1);

};
```

上面的代码将处理一个我们的代理不处理的域中的请求，if(!uri==myself)，使用调用 t_relay 的 route(1)来前转该请求。默认的情况下，代理只是一个开放式的中继器。在下面的章节中我们将讨论怎样改进带外通话（outbound call）的处理过程。将请求前转至其他的代理是很重要的；然而，还应该进行一些合适的标识检查。现在，我们将处理指向被我们的 SIP 代理处理的域的请求。

```
if (uri==myself) {

    if (method=="REGISTER") {
```

```

# Uncomment this if you want to use digest
authentication

#if (!www_authorize("openser.org",
    "subscriber")) {

#    www_challenge("openser.org", "0");

#    exit;

#};

save("location");

exit;

};

```

如果是 REGISTER 请求，使用 save("location") 将 AOR 保存至位置表（location table）中。理解两个概念至关重要。验证被取消（www_authorize 被注释掉）和因为我们没有把数据库安装在 SIP 代理上，所以定位数据库不是持续起作用的。

```

lookup("aliases");

if (!uri==myself) {

    append_hf("P-hint: outbound alias\r\n");

    route(1);

};

```

别名（alias）是可选的 URI（比如，8590@voffice.com.br 可以是原有 URI flavio@voffice.com.br 的别名）。lookup("aliases") 函数寻找在请求中表示的 URI 的典型的 URI。如果该 URI 被找到，则在操作进行前用其取代 R-URI。最终的 URI 既可以被放置在我们域的内部，也可以是外面。如果在外面，那么系统只是简单的将包前转到负责该域的 SIP 代理上。如果在外面则开始请求的处理。

```

if (!lookup("location")) {

    sl_send_reply("404", "Not Found");

```



```

        exit;

};

append_hf("P-hint: usrloc applied\r\n");

```

lookup("location")函数将试着恢复 R-URI 的 AOR。如果 AOR 被定位到 (UA 注册了) 那么 UA 的 ip 地址将替代 R-URI。如果没有找到，我们只是简单的发回错误消息“404 Not Found”。

```

route(1);

如果找到 AOR 我们将使用 route(1)结束；

route[1] {

    # send it out now; use stateful forwarding as it works

reliably

    # even for UDP2TCP

    if (!t_relay()) {

        sl_reply_error();

    };

    exit;

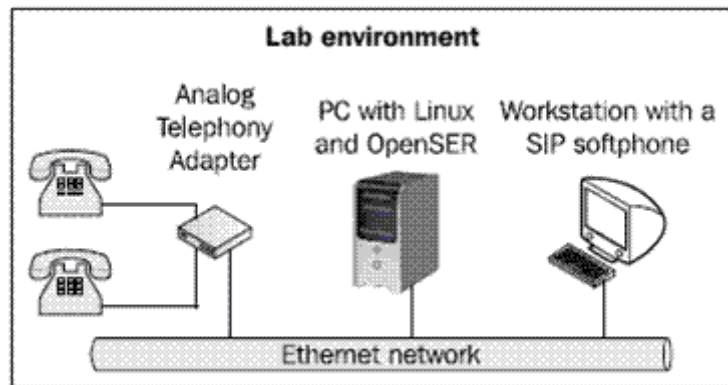
}

```

最后，路由块被调用。t_relay()函数前转基于请求 URI 的有状态的请求。域部分将使用 NAPTR，SRV 和 A records 等来进行解析。这个函数是由 TRANSACTION 模块 (tm.so) 提供的，负责发送请求并处理重发和响应。如果请求不能被成功的被发送到目的地，错误信息将由 t_relay()函数自动产生。如果错误产生，sl_reply_error()将发送响应信息给 UA。

Using 标准配置 (Using the Standard Configuration)

在这个实验中，我们将使用协议分析机来抓取一个完整的 SIP 通话。我们将分析消息头和消息流程。你可以创建一个 PC 和带有两个 UAC 的实验环境。UAC 可以是软电话，ATA，或是 IP 电话。



调整以满足你的实验需要。

1: 使用 ngrep 开始抓取包。如果你没有安装 ngrep,

使用下面的命令安装:

```
apt-get install ngrep
```

2: 使用下面命令抓取包:

```
ngrep -p -q -W byline port 5060 > test.txt
```

3: 配置 UAC (软电话, IP 电话, 或 ATA)。

使用下面的描述来配置第一个 UAC:

```
sip proxy 10.1.x.y – IP of your proxy
```

```
user: 1000
```

```
password: 1000
```

使用下面的描述来配置第二个 UAC:

```
sip proxy 10.1.x.y – IP of your proxy
```

user: 1001

password: 1001

在完成配置后，你需要注册 IP 话机。不是所有的设备都会做自动注册的操作。

1. 使用下面的命令检查电话是否注册上：

```
openserctl u1 show
```

2. 用第一个 UAC 拨打 1001。第二个 UAC 会振铃。
3. 验证抓取的包没有相对于 INVITE 请求的“407- Proxy authentication required”错误和相对于 REGISTER 请求的“401- Unauthorized”错误。这证明了不需要验证。
4. 你可以使用如下命令查看抓包：

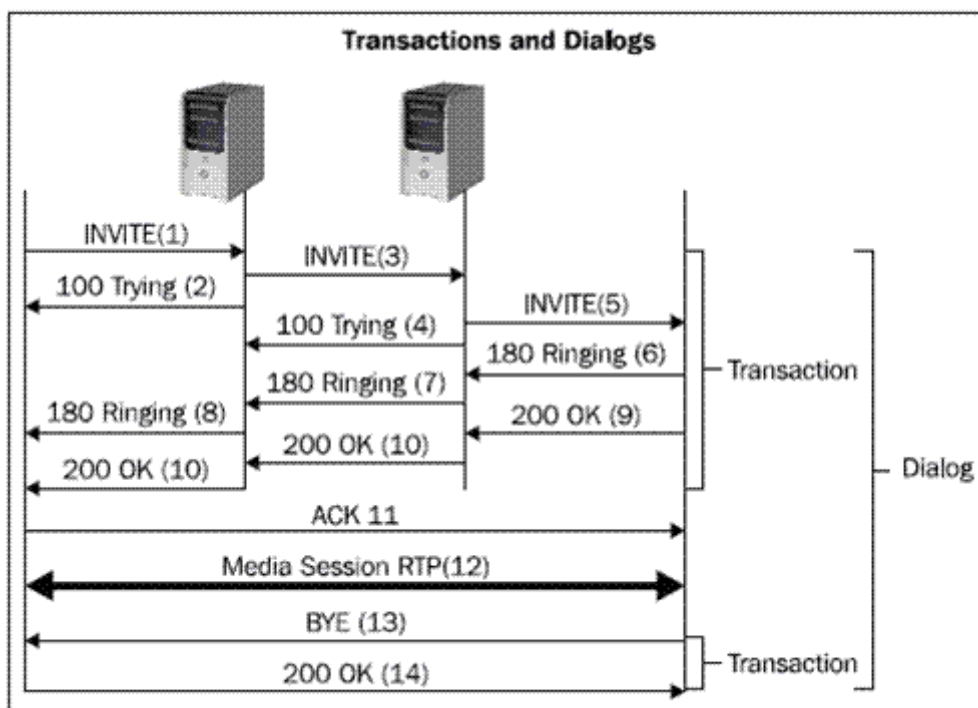
```
more test.txt
```

路由基础（ **Routing Basics** ）

探明如何路由 SIP 包并不是一件容易的事。我们将在这一节中说明通过代理服务器路由 SIP 包的一些基本概念。第一个重要的基本概念包含事务和对话。

事务和对话（ **Transactions and Dialogs** ）

一个事务通常由一个请求开始，由一个响应码（ a response code ）结束 。 VIA 头域中的 branch 参数用来标识一个事务。对话可以是开始于一个 INVITE 事务，结束于一个 BYE 事务。一个对话由 FROM，TO 和 CALL-ID 头域的结合所标识。并不是所有的 SIP 方法都可以启动一个对话，REGISTER 和 MESSAGE 方法就不行。



初始和接续请求（ **Initial and Sequential Requests** ）

了解初始和接续请求消息之间的区别是很重要的。对于初始请求，你必须决定如何使用发现机制来进行路由，通常它是基于 DNS 或是位置表（a location table）。

初始请求使用 VIA 头域来记录路由信息，如果你打开 record-routing，那么也可以使用 ROUTE 头域来进行记录。在一个事务中，SIP 包将被路由回到之前经过的返回到的每一个代理的 VIA 头域记录的地址。随后的请求则使用 CONTACT 头域中的地址进行路由。然而，如果你打开 record-routing，接下来的请求则会使用被发现的路由集合路由回去。

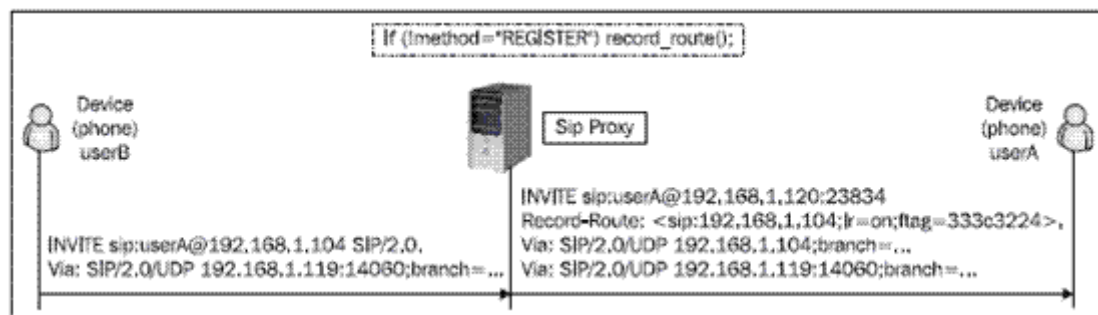
你可以用 TO 头域中的 TAG 参数来分辨初始和接续请求。

事务的上下文中的路由过程（ **Routing in a Context of a Transaction** ）

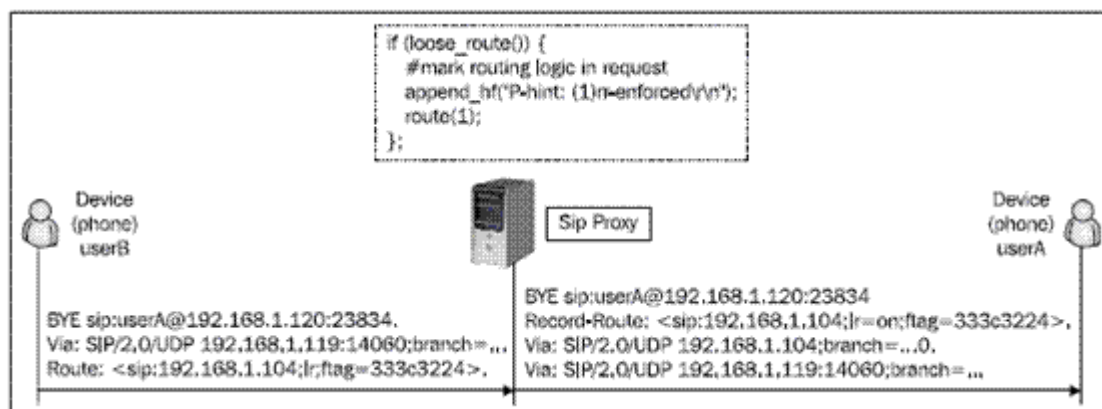
在一个事务中所有的请求都使用 VIA 头域来进行路由。所以，在到达最终的目的地之前，所有的响应都会经过代理。如果你使用函数 `t_relay()` 来路由请求，那么 SIP 代理会处在有状态模式下，则你可以使用区段 `onreply_route[]` 和 `failure_route[]` 来处理响应和失败。

对话的上下文中的路由过程（**Routing in the Context of a Dialog**）

同一个对话中的接续请求将直接使用 CONTACT 头域来进行点到点（peer-to-peer）的路由。不过大部分时间你会希望一些接续事务，例如 BYE，可以通过代理，这样就可以进行计费 and 对话控制。你可以打开 record-routing 来完成这项工作。这样做将指示脚本来进行 record-routes。

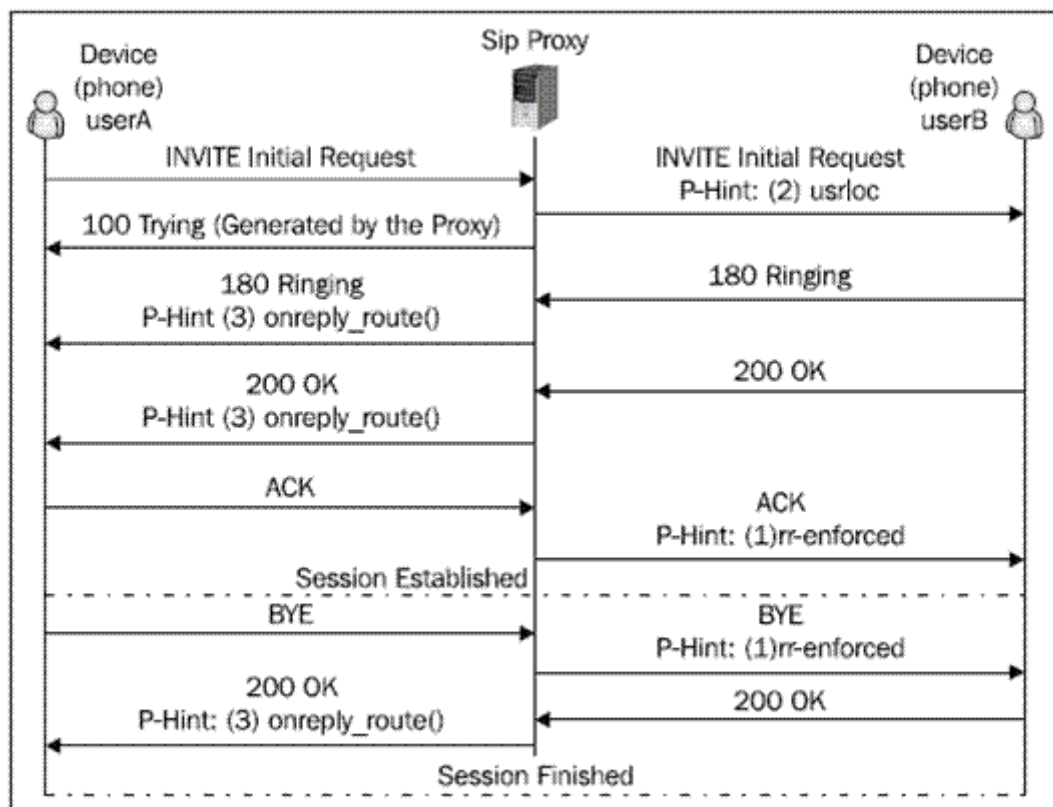


之后，你可以使用之前记录过的路由信息，也就是我们知道的路由集合（route set），来进行接续请求的前转。这是最通常的配置，也是默认的配置文件中的配置。



实验——跟踪一个完整的对话（**Lab-----Tracking a Complete Dialog**）

在这个实验中，我们将使用一个简化了的脚本来理解路由的概念。我们用函数 `append_hf()` 来添加一个头域到包中以标记包被脚本处理的位置信息。



步骤 1：使用下面的脚本（ openser.chapter4-2 ）。重启 OpenSER 并且重新注册电话。

```

route{

    # All messages, except for REGISTER will pass here

    if (!method=="REGISTER") record_route();

    # subsequent messages withing a dialog should take the

    # path determined by record-routing

    if (loose_route()) {

        # mark routing logic in request

        append_hf("P-hint: (1)rr-enforced\r\n");

        route(1);

    };

    # We will route only intra-domain requests
  
```

```

if (!uri==myself) {

    exit();

};

# main routing of intra-domain requests

if (uri==myself) {

    if (method=="REGISTER") {

        save("location");

        exit;

    };

    # native SIP destinations are handled using our USRLOC DB

    if (!lookup("location")) {

        sl_send_reply("404", "Not Found");

        exit;

    };

    append_hf("P-hint: (2)usrloc applied\r\n");

};

route(1);

}

route[1] {

    # send it out now; use stateful forwarding

    t_on_reply("1");

    t_on_failure("1");

    if (!t_relay()) {

```

```

        sl_reply_error();

    };

    exit;

}

onreply_route[1] {

    append_hf("P-hint: (3)passed thru onreply_route[1]\r\n");

}

failure_route[1] {

    append_hf("P-hint: (4)passed thru failure_route[1]\r\n");

}

```

步骤 2：使用 ngrep 抓取请求和响应包，并存入文件

```
ngrep -p -q -W byline port 5060 >rr-stateful
```

步骤 3：由 1000 到 1001 发起一通通话（任何注册的话机都可以）

步骤 4：用 CTRL-C 来终止 ngrep 的运行

使用文本编辑器来检查包并观察 P-Hint 头域。他们应该要和实验开始时的那幅图中的一样才是。

实验——无状态运行（**Lab-----Running Stateless**）

如果你使用函数 forward()来替代函数 t_relay()，那么 SIP 代理将运行在无状态的模式下。所有的工作还在进行，但是你将不能处理响应消息。代理在这时候将无法将同一个事务中请求和响应关联上。响应仍将使用 VIA 头域来处理。

步骤 1：使用 forward()函数替代 t_relay()函数。

Replace：

```

if (!t_relay()) {

    sl_reply_error();
}

```



```
};
```

By:

```
forward()
```

步骤 2: 重启 OpenSER, 重新注册话机

步骤 3: 使用 ngrep 抓取请求和响应到一个文件中

```
ngrep -p -q -W byline port 5060 > rr-stateless
```

步骤 4: 1000 打给 1001 一通电话

步骤 5: 结束通话后, 终止 ngrep

步骤 6: 使用文本编辑器查看 rr-stateless 文件。你将注意到响应消息中没有 P-Hint 头域。这表明他们没有被区段 onreply_route 处理。所以, 如果你使用无状态模式, 除了将消息前转至目的地外, 你将无法对回复的消息进行任何处理。

实验——关闭 **record-route** (**Lab-----Disabling record-route**)

在这个实验中, 我们将停止记录路由。对话中的接续请求将绕过 SIP 代理, 直接由一台话机传到另外一台上。当然, 使用了 CONTACT 头域的消息。

步骤 1: 将负责记录路由的那一行注释掉

```
#if (!method=="REGISTER") record_route();
```

步骤 2: 重启 OpenSER 并重新注册话机

步骤 3: 使用 ngrep 来抓取请求和响应消息存入文件

```
ngrep -p -q -W byline port 5060 > norr-stateless
```

步骤 4: 1000 打给 1001 一通电话

步骤 5: 结束通话后, 终止 ngrep

步骤 6：使用文本编辑器查看 `norrr-stateless` 文件。现在，你将注意到，无法看到 `BYE` 和 `ACK` 请求。因为他们此时直接由一端传到了另一端。如果你想对通话计费，那么你是不要 SIP 代理进行这个实验的行为的。

概要（ **Summary** ）

在这一章中，你已经学到了 `openser.cfg` 配置文件中每个区段的一些状态。这是最简单的一个配置文件。在下一章中，我们将增加脚本的功能和复杂度。这一章只是作为开发更高级脚本的一个起点。即使它比较简单，但是这样的脚本也已经可以让你能够连接两台话机，并能够互相通话了。

第五章：用 **MySQL** 添加认证（ **Adding Authentication with MySQL** ）

在这一章中我们将学到如何使用几种数据库后端来对 SIP 请求进行鉴定并提供诸如位置和别名表等数据的持续性。最主要的是，我们将使用 MySQL 来做每一件事。这一章分为两个部分。第一个部分，我们将学到如何实现认证，第二部分我们将学到如何处理不同方向的通话。

这一章的最后，你将能够：

- ☐ 配置 MySQL 来对 SIP 设备进行认证
- ☐ 使用 `openserctl` 工具来实现基本的操作，如添加和删除用户
- ☐ 改变脚本 `openser.cfg` 来配置 MySQL 认证
- ☐ 实现订阅列表的连续性
- ☐ 实现位置列表的连续性
- ☐ 重启服务器但是不丢失位置记录
- ☐ 正确的处理 inbound-to-inbound，inbound-to-outbound，outbound-to-inbound，和 outbound-to-outbound 会话
- ☐ 正确的处理 CANCEL 请求

我们在哪儿？（ **Where Are We？** ）

现在，我们仍然将集中精力于 SIP 代理上。然而，我们将包含一个新的部件，数据库。OpenSER 可以使用 MySQL 和 PostgreSQL。在这本书中，我们选择的是 MySQL。到目前为止它是 OpenSER 使用最多的数据库。

AUTH_DB 模块（ **The AUTH_DB Module** ）

以数据库为基础的认证是由 AUTH_DB 模块实现的。其他类型的认证，如 radius 和 diameter 可以分别使用 AUTH_RADIUS 和 AUTH_DIAMETER 来实现。AUTH_DB 和诸如 MySQL 和 PostgreSQL 等数据库模块一起工作。AUTH_DB 有一些参数没有在脚本中明确的声明。让我们看看 AUTH_DB 模块的默认参数。

Parameter	Default	Description
db_url	"mysql://openser:openser@localhost/openser"	URL of the database
user_column	"username"	Name of the column holding domains of users
domain_column	"domain"	Name of the column holding domains of users
password_column	"ha1"	Name of the column holding passwords
password_column2	"ha1b"	Name of the column holding pre-calculated HA1 strings that were calculated including the domain in the username.
calculate_ha1	0 (server assumes that ha1 strings are already calculated in the database)	Tell the server whether it should expect plaintext passwords in the database or not.
use_domain	0 (domains won't be checked when looking up in the subscriber database)	Use this parameter set to 1 if you have a multi-domain environment.
load_credentials	"rpid"	Specifies the credentials to be fetch from the database when the authentication is performed. The loaded credentials will be stored in AVPs.

由 ATUH_DB 模块会导出两个函数。

`www_authorize(realm, table)`

这个函数在 REGISTER 的认证中被使用，和 RFC2617 保持一致。

`proxy_authorize(realm, table)`

这个函数按照 RFC2617 对 non-REGISTER 请求的证书进行验证。如果验证成功，则该证书将被标记为已认证。

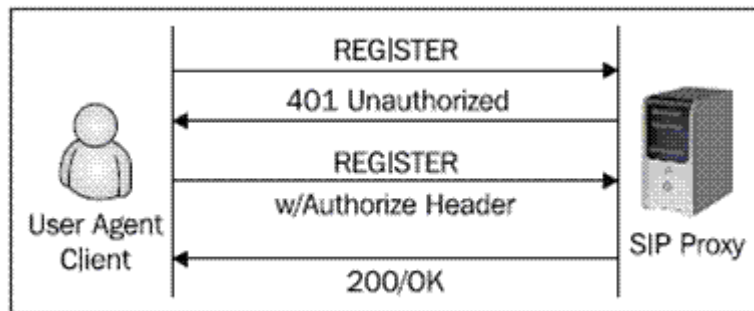
如果你的服务器是请求的终点，那么你必须使用 `www_authorize` 函数。而当请求的最终目的地不是你的服务器，那么则使用 `proxy_authorize` 函数，然后你要对请求进行前转，这时，服务器实际上是作为代理在工作。

`www_authorize` 和 `proxy_authorize` 的不同使用之处就在于请求的终点是否是你的服务器 (REGISTER)。

REGISTER 认证顺序 (The REGISTER Authentication Sequence)

脚本应该对 REGISTER 和 INVITE 消息进行认证。在修改 openser.cfg 脚本之前，让我们先展示这是如何发生的。当 OpenSER 收到 REGISTER 消息时，它先检查其是否有 Authorize 头域。如果没有找到，它会请求 UAC 的证书并退出。

当 UAC 收到请求后，向其再次发送 REGISTER 消息，此时的消息中带有 Authorize 头域。



注册过程 (ngrep 抓的包) (Register Sequence (Packets Captured by ngrep))

注册过程可以从下面的抓包中看到：

```
U 192.168.1.119:29040 -> 192.168.1.155:5060
```

```
REGISTER sip:192.168.1.155 SIP/2.0.
```

```
Via: SIP/2.0/UDP 192.168.1.119:29040;branch=z9hG4bK-d87543-13517a5a8218ff45-1--d87543-;rport.
```

```
Max-Forwards: 70.
```

```
Contact: <sip:1000@192.168.1.119:29040;rinstance=2286bddd834b3cfe>.
```

```
To: "1000" <sip:1000@192.168.1.155>.
```

```
From: "1000" <sip:1000@192.168.1.155>;tag=0d10cc75.
```

```
Call-ID:
```

```
e0739d571d287264NjhiZjM2N2UyMjhmNDViYTgzY2I4ODMxYTVIZITY0NDc..
```

```
CSeq: 1 REGISTER.
```

```
WWW-Authenticate: Digest realm="192.168.1.155",
```

```
nonce="46263864b3abb96a423a7ccf052fa68d4ad5192f".
```

```
Server: OpenSER (1.2.0-notls (i386/linux)).
```

```
Content-Length: 0.
```

U 192.168.1.119:29040 -> 192.168.1.155:5060
REGISTER sip:192.168.1.155 SIP/2.0.
Via: SIP/2.0/UDP 192.168.1.119:29040;branch=z9hG4bK-d87543-da776d09bd6fcb65-1--d87543-;rport.
Max-Forwards: 70.
Contact: <sip:1000@192.168.1.119:29040;rinstance=2286bddd834b3cfe>.
To: "1000"<sip:1000@192.168.1.155>.
From: "1000"<sip:1000@192.168.1.155>;tag=0d10cc75.
Call-ID: e0739d571d287264NjhiZjM2N2UyMjhmNDViYTgzY2I4ODMxYTVIZTY0NDc..
CSeq: 2 REGISTER.

Expires: 3600.

Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE, SUBSCRIBE, INFO.

User-Agent: X-Lite release 1003l stamp 30942.

Content-Length: 0.

U 192.168.1.155:5060 -> 192.168.1.119:29040

SIP/2.0 401 Unauthorized.

Via: SIP/2.0/UDP 192.168.1.119:29040;branch=z9hG4bK-d87543-13517a5a8218ff45-1--d87543-;rport=29040.

To:

"1000"<sip:1000@192.168.1.155>;tag=329cfeaa6ded039da25ff8cbb8668bd2.41bb.

From: "1000"<sip:1000@192.168.1.155>;tag=0d10cc75.

Call-ID: e0739d571d287264NjhiZjM2N2UyMjhmNDViYTgzY2I4ODMxYTVIZTY0NDc..
CSeq: 1 REGISTER.

WWW-Authenticate: Digest

realm="192.168.1.155",nonce="46263864b3abb96a423a7ccf052fa68d4ad5192f".

Server: OpenSER (1.2.0-notls (i386/linux)).

Content-Length: 0.

U 192.168.1.119:29040 -> 192.168.1.155:5060

REGISTER sip:192.168.1.155 SIP/2.0.

Via: SIP/2.0/UDP 192.168.1.119:29040;branch=z9hG4bK-d87543-da776d09bd6fcb65-1--d87543-;rport.

Max-Forwards: 70.

Contact: <sip:1000@192.168.1.119:29040;rinstance=2286bddd834b3cfe>.

To: "1000"<sip:1000@192.168.1.155>.

From: "1000"<sip:1000@192.168.1.155>;tag=0d10cc75.

Call-ID: e0739d571d287264NjhiZjM2N2UyMjhmNDViYTgzY2I4ODMxYTVIZTY0NDc..
CSeq: 2 REGISTER.

Expires: 3600.

Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE, SUBSCRIBE, INFO.

User-Agent: X-Lite release 1003l stamp 30942.

Authorization: Digest

username="1000",realm="192.168.1.155",nonce="46263864b3abb96a423a7ccf052fa68d4ad5192f",uri="sip:192.168.1.155",response="d7b33793a123a69ec12c8fc87abd4c03",algorithm=MD5.

Content-Length: 0.

U 192.168.1.155:5060 -> 192.168.1.119:29040

SIP/2.0 200 OK.

Via: SIP/2.0/UDP 192.168.1.119:29040;branch=z9hG4bK-d87543-da776d09bd6fcb65-1--d87543-;rport=29040.

To:

"1000"<sip:1000@192.168.1.155>;tag=329cfeaa6ded039da25ff8cbb8668bd2.c577.

From: "1000"<sip:1000@192.168.1.155>;tag=0d10cc75.

Call-ID: e0739d571d287264NjhiZjM2N2UyMjhmNDViYTgzY2I4ODMxYTVIZTY0NDc..

CSeq: 2 REGISTER.

Contact:

<sip:1000@192.168.1.119:29040;rinstance=2286bddd834b3cfe>;expires=3600.

Server: OpenSER (1.2.0-notls (i386/linux)).

Content-Length: 0.

注册过程的代码片段 (**Register Sequence Code Snippet**)

现在让我们看看这个过程在 openser.cfg 脚本中是如何写的吧：

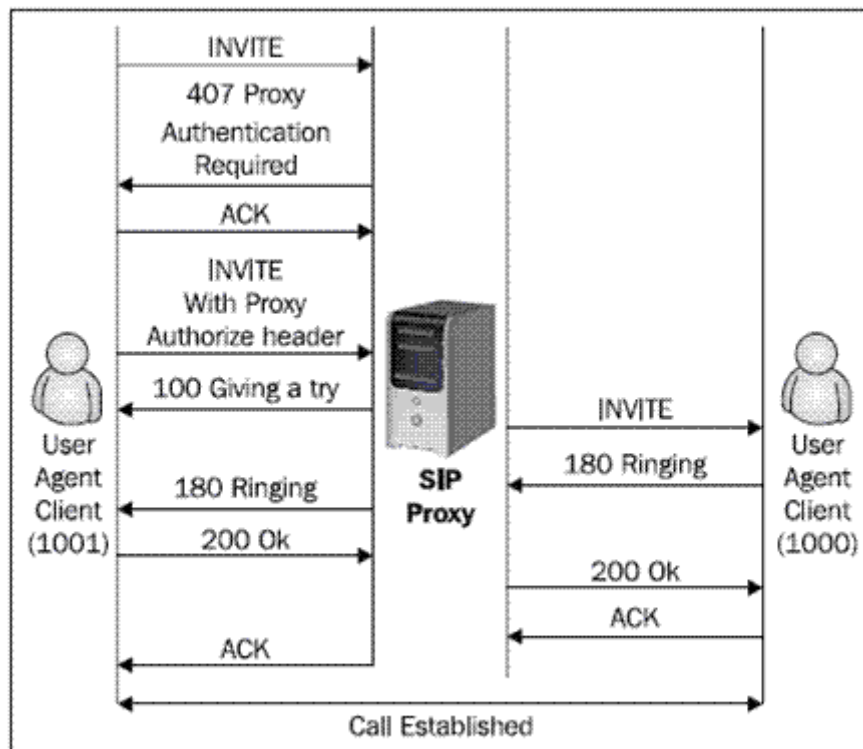
```
if (method=="REGISTER") {  
    # Uncomment this if you want to use digest authentication  
    if (!www_authorize("", "subscriber")) {  
        www_challenge("", "0");  
        exit;  
    };  
    save("location");  
  
    exit;  
}
```

在上面的过程中，第一次传的 REGISTER 包没有被 www_authorize 函数认证。然后，

www_challenge 语句被调用。它发送了“401 Unauthorized”包，这个包按照摘要认证方法（digest authentication scheme）包含了认证请求。UAC 第二次传的 REGISTER 包添加了 Authorize 头域，然后，save（“location”）函数被调用用来保存 AOR 到 MySQL 的位置表。

INVITE 认证过程（The INVITE Authentication Sequence）

相对的则是一通普通通话的 INVITE 认证过程。代理服务器总是会对第一个 INVITE 请求进行响应，使用的是“407 Proxy Authentication Required”消息。这个消息包含了 Authorize 头域，含有关于摘要认证的信息，如 realm 和 nonce。一旦 UAC 收到这个消息，它就会立即发送一条新的 INVITE 消息。现在，Authorize 中包含了可以使用 MD5 算法进行计算的摘要，包括 username，password，realm，和 nonce。如果在服务器上使用与之相同的参数进行计算的摘要和 UAC 传递的摘要相符时，用户得到认证。



INVITE 过程的抓包 (INVITE Sequence Packet Capture)

我们用 ngrep 抓了一套 INVITE 认证过程的包。这个过程能够帮助你理解上面的图。为了避免列的过长，我们没有将 SDP 头的内容一起附上。

U 192.168.1.169:5060 -> 192.168.1.155:5060

INVITE sip:1000@192.168.1.155 SIP/2.0.

Via: SIP/2.0/UDP 192.168.1.169;branch=z9hG4bKf45d977e65cf40e0.

From: <sip:1001@192.168.1.155>;tag=a83bebd75be1d88e.

To: <sip:1000@192.168.1.155>.

Contact: <sip:1001@192.168.1.169>.

Supported: replaces.

Call-ID: 8acb7ed7fc07c369@192.168.1.169.

CSeq: 39392 INVITE.

User-Agent: TMS320V5000 TI50002.0.8.3.

Max-Forwards: 70.

Allow: INVITE,ACK,CANCEL,BYE,NOTIFY,REFER,OPTIONS,INFO,SUBSCRIBE.

Content-Type: application/sdp.

Content-Length: 386.

(sdp header striped off).

U 192.168.1.155:5060 -> 192.168.1.169:5060

SIP/2.0 407 Proxy Authentication Required.

Via: SIP/2.0/UDP 192.168.1.169;branch=z9hG4bKf45d977e65cf40e0.

From: <sip:1001@192.168.1.155>;tag=a83bebd75be1d88e.

To: <sip:1000@192.168.1.155>;tag=329cfeaa6ded039da25ff8cbb8668bd2.b550.

Call-ID: 8acb7ed7fc07c369@192.168.1.169.

CSeq: 39392 INVITE.

Proxy-Authenticate: Digest realm="192.168.1.155",
nonce="4626420b4b162ef84a1a1d3966704d380194bb78".

Server: OpenSER (1.2.0-notls (i386/linux)).

Content-Length: 0.

U 192.168.1.169:5060 -> 192.168.1.155:5060

ACK sip:1000@192.168.1.155 SIP/2.0.

Via: SIP/2.0/UDP 192.168.1.169;branch=z9hG4bKf45d977e65cf40e0.

From: <sip:1001@192.168.1.155>;tag=a83bebd75be1d88e.

To: <sip:1000@192.168.1.155>;tag=329cfeaa6ded039da25ff8cbb8668bd2.b550.

Contact: <sip:1001@192.168.1.169>.

Call-ID: 8acb7ed7fc07c369@192.168.1.169.

CSeq: 39392 ACK.

User-Agent: TMS320V5000 TI50002.0.8.3.

Max-Forwards: 70.

Allow: INVITE,ACK,CANCEL,BYE,NOTIFY,REFER,OPTIONS,INFO,SUBSCRIBE.

Content-Length: 0.

U 192.168.1.169:5060 -> 192.168.1.155:5060

INVITE sip:1000@192.168.1.155 SIP/2.0.

Via: SIP/2.0/UDP 192.168.1.169;branch=z9hG4bKcdb4add5db72d493.

From: <sip:1001@192.168.1.155>;tag=a83bebd75be1d88e.

To: <sip:1000@192.168.1.155>.

Contact: <sip:1001@192.168.1.169>.

Supported: replaces.

**Proxy-Authorization: Digest username="1001", realm="192.168.1.155",
algorithm=MD5, uri="sip:1000@192.168.1.155",
nonce="4626420b4b162ef84a1a1d3966704d380194bb78",
response="06736c6d7631858bb1cbb0c86fb939d9".**

Call-ID: 8acb7ed7fc07c369@192.168.1.169.

CSeq: 39393 INVITE.

User-Agent: TMS320V5000 TI50002.0.8.3.

Max-Forwards: 70.

Allow: INVITE,ACK,CANCEL,BYE,NOTIFY,REFER,OPTIONS,INFO,SUBSCRIBE.

Content-Type: application/sdp.

Content-Length: 386.

(sdp header striped off)

INVITE Code Snippet

In the code below, the SIP proxy will challenge the user for

credentials on any request different from REGISTER. We consume the credentials after authentication, for security reasons, to avoid sending encrypted material ahead.

```
if (!proxy_authorize("", "subscriber")) {
```

```
    proxy_challenge("", "0");
```

```
    exit;
```

```
};
```

```
consume_credentials();
```

```
lookup("aliases");
```

```

if (!uri==myself) {

append_hf("P-hint: outbound alias\r\n");

route(1);

};

# native SIP destinations are handled using our USRLOC DB

if (!lookup("location")) {

sl_send_reply("404", "Not Found");

exit;

};

append_hf("P-hint: usrloc applied\r\n");

};

route(1);

```

摘要认证 (**Digest Authentication**)

摘要认证的基础是 RFC2617 中的“HTTP Basic and Digest Access Authentication”。我们这一章节的目的是介绍一个带有摘要认证的系统的要素。它不是对于 SIP 的所有可能的安全性问题的回答，但确实是一种可以保护在网络传输中的用户名和密码的好方法。



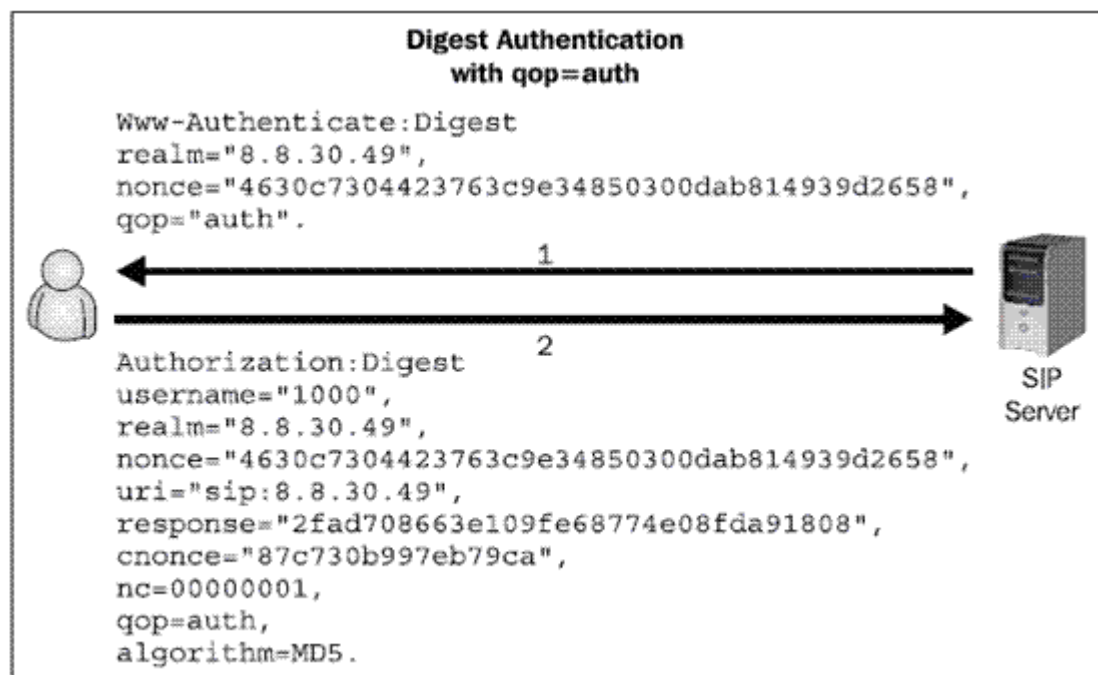
摘要方法是一种简单的请求响应机制（challenge-response mechanism）。使用 nonce 值向 UA 发出请求。一个有效的响应包含所有参数的一个校验和（checksum）。因此，密码从不以简单的文本形式进行传输。

WWW-Authenticate 响应头（ WWW-Authenticate Response Header ）

如果服务器没有收到带有有效 Authorize 头域的 REGISTER 或 INVITE 请求，那么它会返回一个带有 WWW-Authenticate 头域的“401 unauthorized”消息。这个头包含一个 realm 和一个 nonce。

认证请求头（ The Authorization Request Header ）

我们希望用户能够再试一次，但是要带上 Authorize 头域。该头要包含用户名，realm 和 nonce（由 server 提供），uri，一个 32 位的十六进制数，还有一种认证方法（此例中是 MD5）。这种响应是用户使用一种特定的算法产生的校验和。



QOP——保护质量 (QOP ----- Quality of Protection)

qop 参数表明了用户应用到该消息上的保护的质量。如果出现了这个参数，那么它的值要是服务器支持的所有可选值之一。这些选择在 WWW-Authenticate 头域中表明。这些值会影响摘要的计算。之所以有这条指令，是为了和 RFC2809 的最小实现保持兼容。

你可以在两个函数中对该参数进行配置，分别是 `www_challenge(realm, qop)` 和 `proxy_challenge(realm, qop)`。如果被配成 1，那么服务器就会要求有 qop 参数。总是使用 qop=1 会帮助你避免“replay”攻击。然而，一些用户对于 qop 会不兼容。对于摘要认证的详细描述见 RFC2617。

安装 MySQL 支持

为了允许持续性，换句话说，为了能够将用户的证书保存在数据库中，也就是说，为了避免这些数据在断电和重启的情况下被毁，OpenSER 需要配置使用如 MySQL 的数据库。在你继续之前，确认你已经安装了 MySQL 并且已经编译并安装了 OpenSER MySQL 模块是非常重要的。

在第三章，我们在编译 OpenSER 的时候加入了对 MySQL 的支持。检查 `/lib/openser/modules` 文件夹看看有没有 `mysql.so` 模块。

在你可以使用带有 MySQL 的 OpenSER 之前，还有一些任务需要去做。

步骤 1：确认 `mysql.so` 模块在指定的文件夹中是否存在：

```
ls /lib/openser/modules/mysql.so
```

如果该模块不存在，那么请重新编译 OpenSER 以支持 MySQL。

步骤 2：使用 `openser_mysql.sh` shell 脚本创建 MySQL 表。

这个脚本将使用下面的参数来创建 MySQL 表：

```
DBNAME="openser"
DBHOST="localhost"
DBRWUSER="openser"
DBRWPW="openserrw"
DBROUSER="openserro"
DBROPW="openserro"
DBROOTUSER="root"
cd/sbin
./openser_mysql.sh create
```

MySQL password for root:
Enter password:
Enter password:
creating database opener ...
Core OpenSER tables succesfully created.
Install presence related tables ?(y/n):y
creating presence tables into opener ...

Presence tables succesfully created.

Install extra tables - imc,cpl,siptrace,domainpolicy ?(y/n):y
creating extra tables into opener ...
Extra tables succesfully created.
Install SERWEB related tables ?(y/n):n

Domain (realm) for the default user 'admin': voffice.com.br

需要密码才能够访问该数据库。这时候，密码是空的。该脚本会询问密码两次；两次都要按回车键。此脚本也会询问域（realm）；要告诉管理者用户的域。

步骤 3：配置 OpenSER 使用 MySQL。

按照下面文件的高亮部分进行修改：

```
# ----- module loading -----  
#set module path  
mpath="//lib/openser/modules/"  
# Uncomment this if you want to use SQL database  
loadmodule "mysql.so"  
loadmodule "sl.so"  
loadmodule "tm.so"  
loadmodule "rr.so"  
loadmodule "maxfwd.so"  
loadmodule "usrloc.so"  
loadmodule "registrar.so"  
loadmodule "textops.so"  
loadmodule "mi_fifo.so"  
# Uncomment this if you want digest authentication  
# mysql.so must be loaded !  
loadmodule "auth.so"  
loadmodule "auth_db.so"  
# ----- setting module-specific parameters -----
```

```

# -- mi_fifo params --
modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
# -- usrloc params --
#modparam("usrloc", "db_mode", 0)
# Uncomment this if you want to use SQL database
# for persistent storage and comment the previous line
modparam("usrloc", "db_mode", 2)
# -- auth params --
# Uncomment if you are using auth module

#

modparam("auth_db", "calculate_ha1", yes)
#
# If you set "calculate_ha1" parameter to yes (which true in this config),
# uncomment also the following parameter)
#
modparam("auth_db", "password_column", "password")
# -- rr params --
# add value to ;lr param to make some broken UAs happy
modparam("rr", "enable_full_lr", 1)
# ----- request routing logic -----
# main routing logic
route{
# initial sanity checks -- messages with
# max_forwards==0, or excessively long requests
if (!mf_process_maxfwd_header("10")) {
sl_send_reply("483", "Too Many Hops");
exit;
};
if (msg:len >= 2048 ) {
sl_send_reply("513", "Message too big");
exit;
};
# we record-route all messages -- to make sure that
# subsequent messages will go through our proxy; that's
# particularly good if upstream and downstream entities
# use different transport protocol
if (!method=="REGISTER")
record_route();
# subsequent messages withing a dialog should take the

```



```

# path determined by record-routing
if (loose_route()) {
# mark routing logic in request
append_hf("P-hint: rr-enforced\r\n");
route(1);
};
if (!uri==myself) {
# mark routing logic in request
append_hf("P-hint: outbound\r\n");
# if you have some interdomain connections via TLS
#if(uri=~"@tls_domain1.net") {
# t_relay("tls:domain1.net"); # exit;
#} else if(uri=~"@tls_domain2.net") {
# t_relay("tls:domain2.net");
# exit;
#}
route(1);
}; # if the request is for other domain use UsrLoc
# (in case, it does not work, use the following command
# with proper names and addresses in it)
if (uri==myself) {
if (method=="REGISTER") {
# Uncomment this if you want to use digest.
if (!www_authorize("", "subscriber")) {
www_challenge("", "0");
exit;
};
save("location");
exit;
};
if (!proxy_authorize("", "subscriber")) {
proxy_challenge("", "0");
exit;
};
consume_credentials();
lookup("aliases");
if (!uri==myself) {
append_hf("P-hint: outbound alias\r\n");
route(1);
};          # native SIP destinations are handled using our USRLOC DB
if (!lookup("location")) {

```

```

sl_send_reply("404", "Not Found");
exit;
};
append_hf("P-hint: usrloc applied\r\n");
};
route(1);
}
route[1] {

# send it out now; use stateful forwarding as it works

reliably
# even for UDP2TCP
if (!t_relay()) {
sl_reply_error();
};
exit;

}

```

Openser.cfg 文件分析 (openser.cfg File Analysis)

现在，配置已经做好准备来对 REGISTER 事务进行认证了。我们可以将 AOR 保存在位置数据库中来实现持续性。这就允许我们可以重启服务器而不会丢失 AOR 记录和影响 UACs。

另一个重要的方面是，OpenSER 现在要对 REGISTER 请求进行认证。之后我们要对 INVITE 请求的认证进行实现。现在需要的是 UACs 的注册认证。

```

loadmodule "mysql.so"
loadmodule "auth.so"

loadmodule "auth_db.so"

```

MySQL 支持可以通过在要加载的模块列表中添加包含 mysql.so 语句来轻松实现。MySQL 模块要在其他模块之前加载。有些模块，例如 uri_db，需要依靠 MySQL 来加载。

认证的能力是由 auth.so 和 auth_db.so 模块提供的。这些模块被用来使能认证功能。模块 uri_db 列出了一些认证函数。

```

modparam("auth_db", "calculate_ha1", 1)

```

```
modparam("usrloc", "db_mode", 2)
```

参数 `calculate_hal` 告诉 `auth_db` 模块要使用明文密码。我们将使用明文密码来与 `SerMyAdmin` 兼容。

`db_mode` 参数告诉 `usrloc` 模块存储或获取数据库中的 AOR 记录。

```
if (method=="REGISTER") {
    # Uncomment this if you want to use digest auth.
    if (!www_authorize("", "subscriber")) {
        www_challenge("", "0");
        exit;
    };

    save("location");

    exit;
} else if (method=="INVITE") {
    if (!proxy_authorize("", "subscriber")) {
        proxy_challenge("", "0");
        exit;
    };
    consume_credentials();
};
```

在上面的代码片段中，我们检查了 `INVITE` 和 `REGISTER` 方法的认证过程。

如果是 `REGISTER` 方法，并且证书是正确的，那么 `www_authorize` 返回 `true`。在认证完成后，系统保存该 UAC 的位置信息数据。第一个参数指定了用户被进行认证的域（realm）。Realm 通常是域名（domain name）或主机名（host name）。第二个参数告诉 `OpenSER` 到底要寻找那一个 MySQL 表。

```
www_challenge("", "0");
```

如果包中不含有 `Authorize` 头域，我们则向 UAC 发送“401 unauthorized”消息。它告诉 UAC 重新传送包含证书的请求消息。`www_challenge` 命令有两个参数。第一个是 UAC 用来计算摘要的域（realm）。第二个参数则会影响发送给 UAC 消息中的 `qop` 参数的使用。1 表明在摘要中要包含 `qop`。有些话机可能不支持 `qop`。你可以在这种状况下试试将参数设为 0。

```
consume_credentials();
```

我们不想冒险将摘要证书发送到服务器上。因此，我们使用函数 `consume_credentials()` 函数在将请求中继前移除 `Authorize` 头域。

```
if (!proxy_authorize("", "subscriber")) {
```

我们使用 `proxy_authorize()` 函数来对认证头进行检查。如果我们不检查证书我们会被认为是一个比较开放的中继器。它的参数与 `www_authorize` 的参数相同

Openserctl shell 脚本

`Openserctl` 工具是安装在 `/usr/sbin` 上的 shell 脚本。被用来使用命令行的方式来对 `OpenSER` 进行管理。可以用来进行：

- ☐ 启动，终止，重启 `OpenSER`
- ☐ 展示，授权，撤销 `ACLs`
- ☐ 添加，删除，列出别名
- ☐ 添加，删除，配置 `AVP`
- ☐ 管理 `LCR` (low cost routes)
- ☐ 管理 `RPID`
- ☐ 添加，删除，列出订阅者
- ☐ 添加，删除，展示 `usrloc` 表“in-ram”
- ☐ 监控 `OpenSER`

我们将在下面的章节中学习一些它的选项.下面是 `openserctl help` 命令的输出信息：

```
/etc/openser# openserctl help
database engine 'MYSQL' loaded
Control engine 'FIFO' loaded
/usr/sbin/openserctl 1.2 - $Revision: 1.3 $
Existing commands:
-- command 'start|stop|restart'
restart ..... restart OpenSER
start ..... start OpenSER
stop ..... stop OpenSER
```

```

-- command 'acl' - manage access control lists (acl)
acl show [<username>] ..... show user membership
acl grant <username> <group> ..... grant user membership (*)
acl revoke <username> [<group>] .... grant user membership(s) (*)

-- command 'alias_db' - manage database aliases
alias_db show <alias> ..... show alias details
alias_db list <sip-id> ..... list aliases for uri
alias_db add <alias> <sip-id> ..... add an alias (*)
alias_db rm <alias> ..... remove an alias (*)
alias_db help ..... help message
- <alias> must be an AoR (username@domain)"
- <sip-id> must be an AoR (username@domain)"

-- command 'avp' - manage AVPs
avp list [-T table] [-u <sip-id|uuid>]
[-a attribute] [-v value] [-t type] ... list AVPs
avp add [-T table] <sip-id|uuid>

<attribute> <type> <value> ..... add AVP (*)

avp rm [-T table] [-u <sip-id|uuid>]
[-a attribute] [-v value] [-t type] ... remove AVP (*)
avp help ..... help message
- -T - table name
- -u - SIP id or unique id
- -a - AVP name
- -v - AVP value
- -t - AVP name and type (0 (str:str), 1 (str:int),
2 (int:str), 3 (int:int))
- <sip-id> must be an AoR (username@domain)
- <uuid> must be a string but not AoR

-- command 'db' - database operations
db exec <query> ..... execute SQL query
db show <table> ..... display table content

-- command 'lcr' - manage least cost routes (lcr)
* lcr *

* IP addresses must be entered in dotted quad format e.g. 1.2.3.4 *
* <uri_scheme> and <transport> must be entered in integer or text,*
* e.g. transport '2' is identical to transport 'tcp'. *
* scheme: 1=sip, 2=sips; transport: 1=udp, 2=tcp, 3=tls *
* Examples: lcr addgw_grp usa 1 *
* lcr addgw level3 1.2.3.4 5080 sip tcp 1 *

```

```

* lcr addroute +1 % 1 1 *
lcr show .....
..... show routes, gateways and groups
lcr reload .....
..... reload lcr gateways
lcr addgw_grp <grp_name> .....
..... add gateway group, autocreate grp_id
lcr addgw_grp <grp_name> <grp_id> .....
..... add gateway group with grp_id
lcr rmgw_grp <grp_id> .....
..... delete the gw_grp
lcr addgw <gw_name> <ip> <port> <scheme> <transport> <grp_id> .....
..... add a gateway
lcr addgw <gw_name> <ip> <port> <scheme> <transport> <grp_id> <prefix> .....
..... add a gateway with prefix
lcr addgw <gw_name> <ip> <port> <scheme> <transport> <grp_id> <prefix>
<strip>
..... add a gateway with prefix and strip
lcr rmgw <gw_name> .....
..... delete a gateway
lcr addroute <prefix> <from> <grp_id> <prio> .....
..... add a route

lcr rmroute <prefix> <from> <grp_id> <prio> .....
..... delete a route
-- command 'rpidd' - manage Remote-Party-ID (RPID)
rpidd add <username> <rpidd> ..... add rpidd for a user (*)
rpidd rm <username> ..... set rpidd to NULL for a user (*)
rpidd show <username> ..... show rpidd of a user
-- command 'speeddial' - manage speed dials (short numbers)
speeddial show <speeddial-id> ..... show speeddial details
speeddial list <sip-id> ..... list speeddial for uri
speeddial add <sip-id> <sd-id> <new-uri> [<desc>] ...
..... add a speedial (*)
speeddial rm <sip-id> <sd-id> ..... remove a speeddial (*)
speeddial help ..... help message
- <speeddial-id>, <sd-id> must be an AoR (username@domain)
- <sip-id> must be an AoR (username@domain)
- <new-uri> must be a SIP AoR (sip:username@domain)
- <desc> a description for speeddial

```

```

-- command 'add|mail|passwd|rm' - manage subscribers
add <username> <password> <email> .. add a new subscriber (*)
passwd <username> <passwd> ..... change user's password (*)
rm <username> ..... delete a user (*)
mail <username> ..... send an email to a user
-- command 'cisco_restart' - restart CISCO phone (NOTIFY)
cisco_restart <uri> ..... restart phone configured for <uri>
-- command 'online' - dump online users from memory
online ..... display online users
-- command 'monitor' - show internal status
monitor ..... show server's internal status
-- command 'ping' - ping a SIP URI (OPTIONS)
ping <uri> ..... ping <uri> with SIP OPTIONS
-- command 'ul|alias' - manage user location or aliases
ul show [<username>]..... show in-RAM online users
ul rm <username> [<contact URI>].... delete user's UrLoc entries
ul add <username> <uri> ..... introduce a permanent UrLoc entry
ul add <username> <uri> <expires> .. introduce a temporary UrLoc entry
-- command 'fifo'

fifo ..... send raw FIFO command

```

Openserctl 资源文件 (Openserctl Resource File)

在版本 1.1 中，我们介绍一个叫做 openserctlrc 的资源文件。这个脚本可以在/etc/openser 中找到。openserctl 工具对其进行语法解析然后使用其配置数据库认证和一些进行沟通需要的参数。通常，它使用 FIFO 机制来向 OpenSER 守护进程发送命令。

| 安全起见，改变默认的访问数据库的用户名和密码是很重要的 |

Openserctlrc 文件 (Openserctlrc File)

To show the file, issue a command:

```

cat /etc/openser/openserctlrc
# $Id: openserctlrc,v 1.2 2006/07/05 19:37:20 miconda Exp $
#
# openser control tool resource file
#
# here you can set variables used in the openserctl
## your SIP domain

```

```
SIP_DOMAIN=voffice.com.br
## database type: MYSQL or PGSQL, by default none is loaded
DBENGINE=MYSQL
## database host
DBHOST=localhost
## database name
DBNAME=openser
## database read/write user
DBRWUSER=openser
## database read only user
DBROUSER=openserro
## password for database read only user
DBROPW=openserro
## database super user
DBROOTUSER="root"
## type of aliases used: DB - database aliases; UL - usrloc aliases
## - default: none
ALIASES_TYPE="DB"

## control engine: FIFO or UNIXSOCK

## - default FIFO
CTLENGINE="FIFO"
## path to FIFO file
OSER_FIFO="FIFO"
## check ACL names; default on (1); off (0)
VERIFY_ACL=1
## ACL names - if VERIFY_ACL is set, only the ACL names from below list
## are accepted
ACL_GROUPS="local ld int voicemail free-pstn"
## verbose - debug purposes - default '0'

VERBOSE=1

## do (1) or don't (0) store plaintext passwords

## in the subscriber table - default '1'

# STORE_PLAINTEXT_PW=0
```


使用带有认证功能的 **OpenSER** (**Using OpenSER with Authentication**)

现在，让我们以一种实践的方式来实现认证。

步骤 1：按照上面章节中描述的内容来修改 openser.cfg 文件。

步骤 2：使用命令/etc/init.d/openser restart 来重启 OpenSER。

步骤 3：使用被 openserctl 使用的默认的参数来配置 openserctrlrc。

```
# $Id: openserctrlrc 1827 2007-03-12 15:22:53Z bogdan_iancu $
#
# openser control tool resource file
#
# here you can set variables used in the openserctl
## your SIP domain
SIP_DOMAIN=voffice.com.br
## database type: MYSQL or PGSQL, by defaulte none is loaded
DBENGINE=MYSQL
## database host
DBHOST=localhost
## database name
DBNAME=openser
## database read/write user

DBRWUSER=openser

## database read only user
DBROUSER=openserro
## password for database read only user
DBROPW=openserro
## database super user
DBROOTUSER="root"
## type of aliases used: DB - database aliases; UL - usrloc aliases
## - default: none
ALIASES_TYPE="DB"
## control engine: FIFO or UNIXSOCK
## - default FIFO
CTLENGINE="FIFO"
## path to FIFO file
```

```

OSER_FIFO="/tmp/openser_fifo"
## check ACL names; default on (1); off (0)
VERIFY_ACL=1
## ACL names - if VERIFY_ACL is set, only the ACL names from below list
## are accepted
ACL_GROUPS="local ld int voicemail free-pstn"
## presence of serweb tables - default "no"
HAS_SERWEB="yes"
## verbose - debug purposes - default '0'
VERBOSE=1
## do (1) or don't (0) store plaintext passwords
## in the subscriber table - default '1'

STORE_PLAINTEXT_PW=1

```

步骤 4：使用 `openserctl` 工具来配置两个用户帐户。

```
/sbin/openserctl add 1000 password 1000@voffice.com.br
```

```
/sbin/openserctl add 1001 password 1001@voffice.com.br
```

| 如果你碰到“复制关键字 (Duplicate Keys) ”的问题，请检查你是否 |

| 安装了 SerWEB 表。如果你已经装了，那么只要将 HAS_SERWEB 设 |

| 为“yes”即可。 |

| 当你被要求密码时，使用 `openserrw` |

你可以使用 `openserctl` 的 `rm` 命令来删除用户，使用 `openserctl` 的 `passwd` 命令来修改密码。

步骤 5：使用 `ngrep` 工具来观察 SIP 消息：

```
#ngrep -p -q -W byline port 5060 >register.pkt
```

步骤 6：使用用户名和密码注册双方话机：

步骤 7：验证话机是否注册上，使用下面的命令：

```
#openserctl ul show
```

步骤 8：你可以使用下面的命令来验证哪些用户在线：

```
#openserctl online
```

步骤 9: 你可以使用下面的命令来 ping 一个用户:

```
#openserctl ping 1000
```

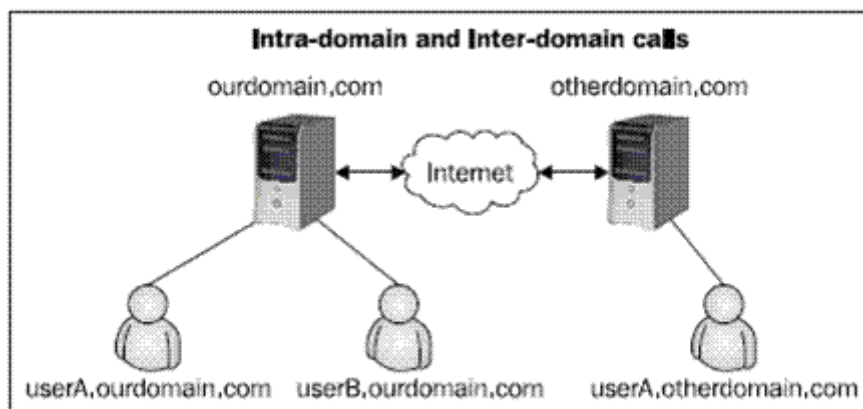
步骤 10: 使用 ngrep 工具来验证认证信息

步骤 11: 从一个用户向另外一个用户打通电话

步骤 12: 使用下面命令验证在 register.pkt 文件中的认证

```
#pg register.pkt
```

增强型脚本 (**Enhancing the Script**)



由 SIP 代理处理的通话可以被分为:

- ☐ 内域
- ☐ 带外内域
- ☐ 带内内域
- ☐ 带外到带外

让我们来描述一些目前我们脚本的一些问题:

问题#1: 目前, 我们没有检查从其他域过来的带外通话的标识。这就使得我们的服务器成为了比较开放的中继。因此, 任何人都可以使用我们的服务器了隐藏他们的标识。

问题#2: 我们的脚本不接受来自另一个域的来电

问题#3: 一个用户可以使用另一个用户的证书来伪造 INVITE 请求中的 FROM 头域。

问题#4：一个用户可以使用另一个用户的证书来伪造 REGISTER 请求中的 TO 头域。

问题#5：此脚本不会准备对多域（multiple domains）进行管理。

管理多域（**Managing Multiple Domains**）

到现在为止，我们已经使用 `uri==myself` 这一行指令来对请求进行了校验。然而，这条指令仅仅校验了本地名字和地址。如果我们需要管理多域，我们必须使用域模块和它的 `is_from_local()` 和 `is_uri_host_local()` 函数。

就像我之前说的那样，域模块导出了两个函数，这两个函数将在我们的脚本中使用。第一个是 `is_from_local()`，用来校验 FROM 头域是否包含我们代理管理着的域。第二个函数是 `is_uri_host_local()`，用来替代 `uri==myself` 指令。这些函数的优点是它们可以检查 MySQL 表中的域。然后你就可以在你的配置中处理多域了。

	这个函数需要所有的被管理的域都要被插入数据库中	
	对于使用这个功能资源的用户来说，一个相当普遍的	
	错误就是在对话机进行注册之前，忘记将域插入到 MySQL	
	数据库。	

替代路线（**Alternative Routes**）

为了简化我们的脚本，我们将创建几个替代路线。我们已经看到脚本可能会变得非常复杂和难懂。为了避免这种情况的发生，我们已经创建了一些和子函数类似的替代路线。使用替代路线，允许我们将代码分片以加强它的可读性。

注册请求（Register Requests）（`route[2]`）

注册请求路线负责处理所有的注册请求。这段代码对用户进行认证并保存 UAC 的位置信息。

```
route[2] {
#
# -- Register request handler --
#
if (is_uri_host_local()) {
if (!www_authorize("", "subscriber")) {
www_challenge("", "1");
exit;
};
if (!check_to()) {
sl_send_reply("401", "Unauthorized");
```

```

exit;
};
save("location");
exit;
} else if {
sl_send_reply("401", "Unauthorized");
};
}

```

非注册请求 (Non-Register Requests) (route[3])

非注册请求路线处理所有其他的请求。请求需要再次进行认证。我们已经决定把请求分成下面几个部分：

- ☐ 带内到带内 (route[10])
- ☐ 带内到带外 (route[11])
- ☐ 带外到带内 (route[12])
- ☐ 带外到带外 (route[13])

通常，你会允许带认证的带内到带内的请求。这时比较普通的情况。带内到带外和带外到带内被用来处理内域请求。大多数的 VoIP 服务提供商不允许内域之间的交互，因为这回减少其潜在收入。带外到带外更是不会被允许的。大多数情况下，这种交互被认为是一种安全漏洞。

```

route[3] {
#
# -- non-register requests handler --
#
# Verify the source (FROM)
if (is_from_local()){
# From an internal domain -> check the credentials and the FROM
if (!proxy_authorize("", "subscriber")) {
proxy_challenge("", "1");
exit;
} else if (!check_from()) {
sl_send_reply("403", "Forbidden, use From=ID");
exit;
};
consume_credentials();
# Verify aliases
lookup("aliases");
# Verify the destination (URI)
if (is_uri_host_local()) {
# -- Inbound to Inbound
route(10);
} else {
# -- Inbound to outbound
route(11);
};
}

```

```

} else {
#Verify aliases, if found replace R-URI.
lookup("aliases"); # Verify the destination (URI)
if (is_uri_host_local()) {
#-- Outbound to inbound
route(12);
} else {
# -- Outbound to outbound
route(13);
};
};
}

```

管理我们域中发起的通话 (Managing Calls Coming from Our Domain)

我们的脚本 opener.cfg 现在使用源地址和目的地址来区分通话。使用函数 if(is_uri_host_local())来决定目的地, 使用 if(is_from_local())来决定源地址。

对于带内发起的通话, 我们会首先检查标识并删除其证书以避免发送之。我们在检查目的地和前转请求前会对定义的别名进行解析。

如果通话目的地是我们管理域中之一 (使用 is_uri_host_local()检查), 我们将其送至 route(10)否则如果是一个外部域则将其送至 route(11)。

带内到带内——route[10] (Inbound-to-Inbound-----route[10])

带内目的地会被用户位置数据库处理。

```

route[10] {
#from an internal domain -> inbound
#Native SIP destinations are handled using the location table
append_hf("P-hint: inbound->inbound \r\n");
if (!lookup("location")) {
sl_send_reply("404", "Not Found");
exit;
};
route(1);
}

```

带内到带外——route[11] (Inbound-to-outbound-----route[11])

我们将使用 DNS 搜索来将通话路由到外部的目的地。

```

route[11] {
# from an internal domain -> outbound
# Simply route the call outbound using DNS search
append_hf("P-hint: inbound->outbound \r\n");
route(1);
}

```

带外到带内——route[12] (Outbound-to-Inbound-----route[12])

我们允许通话从外部域到我们自己的话机。这个配置会让许多垃圾信息传到我们的话机上, 但是实际情况却不是这样。我相信这样做的收益远大于风险。以后到底会怎样, 谁也不知道。对于我来说比较合理的是, 开放的接收通话就应该像开放的接收传统通话, 开放的接收 emails 一样。

```

route[12] {
# From an external domain -> inbound
# Verify aliases, if found replace R-URI.
lookup("aliases");
if (!lookup("location")) {
sl_send_reply("404", "Not Found");
exit;
};
route(1);
}

```

带外到带外——route[13] (Outbound-to-Outbound-----route[13])

我们不想做一个开放的中继器来对外部消息进行中继。如果下面的配置没有进行，那么其他人就能够使用我们的代理来匿名的路由通话。

```

route[13] {
#From an external domain outbound
#we are not accepting these calls
sl_send_reply("403", "Forbidden");
exit;
}

```

函数 **check_to()**和 **check_from()** (The Functions **check_to()** and **check_from()**)

当操作 SIP 代理时，你应该确认的是一个有效的帐户不会被一个没有进行认证的用户所使用。Check_to()和 check_from()函数被用来将 SIP 用户通认证用户做映射。SIP 用户在 FROM 和 TO 头域中，而 auth 用户仅仅用来进行认证 (Authorize 头域) 并有自己的密码。在当前的例子中，此函数检查 SIP 用户同 auth 用户是否相同。这就能够避免一个用户使用另一个用户的认证信息。这些函数通过 URI_DB 模块进行使能。

使用别名 (**Using Aliases**)

有些情况下，你想要允许一个用户拥有多个地址，譬如和一个主地址相关的电话号码。你可以使用别名来实现这个目的。

为了添加别名，使用下面命令：

```

#openerctl alias add flavio@asteriskguide.com sip:1000@asteriskguide.com
database engine 'MYSQL' loaded
Control engine 'FIFO' loaded

```

MySQL password for user 'openser@localhost':

```
lookup("aliases");
```

函数 lookup("aliases") 检查数据库中的别名表，如果一个注册者被找到，就将其别名翻译成正规的地址（订阅列表中的地址）。这个特性还能够被用来将 DIDs 重定向到最终用户。还有一个 Alias_db 模块。它不是从内存而是数据库中直接对别名进行搜索。牺牲了一点性能的好处就是，它能够简化数据库中别名的直接准备。

处理 **CANCEL** 请求和重传（ **Handling CANCEL requests and retransmissions** ）

按照 RFC3261，cancel 请求消息和 INVITE 请求按照同一种方式进行路由。下面的脚本检查 CANCEL 请求是否与现存的事务相匹配，并且关注所有的必要的路由。有时候，我们需要进行一些与现存事务相关的重传。在这种情况下，函数 t_check_trans() 将处理之并退出脚本。

```
#CANCEL processing
if (is_method("CANCEL"))
{
if (t_check_trans())
t_relay();
exit;
}

t_check_trans();
```

带有上面所有资源的完整脚本（ **Full Script with All the Resources Above** ）

```
# ----- module loading -----
#set module path
mpath="//lib/openser/modules/"
# Uncomment this if you want to use SQL database
loadmodule "mysql.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
```



```

loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "mi_fifo.so"
loadmodule "uri.so"
loadmodule "uri_db.so"
loadmodule "domain.so"
# Uncomment this if you want digest authentication
# mysql.so must be loaded !
loadmodule "auth.so"
loadmodule "auth_db.so"
# ----- setting module-specific parameters -----
# -- mi_fifo params --
modparam("mi_fifo", "fifo_name", "/tmp/openssl_fifo")
# -- usrloc params --
#modparam("usrloc", "db_mode", 0)
# Uncomment this if you want to use SQL database
# for persistent storage and comment the previous line
modparam("usrloc", "db_mode", 2)
# -- auth params --
# Uncomment if you are using auth module
#
modparam("auth_db", "calculate_ha1", 0)
#
# If you set "calculate_ha1" parameter to yes,
# uncomment also the following parameter)
#
#modparam("auth_db", "password_column", "password")
# -- rr params --
# add value to ;lr param to make some broken UAs happy
modparam("rr", "enable_full_lr", 1)
# ----- request routing logic -----
# main routing logic route{
# initial sanity checks -- messages with
# max_forwards==0, or excessively long requests
if (!mf_process_maxfwd_header("10")) {
sl_send_reply("483", "Too Many Hops");
exit;
}; if (msg:len >= 2048 ) {
sl_send_reply("513", "Message too big");
exit;
};
# we record-route all messages -- to make sure that
# subsequent messages will go through our proxy; that's

```

```

# particularly good if upstream and downstream entities
# use different transport protocol
if (!method=="REGISTER")
record_route();
# subsequent messages withing a dialog should take the
# path determined by record-routing
if (loose_route()) {
# mark routing logic in request
append_hf("P-hint: rr-enforced\r\n");
route(1);
}; #CANCEL processing
if (is_method("CANCEL")) {
if (t_check_trans()) t_relay();
exit;
}
if (method=="REGISTER") {
route(2);
} else {
route(3);
};
} route[1] {
# send it out now; use stateful forwarding as it works reliably
# even for UDP2TCP
if (!t_relay()) {
sl_reply_error();
};
exit;
}
route[2] {
#
# -- Register request handler --
#
if (is_uri_host_local()) {
if (!www_authorize("", "subscriber")) {
www_challenge("", "1");
exit;
};
if (!check_to()) {
sl_send_reply("403", "Forbidden");
exit;
}; save("location");
exit;
} else if {
sl_send_reply("403", "Forbidden");

```

```

};
}      route[3] {
#
# -- INVITE request handler --
#
if (is_from_local()){
# From an internal domain -> check the credentials and the FROM
if (!proxy_authorize("", "subscriber")) {
proxy_challenge("", "1");
exit;
} else if (!check_from()) {
sl_send_reply("403", "Forbidden, use From=ID");
exit;
};
consume_credentials();
# Verify aliases
lookup("aliases");
if (is_uri_host_local()) {
# -- Inbound to Inbound route(10);
} else {
# -- Inbound to outbound
route(11);
};
} else {
# From an external domain -> do not check credentials
#Verify aliases, if found replace R-URI.
lookup("aliases");
if (is_uri_host_local()) {
#-- Outbound to inbound
route(12);
} else {
# -- Outbound to outbound
route(13);
};
};
}      route[10] {
#from an internal domain -> inbound
#Native SIP destinations are handled using the location table
append_hf("P-hint: inbound->inbound \r\n");
if (!lookup("location")) {
sl_send_reply("404", "Not Found");
exit;
};
route(1);

```

```

}
route[11] {
# from an internal domain -> outbound
# Simply route the call outbound using DNS search
append_hf("P-hint: inbound->outbound \r\n");
route(1);
}
route[12] {
# From an external domain -> inbound
# Verify aliases, if found replace R-URI.
lookup("aliases");
if (!lookup("location")) {
sl_send_reply("404", "Not Found");
exit;
};
route(1);
}
route[13] {
#From an external domain outbound
#we are not accepting these calls
append_hf("P-hint: outbound->inbound \r\n");
sl_send_reply("403", "Forbidden");
exit;
}

```

实验——加强安全性（ **Lab——Enhancing the Security** ）

步骤 1：试着用新的配置注册你的话机。你会注意到在你的话机注册时出现了一个错误。

步骤 2：上面的配置目前使用的是 domain.so 模块。现在，为了认证，域必须在 MySQL 数据库中的域表中。

为了增加一个域，使用 openserctl 工具。

openserctl domain add your-ip-address

openserctl domain add your-domain

对于每一个域重复上面的过程。

步骤 3：再次尝试注册话机。现在注册过程将正常。

实验——使用别名（ **Lab——Using Aliases** ）

步骤 1：为订阅者 1000 添加别名

```
#openserctl alias add john@youripordomain sip:1000@youripordomain
```

```
database engine 'MYSQL' loaded
```

```
Control engine 'FIFO' loaded
```

```
MySql password for user 'openser@localhost':
```

```
|          使用 openserrw 作为密码          |
```

步骤 2：从注册为 1001 的软电话打给 John

通话完整么？为什么？

概要（ **Summary** ）

在这一章中，你已经学会了如何将 MySQL 整合进 OpenSER 中。现在，我们的脚本可以认证用户，检查 TO 和 FROM 头域，按照带内和带外来处理通话了。重要的是要牢记域必须被插入数据库中，因为要支持多域。如果你改变了你的域或 IP 地址，请记着更新你的数据库。

第六章：使用 **SerMyAdmin** 构建用户入口

在上一章中，我们用 MySQL 数据库实现了认证。现在，我们将需要一个工具来帮助用户和管理员们。显然，这个工具必须要比 `openserctl` 容易操作。手动对成百上千的用户进行管理是很困难的，所以用户预置工具在其过程中就变得相当重要。在这一章中，将看到 SerMyAdmin 工具是如何来被创建来帮助构建用户和管理员入口的。

本章末，你将能够：

- 知道为什么需要一个用户入口进行管理
- 安装 SerMyAdmin 和它的依赖
- 配置诸如管理者和用户访问的资源
- 添加和删除域
- 用你公司的颜色和 logo 来定制入口

SerMyAdmin

这段材料本来是写给 SerWeb。SerWeb 原本是为 SER 项目开发的。不幸的是，SerWeb 和最新版本的 OpenSER 不兼容。我们没有选择 SerWeb 的另外一个重要的原因是它被认为有漏洞。OpenSER 的一些网页接口缺少很多选项。其中我们曾发现的一种工具是 OpenSER administrator。这个工具正在使用“Ruby on Rails”开发。似乎这是一个非常好的 OpenSER 服务器管理工具，但是它却不允许与 SerWeb 相同的方式来提供用户接口，并且其缺少多域（multi-domain）支持。OpenSER administrator 能够在 <http://sourceforge.net/projects/openseradmin> 找到。

既然构建 OpenSER 入口的工具找不到，那么我们决定使用 JAVA 建造我们自己的工具，命名为 SerMyAdmin。我们现在已经准备好使用这样的工具来构建本书。它是按照 GPLv2 授权的在 Grails 上开发（Groovy on rails）。可以在 <http://sourceforge.net/projects/sermyadmin> 上下载。

你这儿看到的是一套独立的工具。在我们的轨迹路线中，我们打算将 SerMyAdmin 整合进行 Liferay 入口。使用类似 Liferay 之类的内容管理系统将使你构建入口的工作更加容易。

SerMyAdmin 项目可以在 sermyadmin.sourceforge.net 上找到。它的想法是改进 OpenSER 数据库的管理。SerMyAdmin 在 GPLv2 下许可。



实验——安装 **SerMyAdmin** (**Lab——Installing SerMyAdmin**)

SerMyAdmin 使用 Grails 框架，所以它需要一个应用服务器。你可以自行进行选择，如 IBM WebSphere, JBoss, Jetty, Tomcat 等等。在这本书中，我们使用 Apache Tomcat，因为它是免费的容易安装。因为我们要使用到一些 Java1.5 的特性，我们需要 Sun 的 Java JDK，而不是免费的可供选择的 GCJ。

步骤 1：为 SerMyAdmin 创建管理者：

```
mysql -u root
```

```
use openser
```

```
INSERT INTO 'subscriber' ( 'id' , 'username' , 'domain' , 'password'
, 'first_name' , 'last_name' , 'email_address' , 'datetime_created' ,
'ha1' , 'ha1b' , 'timezone' , 'rpid' , 'version' , 'password_hash' ,
'auth_username' , 'class' , 'domain_id' , 'role_id' )
VALUES (
NULL , 'admin' , 'openser.org' , 'senha' , 'Admin' , 'Admin' , 'admin@
openser.org' , '0000-00-00 00:00:00' , '1' , '1' , '1' , '1' , '1' , NULL ,
'admin@openser.org' , NULL , '1' , '3'
);
```

步骤 2：下一步我们要做的是更新我们的源列表以使用 contrib 仓库和 non-free 包。我们的/etc/apt/sources.list 文件应该看起来是下面这个样子：

```
# /etc/apt/sources.list
```

```
deb http://ftp.br.debian.org/debian/ etch main contrib non-free
```

```
deb-src http://ftp.br.debian.org/debian/ etch main contrib non-free
```

```
deb http://security.debian.org/ etch/updates main contrib non-free
```

```
deb-src http://security.debian.org/ etch/updates main contrib non-free
```

```
/etc/apt/sources.list
```

注意到我们只是在我们的仓库定义后面添加了关键字 contrib 和 non-free。

步骤 3：使用下面的命令来更新包列表。

```
openser:~# apt-get update
```

步骤 4：运行下面的命令安装 Sun's Java1.5

```
openser:~# apt-get install sun-java5-jdk
```

步骤 5：确认你正在使用 Sun's Java。请运行下面的命令以告诉 Debian 你想要使用 Sun's Java 作为你的默认的 Java 应用。

```
openser:~# update-java-alternatives -s java-1.5.0-sun
```

步骤 6：到目前为止，如果每一步都顺利完成，那么你应该运行下面的命令以得到一个类似下面的输出。

```
openser:~# java -version
```

```
java version "1.5.0_14"
```

```
Java(TM) 2 Runtime Environment, Standard Edition (build  
1.5.0_14-b03)
```

```
Java HotSpot(TM) Client VM (build 1.5.0_14-b03, mixed  
mode, sharing)
```

步骤 7：安装 Tomcat。你可以在 <http://tomcat.apache.org/download-60.cgi> 获得 Tomcat。

为了安装 Tomcat，只需要运行下面的命令：

```
openser:/usr/local/etc/openser# cd /usr/local
```



```
openser:/usr/local# wget http://mirrors.uol.com.br/pub/apache/tomcat/tomcat-6/v6.0.16/bin/apache-tomcat-6.0.16.tar.gz
```

```
openser:/usr/local# tar zxvf apache-tomcat-6.0.16.tar.gz
```

```
openser:/usr/local# ln -s apache-tomcat-6.0.16 tomcat6
```

步骤 8：为了在你的服务器初始化中启动 Tomcat，请复制下面的脚本到/etc/init.d/tomcat6。

```
#!/bin/bash -e

#### BEGIN INIT INFO

# Provides: Apache's Tomcat 6.0

# Required-Start: $local_fs $remote_fs $network

# Required-Stop: $local_fs $remote_fs $network

# Default-Start: 2 3 4 5

# Default-Stop: S 0 1 6

# Short-Description: Tomcat 6.0 Servlet engine

# Description: Apache's Tomcat Servlet Engine

### END INIT INFO

#

# Author: Guilherme Loch Góes <glwgoes@gmail.com>

#

set -e

PATH=/bin:/usr/bin:/sbin:/usr/sbin:

CATALINA_HOME=/usr/local/tomcat6

CATALINA_BIN=$CATALINA_HOME/bin

test -x $DAEMON || exit 0
```

```
. /lib/lsb/init-functions

case "$1" in

start)

echo "Starting Tomcat 6" "Tomcat6"

$CATALINA_BIN/startup.sh

log_end_msg $?

;;

stop)

echo "Stopping Tomcat6" "Tomcat6"

$CATALINA_BIN/shutdown.sh

log_end_msg $?

;;

force-reload|restart)

$0 stop

$0 start

;;

*)

echo "Usage: /etc/init.d/tomcat6 {start|stop|restart}"

exit 1

;;

esac

exit 0
```

告诉 Debian 在系统启动时运行你的脚本；我们使用下面的命令实现之：

```
openser: chmod 755 /etc/init.d/tomcat6
```

```
openser:/etc/init.d# update-rc.d tomcat6 defaults 99
```

步骤 10: 为了确认每一步都运行正确, 重启服务器, 并且在你的浏览器中打开 URL <http://localhost:8080>; 如果一切顺利的话, 你会看到 Tomcat 的开始页面。

步骤 11: 为 Tomcat 安装 MySQL 驱动使得 SerMyAdmin 可以访问你的数据库。这个驱动可以在 <http://dev.mysql.com/downloads/connector/j/5.1.html> 上找到。你可以下载驱动然后解包, 之后将 connector 复制到 Tomcat 的共享库文件夹下, 如下所示:

```
openser:/usr/src# tar xzf mysql-connector-java-5.1.5.tar.gz
```

```
openser:/usr/src# cp mysql-connector-java-5.1.5/mysql-connector-java-5.1.5-bin.jar /usr/local/tomcat6/lib
```

步骤 12: 为 SerMyAdmin 声明数据源以使其链接到 OpenSER 的数据库上。你可以在 `/usr/local/tomcat6/conf/context.xml` 上做些修改以实现之: 文件看起来是下面这个样子:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<Context path="/serMyAdmin">
```

```
<Resource auth="Container" driverClassName="com.mysql.jdbc.Driver" maxActive="20"
maxIdle="10" maxWait="-1" name="jdbc/openser_MySQL" type="javax.sql.DataSource"
url="jdbc:mysql://localhost:3306/openser" username="sermyadmin"
password="secret"/>
```

```
</Context>
```

在上面的文件中, 请按照你的环境修改高亮显示的参数。SerMyAdmin 可以安装在其他的机器上, 而不仅仅是拥有数据库的服务器上。当规模需要的时候可以这样做。在 Debian 上默认安装后的 MySQL 只接收本地请求, 所以你应该修改文件 `/etc/mysql/my.cnf`, 让 MySQL 能够接收来自外部主机的请求。

步骤 13: 创建一个用户 (可以参照文件 `context.xml`)。这个用户需要访问该数据库的权限。请运行下面的命令:

```
openser:/var/lib/tomcat5.5/conf# mysql -u root -p
```

Enter password:

Welcome to the MySQL monitor. Commands end with ; or \g.

Your MySQL connection id is 14

Server version: 5.0.32-Debian_7etch5-log Debian etch distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

```
mysql> grant all privileges on opener.* to sermyadmin@'%' identified by 'secret';
```

Query OK, 0 rows affected (0.00 sec)

步骤 14：我们已经接近完成。下一步工作就是部署 SerMyAdmin WAR 文件了。请下载并复制文件 serMyAdmin.war 到 Tomcat 的 webapps 文件夹下。重启以激活更改。

```
openser:/usr/src# cp serMyAdmin-0.4.war /usr/local/tomcat6/webapps/serMyAdmin.war
```

```
openser:/usr/src# invoke-rc.d tomcat6 restart
```

不用担心数据库的修改；SerMyAdmin 将自动的为你进行处理以适应其改变。

步骤 15：配置 Debian 的 MTA (Message Transfer Agent) 以允许 SerMyAdmin 发送一封确认邮件给新用户。运行下面的命令配置 Exim4 (Debian 默认的 MTA)。询问你公司的邮件管理员。

```
openser:/# apt-get install exim4
```

```
openser:/# dpkg-reconfigure exim4-config
```

映入眼帘的是一个以对话框为基础的配置菜单；在这个菜单上，要着重注意两个选项：邮件配置的通用类型 (General type of mail configuration)，这个选项应该被配成 Internet Site，这样我们就可以直接使用 SMTP 来发送和接收邮件；转发邮件的域，这个选项应该被配成从 SerMyAdmin 发出的邮件你想要其表现的来自何处的域。

步骤 16：定制文件

/usr/local/apache-tomcat-6.0.16/webapps/serMyAdmin-0.3/WEB-INF/spring/resource.xml，这个文件包含指定使用哪个 email 服务器来发送邮件的参数和希望如何显示这些邮件的出处的参数。下面是这个文件的一个例子：

```

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.0.xsd">

<bean id="mailSender" class="org.springframework.mail.javamail.JavaMailSenderImpl">

<property name="host"><value>localhost</value></property>

</bean>

<!-- You can set default email bean properties here, eg: from/to/subject -->

<bean id="mailMessage" class="org.springframework.mail.SimpleMailMessage">

<property name="from"><value>admin@sermyadmin.org</value></property>

</bean>

</beans>

```

第一个要改变的参数是我们用来发送邮件的服务器。第二个则是那些邮件要显示的从何而来的参数。

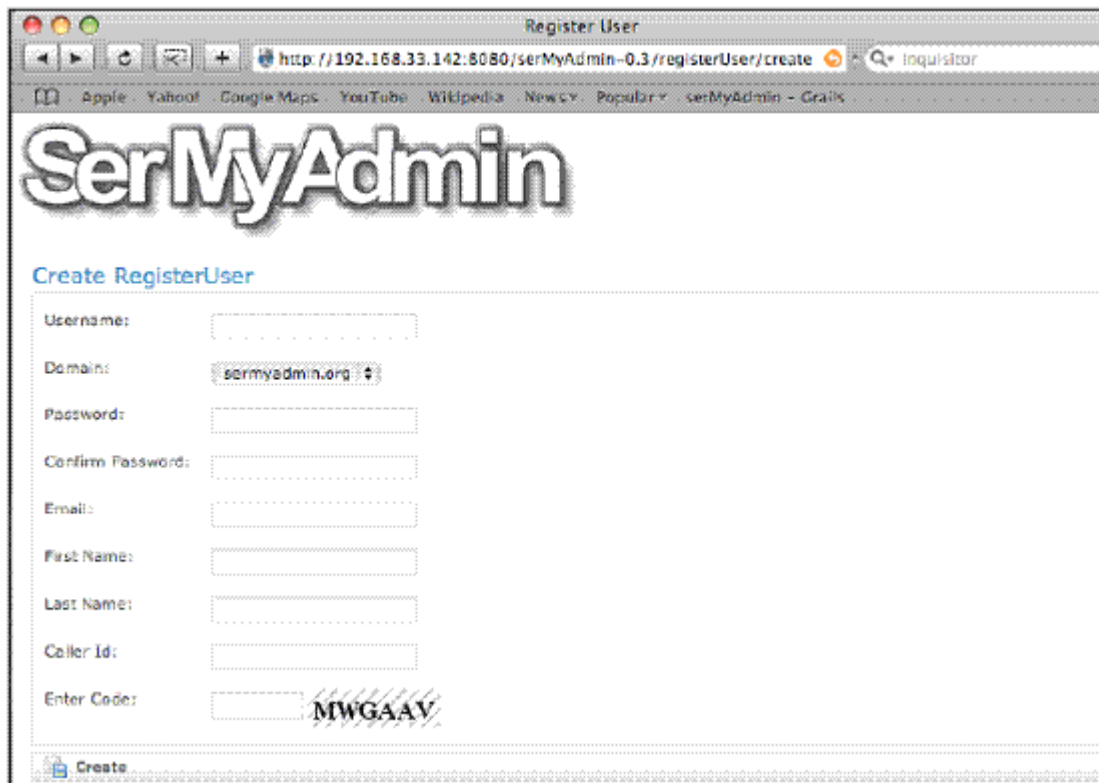
再次重启 Tomcat，此时我们已经准备好要开始了。当你将浏览器指向 `http://<server address>:8080/serMyAdmin`，你将看到登录页面，正如我们在本章开始时呈现的那样。

基本任务（ **Basic Tasks** ）

现在你能够使用 SerMyAdmin 来做很多工作了。在这一章中我们将向你展示如何创建和管理新的用户和组。下一章，我们将使用 SerMyAdmin 做些其他工作，如管理信任列表和 LCR 模块。

注册一个新用户（ **Registering a New User** ）

只需简单的在登录页面上的 Register 按钮点击一下就可以注册一个新的用户了。



填上下面这些，Username，Password，Domain，Email，First Name，Last Name，Caller Id 和确认码（confirmation code）。按下屏幕的最下方的 Create 按钮。用户将被加入数据库。系统管理员和该用户都将收到一封关于注册的电子邮件。在用户可以打电话前，管理员必须要同意该用户。

同意新用户（**Approving a New User**）

按照下面的步骤一步一步做来同意一个新用户：

步骤 1：用 admin@localhost 帐号登录，密码是在 OpenSER 安装时创建的 openserrw。安装时已经为每一个用户创建了叫做 Role 的新的属性。这个属性的目的是用来区分普通用户，域管理员和通用管理员的。Admin 用户自动被设置为通用管理员（Global Administrator）。这个新的属性将帮助我们提供多域的支持。

步骤 2：选择菜单中 Registered Users 条目。



在上面的屏中，选择你想要添加用户，选中对应复用框。按下 Approve 按钮添加用户；该用户将从 register_user 表中移除，而被转移到订阅列表中，然后他将能够注册到 OpenSER 并可以打电话。

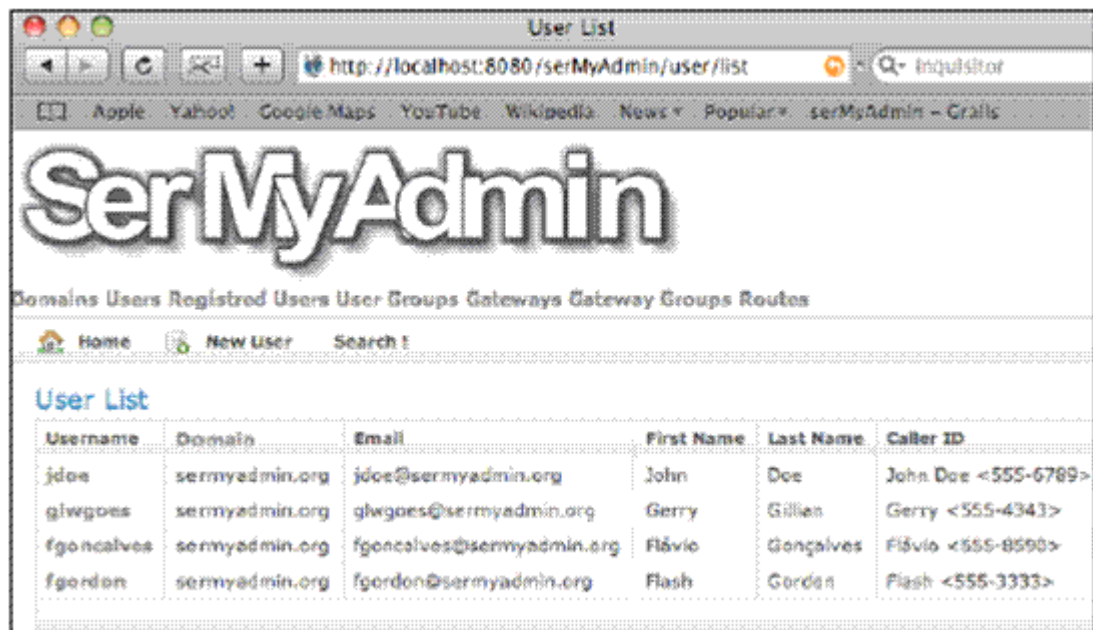


步骤 3：现在用户会显示在你的用户列表中了。点击菜单目录中的用户检查之。



用户管理 (User Management)

你能够查看，添加，编辑和删除用户菜单上的用户了。当你点击它时，所有的用户都会显示在你的系统上。



Username	Domain	Email	First Name	Last Name	Caller ID
jdoe	sermyadmin.org	jdoe@sermyadmin.org	John	Doe	John Doe <555-6789>
glwgoes	sermyadmin.org	glwgoes@sermyadmin.org	Gerry	Gillian	Gerry <555-4343>
fgoncalves	sermyadmin.org	fgoncalves@sermyadmin.org	Flávio	Gonçalves	Flávio <555-8598>
fgordon	sermyadmin.org	fgordon@sermyadmin.org	Flash	Gordon	Flash <555-3333>

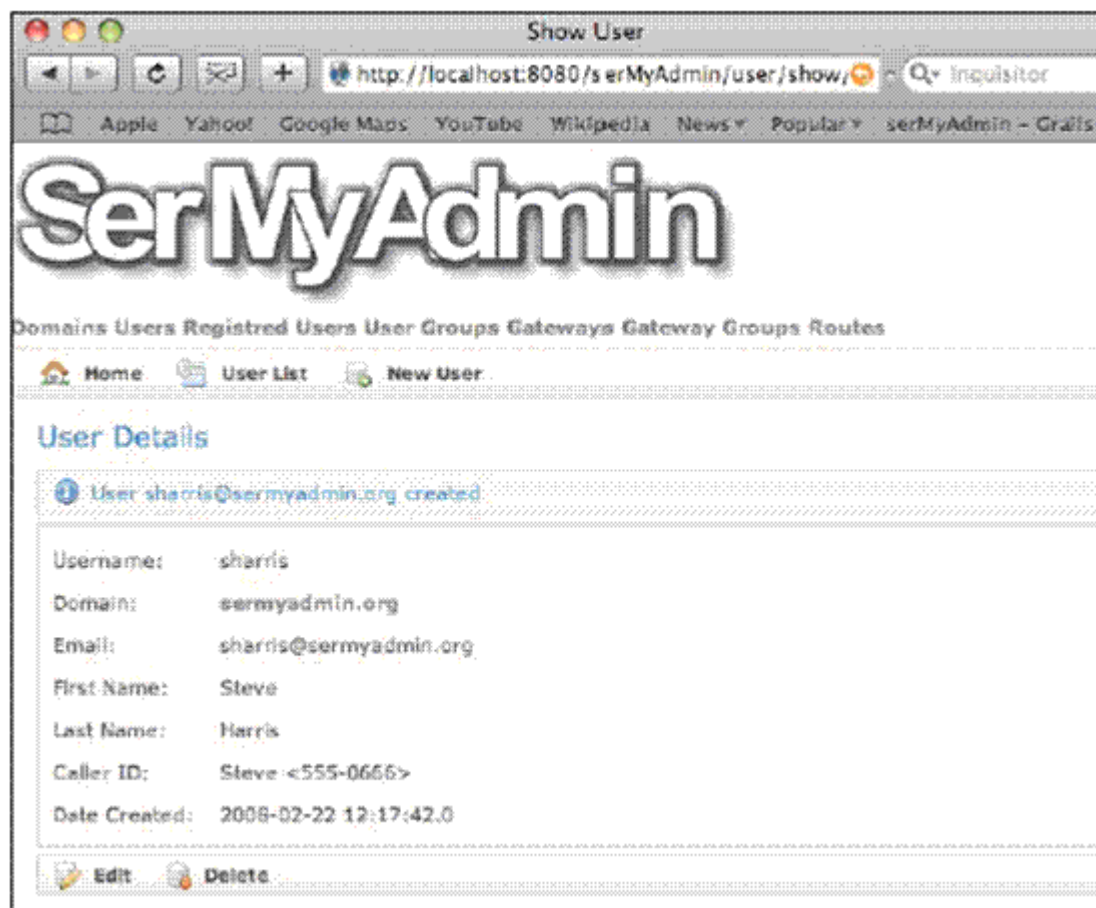
要添加新用户的话，必须要点击“New User”链接。点后你将转到下面的页面上：

The screenshot shows a web browser window titled "Create User" with the URL `http://localhost:8080/serMyAdmin/user/create`. The browser's address bar and tabs are visible. The page features the "SerMyAdmin" logo and a navigation menu with links: Domains, Users, Registered Users, User Groups, Gateways, Gateway Groups, and Routes. Below the navigation menu, there are links for "Home" and "User List". The main content area is titled "Create User" and contains a form with the following fields:

- Username:
- Domain:
- Password:
- Email:
- Group:
- First Name:
- Last Name:
- Caller ID:

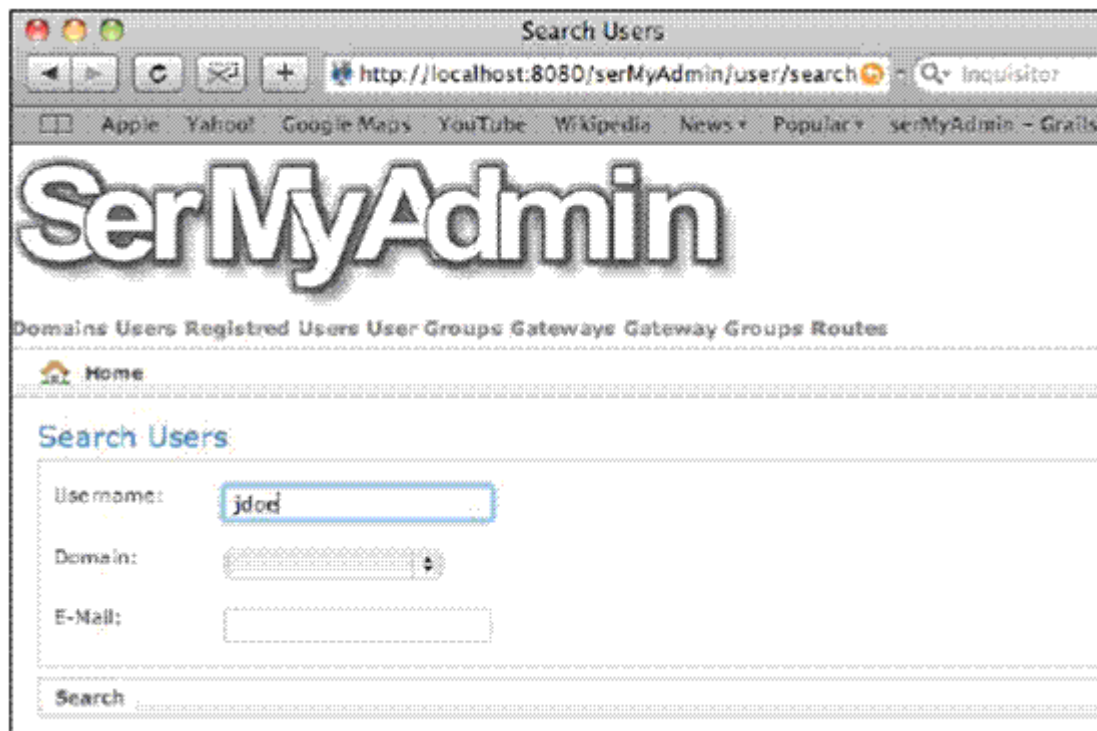
At the bottom of the form is a "Create" button.

你要完全的填满上一张图片中的空档，并点击“Create”链接，然后该用户会被添加到订阅表（subscriber table）中。之后会显示下面的图片：



在这个页面中，你可以点击“Edit”来编辑插入的信息，点击“Delete”来删除该用户。

在“User List”页面中，你可以基于 username，domain，和 email 对用户进行搜索；只要填上所需的条件，然后点击“Search”链接即可。下面的页面中，我们使用用户名 jdoe 来对用户进行搜索。点击“Search”；页面将转移到符合你所输条件的用户列表（user list）上去。



域管理（**Domain Management**）

你可以像管理你的用户那样来管理你的域。点击“Domains”得到一张域列表（domain list）。你可以在其中进行诸如添加一个新的域，删除一个已经存在的域等类似的操作。这里要注意的很重要的一点就是，SerMyAdmin 不允许没有域的用户存在，所以当你删掉一个域后，你也就删除掉了所有属于该域的所有用户。

接口定制（**Interface Customization**）

网站的布局，SerMyAdmin 使用 SiteMesh 框架，所以按照你自己的品位来定制 SerMyAdmin 的显示效果是非常简单的。SiteMesh 显示的页面是基于一份模板，这份模板可以在

openser:/usr/local/apache-tomcat-6.0.16/webapps/serMyAdmin-0.3/WEB-INF/grails-app/view/layouts 中找到。在这里，你将找到 main.gsp 和 notLoggedIn.gsp 文件，这些文件是用来控制页面如何显示的 Groovy 服务器页面（Groovy Server Pages）。

SiteMesh 使用 HTML 元标记来选择要使用哪个布局；这些标记应该可以在每个页面的头元素（head element）中找到，如果一个页面在头元素中含有标

记<meta content="main" name="layout"/>，那么 SiteMesh 将使用 main.gsp 布局显示之。

你可以按照你的意愿对 main.gsp 和 notLoggedIn.gsp 进行修改，但是要了解的很重要的一点是<g:layoutHead />和<g:layoutBody />将使用这个布局来保持页面的 head 和 body 标志。另外一件要知道的事情是<g:render template="/menu" />被用来整理页面碎片；这些页面碎片是 GSP 文件，它们的文件名以下划线打头。

为了使用你自己的 logo 来替代 SerMyAdmin 的，只需要将你的 logo 放到/usr/local/apache-tomcat-6.0.16/webapps/serMyAdmin-0.3/images，并对布局文件中的指向 logo_voffice.png 的标志进行修改，如下：

```
<div class="logo"></div>
```

在上面我们仅仅是通过改变粗体显示的参数就用我们自己的 logo 替代了 SerMyAdmin 的 logo。

你也可以通过修改 CSS 文件来改变 SerMyAdmin 的显示效果和感觉，CSS 文件可以在/usr/local/apache-tomcat-6.0.16/webapps/serMyAdmin-0.3/css 中找到；在 main.css 中，我们将能够发现每一个改变 SerMyAdmin 行为的类（class）。

例子：如果我们使用下面的参数来改变这个文件中的背景类（background class）：

```
body {  
  background: #00f;  
  color: #333;  
  font: 8px verdana, arial, helvetica, sans-serif;  
  
}
```

我们将能够在最后得到如下的页面：



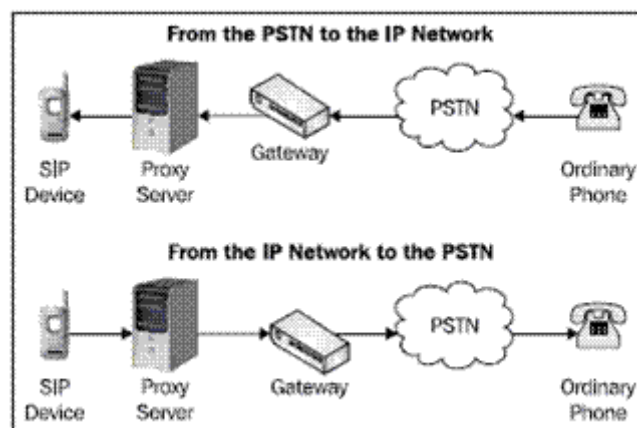
这个页面不是世界上最好看的，但是你可以按照这个例子的方法来使它看起来更好一些。

概要（ **Summary** ）

在这一章中，你已经学到了为什么拥有一个用户和管理入口是如此的重要。它是软件中你应该放大部分精力在其上的部分。但是一些 VoIP 服务提供商并没有分配必要的时间和资源在构建入口这项重要的工作上。OpenSER 是一个令人惊讶的 SIP 代理,但是 SIP 代理也只是 VoIP 服务提供商的部件的一员而已。没有好的管理和用户接口，VoIP 提供项目很容易失败。SerMyAdmin 就是我们在此方面对你项目的一个贡献。它是利用 JAVA 语言，使用 Groovy on Rails 架构进行开发，按照 GPL 版本 2 进行许可的系统。你已然已经学会了如何安装，管理用户和域和如何定制显示效果。这个工具可以做的事情还很多，在下一章我们将向你展示其更多的功能。

第七章：与 **PSTN** 的连通（**Connectivity to the PSTN**）

在前两章中，我们已经使用认证（authentication）和数据库为 OpenSER 处理通话做好了准备。SerMyAdmin 用来处理数据记录。然而，你仍然不能打给普通电话，因为你没有连上 PSTN。现在的挑战是如何将通话由 PSTN 路由进来和如何将通话路由到 PSTN（Public Switched Telephone Network）



为了能将通话路由到 PSTN,你需要一个叫做 SIP PSTN 网关（SIP PSTN Gateway）的设备。在市面上，有很多生产这种设备的厂家，诸如 Cisco，AudioCodes，Nortel，Quintum 等等。你也可以使用 Asterisk PBX 完成这个工作。Asterisk 是一个你绝对能够承担的起的网关，而且与上面提到的各个厂商的设备兼容。它完全是开源的，也是按照 GPL 许可的。

这一章的结束，你将能够：

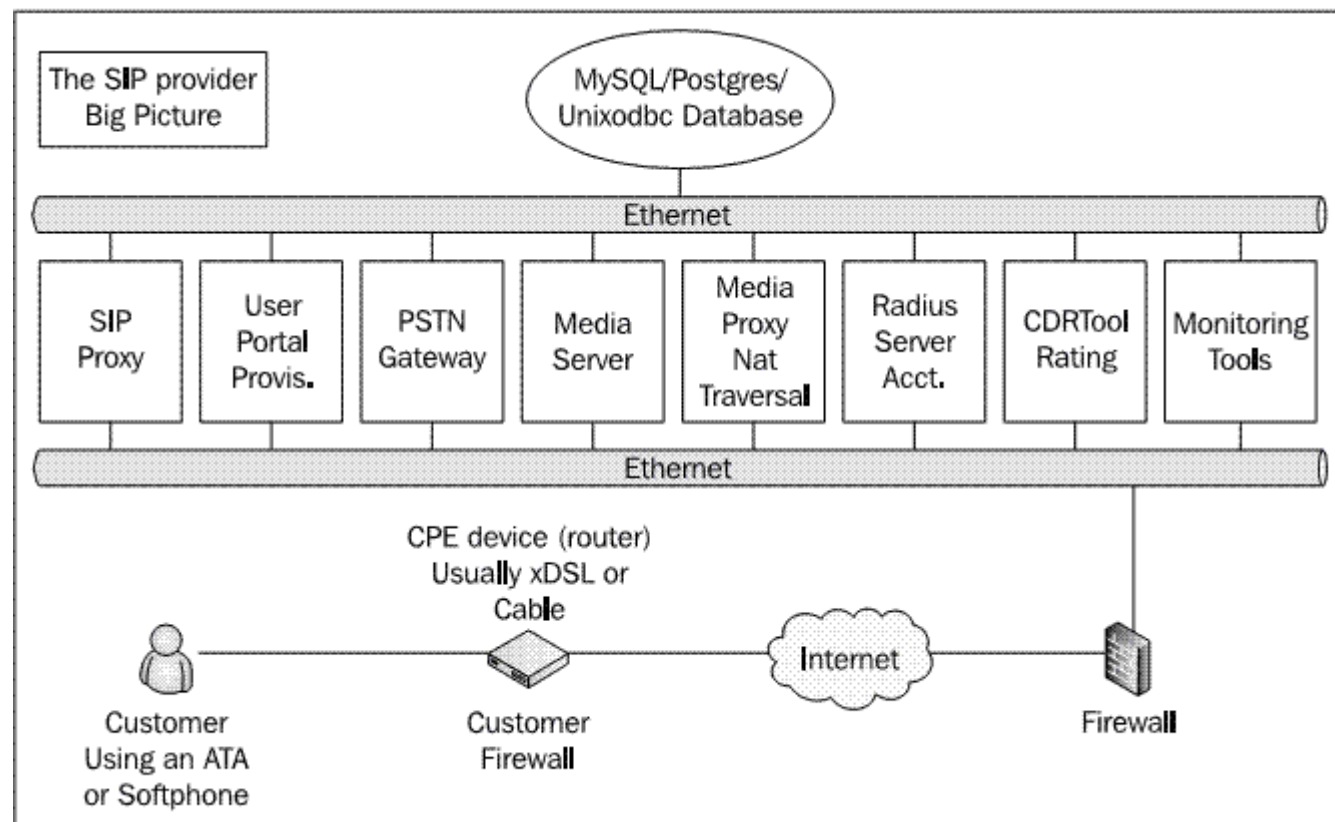
- 将 OpenSER 与 SIP 网关连上
- 将认证应用到带内通话
- 使 ACLs 防止 PSTN 网关不被没有经过认证的用户使用
- 使用 LCR（Least Cost Route）模块路由你的通话
- 使用 SerMyAdmin 来管理授信主机（Trusted Hosts），网关（Gateways）和路由器（Routes）

在这一章中，你将学会如何将通话打到 PSTN。我们将介绍三个新的模块（LCR，PERMISSIONS，还有 GROUP），他们将帮助你路由这些通话并保证他们的安全。你可以在互联网上轻易的找到关于 regexps 的指南。如果你对

regular expressions 或 regexps 不熟悉, <http://www.visibone.com/regular-expressions/>这条链接可以作为参考。

我们在哪儿？（ **Where Are We ?** ）

VoIP 服务提供商的方案中有很多的部件。为了避免迷失，我们将在每一章中展示下面的这张图片。在这一章中，我们将利用 SIP 代理部件和 PSTN 网关一起工作。



本章之后，我们的 VoIP 提供者将能够使用 SIP 网关将通话打给 PSTN。

发往网关的请求（ **Requests Sent to the Gateway** ）

在标明给网关的请求中，我们必须验证该用户属于哪一组（group），以查看其是否被允许使用 PSTN。

```

if (uri=~"^sip:[2-9][0-9]{6}@") {
    if (is_user_in("credentials","local")) {
        route(5);
        exit;
    } else {
        sl_send_reply("403", "No permissions for local calls");
        exit;
    };
};
if (uri=~"^sip:[2-9][0-9]{9}@") {
    if (is_user_in("credentials","ld")) {
        route(5);
        exit;
    } else {
        sl_send_reply("403", "No permissions for long distance calls");
        exit;
    };
};
if (uri=~"^sip:011[0-9]*@") {
    if (is_user_in("credentials","int")) {
        route(5);
        exit;
    } else {
        sl_send_reply("403", "No permissions for international calls");
    };
};
};

```

要达到这个目的，我们要使用到‘group’模块。这个暴露了函数 `is_user_in("credentials", "group")` 用来检查用户是否属于指定组。在上面的例子中，我们已经创建了 3 个组：local 代表本地通话，ld 代表长途，int 代表国际长途。脚本中，我们使用正则表达式（regular expressions）来检查通话是属于上面介绍的三种中的哪一种。

你必须将这些组插进叫做 group 的 MySQL 表中才能使用它。你可以很容易的插入，删除，显示组成员（group membership）：

- `Openserctl acl show [<username>]`
- `Openserctl acl grant <username> <group>`
- `Openserctl acl revoke <username> [<group>]`

使用 SerMyAdmin 来管理你的表也是可能的。要添加和删除组，你可以浏览“User Groups”部分，在那里，你可以添加，删除组（groups）。

要改变用户的组成员，你可以在下面显示的用户菜单菜单中进行编辑：

使用检查栏选择用户属于的这儿显示的指点的组。

来自网关的请求（**Requests Coming From the Gateway**）

现在，我们要使用 PERMISSIONS 模块来对来自没有摘要认证过程
PSTN 网关的通话进行授权。

我们将使用的 `allow_trusted()` 函数有 `PERMISSIONS` 模块曝露出来。许可模块可以被用来授权 (`authorize`) `REGISTER`, `REFER`, 和 `INVITE` 请求。我们可以用 `permissions.allow`, `permissions.deny`, `register.allow`, 和 `register.deny` 文件来对其进行管理和调节。然而这个模块中使用的 `allow_trusted()` 函数要将请求的源 IP 地址同我们数据库中的授信表中的数据进行比较检查。

当通话到来时，函数 `allow_trusted` 要试着去找到一条符合该请求的规则。该规则包含下面的域<IP source address>， <transport protocol>， <regular expression>。

如果下面的规则存在一条，则接受该请求：

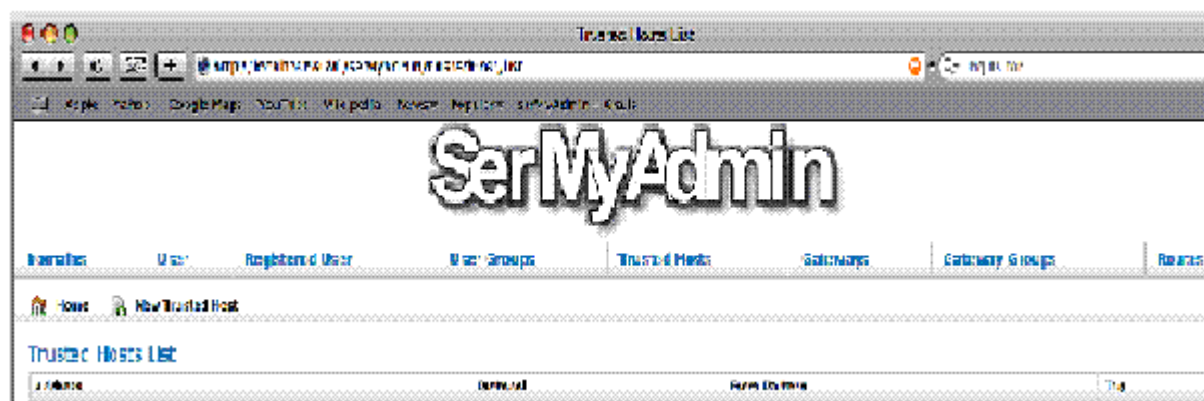
- Ip 地址和请求的源 IP 地址相同
- 传输层协议是“any”或是同请求的传输层协议相符
- 正则表达式为空或符合请求

对于网关来说，不注册到 SIP 代理上是很正常的事。因此，来自网关的请求不应该接收“407 Proxy Authentication Required”响应。在我们目前的脚本中，所有来自我们域的 INVITE 请求都要求他们带有凭据。然而，如果像从网关发出的请求，没有凭据发出，从而通话将会失败。所有，为了修正这一点，我们使用 `allow_trusted()` 函数检查源 IP 地址的方式，而不是去检查凭据。

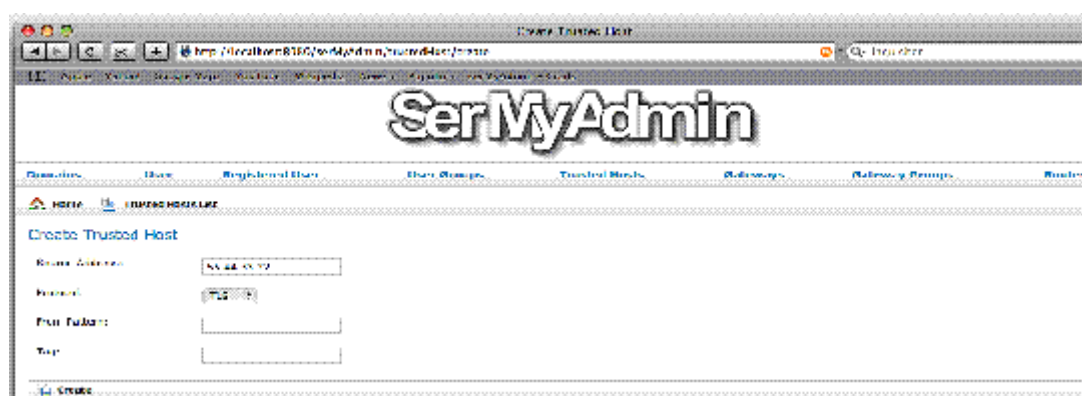
不要忘了将授信的 IP 地址插入我们 MySQL 数据库的授信表中，

这样我们的脚本才能工作。

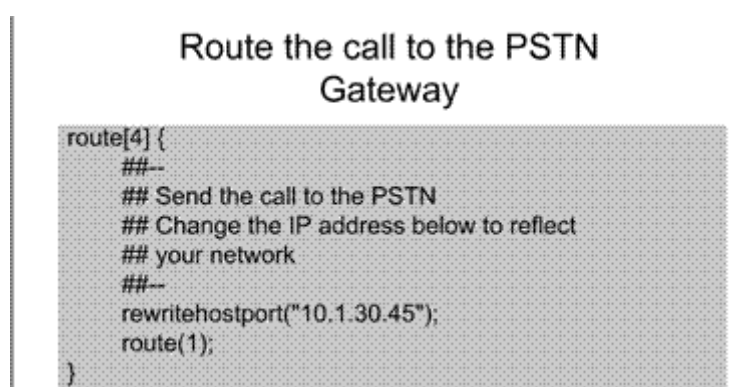
你可以使用 SerMyAdmin 来查看，更新授信主机列表（trusted hosts list）。使用如下显示的授信主机菜单：



要添加新的主机，只需简单的点击“New Trusted Host”菜单选项。



使用函数 `rewritehostport()` 前转通话到 PSTN 网关。



这个脚本的名字为 `openser.pstn`。可以在 <http://www.sermyadmin.org/openser> 找到。一份拷贝展示如下。之前的脚本修改处使用高亮显示。

```
# ----- module loading -----

#set module path

mpath="//lib/openser/modules/"

loadmodule "mysql.so"

loadmodule "sl.so"

loadmodule "tm.so"

loadmodule "rr.so"

loadmodule "maxfwd.so"

loadmodule "usrloc.so"

loadmodule "registrar.so"

loadmodule "textops.so"

loadmodule "uri.so"

loadmodule "uri_db.so"

loadmodule "domain.so"

loadmodule "permissions.so"

loadmodule "group.so"

loadmodule "mi_fifo.so"

# Uncomment this if you want digest authentication

# mysql.so must be loaded !

loadmodule "auth.so"

loadmodule "auth_db.so"

# ----- setting module-specific parameters -----

modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
```

```

modparam("usrloc", "db_mode", 2)

modparam("auth_db", "calculate_ha1", yes)

modparam("auth_db", "password_column", "password")

modparam("rr", "enable_full_lr", 1)

modparam("auth_db|permissions|uri_db|usrloc","db_url",
        "mysql:// opener:openserrw@localhost/openser")

modparam("permissions", "db_mode", 1)

modparam("permissions", "trusted_table", "trusted")

# ----- request routing logic -----

# main routing logic

route{

    #

    # -- 1 -- Request Validation

    #

    if (!mf_process_maxfwd_header("10")) {

        sl_send_reply("483","Too Many Hops");

        exit;

    };

    if (msg:len >= 2048 ) {

        sl_send_reply("513", "Message too big");

        exit;

    };

    #

```

```

# -- 2 -- Routing Preprocessing

#

## Record-route all except Register

if (!method=="REGISTER") record_route();

##Loose_route packets

if (loose_route()) {

    # mark routing logic in request

    append_hf("P-hint: rr-enforced\r\n");

    route(1);

};

#CANCEL processing

if (is_method("CANCEL")) {

    if (t_check_trans()) t_relay();

    exit;

};

t_check_trans();

#

# -- 3 -- Determine Request Target

#

if (method=="REGISTER") {

    route(2);

} else {

    route(3);

```

```

        };
    }

    route[1] {

        #

        # -- 4 -- Forward request to target

        #

        ## Forward statefully

        if (!t_relay()) {

            sl_reply_error();

        };

        exit;
    }

    route[2] {

        ## Register request handler

        if (is_uri_host_local()) {

            if (!www_authorize("", "subscriber")) {

                www_challenge("", "1");

                exit;

            };

            if (!check_to()) {

                sl_send_reply("403", "Forbidden");

                exit;

            };

```

```

        save("location");

    exit;

} else if {

    sl_send_reply("403", "Forbidden");

};

}

route[3] {

    ## INVITE request handler

    if (is_from_local()){

        # From an internal domain -> check the credentials

        and the FROM

        if(!allow_trusted()){

            if (!proxy_authorize("", "subscriber")) {

                proxy_challenge("", "1");

                exit;

            } else if (!check_from()) {

                sl_send_reply("403", "Forbidden, use From=ID");

                exit;

            };

        } else {

            log("Request bypassed the auth.using allow_trusted");

        };

    };

```

```

consume_credentials();

#Verify aliases, if found replace R-URL.

lookup("aliases");

if (is_uri_host_local()) {

    # -- Inbound to Inbound

    route(10);

} else {

    # -- Inbound to outbound

    route(11);

};

} else {

    #From an external domain ->do not check credentials

    #Verify aliases, if found replace R-URL.

    lookup("aliases");

    if (is_uri_host_local()) {

        #-- Outbound to inbound

        route(12);

    } else {

        # -- Outbound to outbound

        route(13);

    };

};

```



```

}

route[4] {

    # routing to the public network

    rewritehostport("10.1.30.45");

    route(1);

}

route[10] {

    #from an internal domain -> inbound

    #Native SIP destinations are handled using the location table

    #Gateway destinations are handled by regular expressions

    append_hf("P-hint: inbound->inbound \r\n");

    if (uri=~"^sip:[2-9][0-9]{6}@") {

        if (is_user_in("credentials","local")) {

            route(4);

            exit;

        } else {

            sl_send_reply("403", "No permissions for local calls");

            exit;

        };

    };

    if (uri=~"^sip:1[2-9][1-9]{9}@") {

        if (is_user_in("credentials","ld")) {

            route(4);

```

```

        exit;

    } else {

        sl_send_reply("403", "No permissions for long distance");

        exit;

    };

};

if (uri=~"^sip:011[0-9]*@" ) {

    if (is_user_in("credentials","int")) {

        route(4);

        exit;

    } else {

        sl_send_reply("403", "No permissions for

            international calls");

    };

};

if (!lookup("location")) {

    sl_send_reply("404", "Not Found");

    exit;

};

route(1);

}

route[11] {

    # from an internal domain -> outbound

```

```
# Simply route the call outbound using DNS search

append_hf("P-hint: inbound->outbound \r\n");

route(1);

}
```

```
route[12] {

    # From an external domain -> inbound

    # Verify aliases, if found replace R-URI.

    lookup("aliases");

    if (!lookup("location")) {

        sl_send_reply("404", "Not Found");

        exit;

    };

    route(1);

}
```

```
route[13] {

    #From an external domain outbound

    #we are not accepting these calls

    append_hf("P-hint: outbound->inbound \r\n");

    sl_send_reply("403", "Forbidden");

    exit;

}
```

openser.cfg 检查 (opener.cfg Inspection)

PERMISSIONS 模块曝露了一些重要的函数以对到我们 SIP 代理的访问进行控制。其中之一是 `allow_trusted()`，这个函数允许我们控制网关使用 IP 地址访问代理，而不是使用认证凭据。授信表 (trusted table) 是授信地址的存储库。你应该将每个网关的 IP 地址和传输层协议插入这个数据库。这就使得来自网关的请求避免了标准摘要认证的进行。

PERMISSIONS 模块还有些标准的许可和拒绝文件。我们这次不使用这些特性。为了不要日志中的这些信息，请将 PERMISSIONS 模块的 config 文件夹下的文件拷贝到 `/etc/openser` 文件夹下。

```
cp /usr/src/openser-1.2.2/modules/permissions/config/* /etc/openser
```

在许可文件中，基于正则表达式来过滤请求是可能的，这可以改进环境的安全性。检查用例文件是不是符合正确的句法。

group.so 模块用来检查用户的组成员资格。这个叫做 ACL (Access Control List)。你可以使用 `openserctl` 工具 (如下) 来添加，删除或是显示用户 ACLs。

```
loadmodule "permissions.so"
```

```
loadmodule "group.so"
```

下面的第一行告诉模块到哪去寻找传递需要凭据的数据库。第二行提示模块使用数据库上的缓存 (cache) 访问以增加性能。

```
modparam("auth_db|permissions|uri_db|usrloc","db_url",  
"mysql://openser:openserrw@localhost/openser")  
  
modparam("permissions", "db_mode", 1)
```

当的代理服务器收到 INVITE 请求后，通常的行为是向 UAC 请求凭据。然而，PSTN 网关通常并不对认证进行响应。因此，你需要采取特殊的处理过程。函数 `allow_trusted()` 将 INVITE 请求的源 IP 地址同数据库中授信表进行对比检查。如果符合，则接受之。如果不符合，则再请求凭据。

```
if(!allow_trusted()){  
    if (!proxy_authorize("", "subscriber")) {
```

```

        proxy_challenge("", "0");

        exit;

    } else if (!check_from()) {

        sl_send_reply("403", "Forbidden, use FROM=ID");

        exit;

    };

};

```

| 网关的 IP 地址插入数据库是很重要的 |

你可以使用诸如 SerMyAdmin 或是 phpMyAdmin 的工具来维护数据库。这样可比手动在 MySQL 的 CLI (Command line interface) 操作容易的多。

在授信表中，插入的是网关的 IP 地址，传输层协议 (udp, tcp, tls, any) 和正则表达式。

下面是我们按照正则表达式进行通话的路由：

```

if (uri=~"^sip:[2-9][0-9]{6}@") {

    if (is_user_in("credentials", "local")) {

        route(4);

        exit;

    } else {

        sl_send_reply("403", "No permissions for local calls");

        exit;

    };

};

```

```

if (uri=~"^sip:1[2-9][0-9]{9}@") {

    if (is_user_in("credentials","ld")) {

        route(4);

        exit;

    } else {

        sl_send_reply("403", "No permissions for long distance");

        exit;

    };

};

```

```

if (uri=~"^sip:011[0-9]*@") {

    if (is_user_in("credentials","int")) {

        route(4);

        exit;

    } else {

        sl_send_reply("403", "No permissions for

            internat. calls");

        exit;

    };

};

```

本地通话（local call）使用数字 7 分辨并且以 2 到 9 之间的数开头（“^sip:[2-9][0-9]{6}@”）。长途号码符合这条正则 “^sip:1[2-9][0-9]{9}@”，号码以 1 开头，

后面紧接这 2-9 之间的数，加起来一共 9 个数字，这样的号码被认为是长途号码。最后，国际长途号码以 011 + 国家编码 + 地区编码 + 电话号码作为前缀。所有的情况下，脚本都会被抛到路由 4 (route 4)

| 将 ACL 数据插入数据库，对于脚本的正常工作很重要 |

你可以使用 openserctl 工具，SerMyAdmin 或是 phpMyAdmin 来完成。

最后，我们让路由块 4 (routing block 4) 来处理 PSTN 的目的地。函数 `rewritehostport ()` 用来改变 URI 的主机部分，也就是说当你使用 `t_relay ()` 中继请求时，此请求将被发往网关。

```
route[4] {  
  
    ##--  
  
    ## PSTN gateway handling  
  
    ##--  
  
    rewritehostport("10.1.30.45");  
  
    route(1);  
  
}
```

实验——用 Asterisk 做 PSTN 网关 (Lab—— Using Asterisk as a PSTN Gateway)

使用本章提供的框架，写一个脚本将打给 PSTN 的通话传给 PSTN 网关。你可以使用 phpMyAdmin 或是 MySQL 命令行向数据库插入数据。

步骤 1——使用 SerMyAdmin 向授信表中添加网关地址：



如果需要或是如果你觉得方便，可以使用 MySQL 命令行接口来完成同样的事情：

```
#mysql -u opener -p
```

```
-- enter your mysql password --
```

```
mysql> use opener;
```

```
mysql> INSERT INTO trusted ( src_ip, proto, from_pattern )
```

```
VALUES ( '10.1.30.22', 'any', '^sip:.*$');
```

上面的记录告诉 OpenSER 脚本要允许符合下面条件的请求：来自 IP 地址 10.1.30.22，可以是任何的传输层协议，与正则表达式“^sip:.*\$”相符合。如果你不想重新加载 OpenSER，你可以使用下面的命令。

```
#openserctl fifo trusted_reload
```

步骤 2——在域表中包含你的服务域。（如果之前你没有做过）

```
openserctl domain add sermyadmin.org
```

你也可以使用 SerMyAdmin 来做这件事。



步骤 3——在组中包含用户（local,ld 和 int）

```
#openserctl acl grant 1000@sermyadmin.org local
```

```
#openserctl acl grant 1000@sermyadmin.org ld
```

```
#openserctl acl grant 1000@sermyadmin.org int
```

```
#openserctl acl grant 1001@sermyadmin.org local
```

要使用 SerMyAdmin，转向下面这一屏：



步骤 4——配置 Asterisk 做网关。

对于 OpenSER 来说，两个非常流行的网关是 Asterisk 和 Cisco AS5300。

其他厂商的网关也可以使用；检查他们的文档说明作为配置的指导。让我们看看怎样配置带两个 FXO 的 Cisco 2601 和带一个 E1 PSTN 卡 Asterisk。

| 警告：防止将 SIP 包直接发送给网关是很重要的。SIP 网关 |
| 应该在 gateway 的前面，并且要有防火墙防止用户直接将 SIP |
| 请求发送给网关。 |

步骤 5——建立 Asterisk 服务器或是 Cisco 网关

我们现在认为 Asterisk 网关的 PSTN 方面已经配置好了。让我们来改变我们网关的 SIP

配置 (sip.conf) 和它的拨号方案 (extensions.conf) 。我们将配置 Asterisk 来处理从 PSTN 来的或是打给 PSTN 的通话。Asterisk 的一些基本知识是需要的。下面是最简单的一个能够让 Asterisk 与 OpenSER 进行交互的配置。请按照你的布局对这个脚本进行调整。

```
|      警告：只允许来自你的 SIP 服务器的 SIP 包传到你的      |
|
|      asterisk 服务器。不允许来自其他地方的 SIP 包。你      |
|
|      可以用 IP Tables 来达到目的，如果你还是有疑问，请      |
|      资讯 Linux 安全专家。                                    |
```

Asterisk 网关 (sip.conf)

```
[general]
```

```
context=sipincoming
```

```
#calls incoming from the SIP proxy to be terminated in the PSTN lines
```

```
[sipproxy]
```

```
#calls incoming from the PSTN to be forwarded to clients behind the
```

```
SIP
```

```
#proxy
```

```
type=peer
```

```
host=10.1.30.22
```

```
Asterisk (extensions.conf)
```

```
[general]
```

```
[globals]
```

```
[sipincoming]
```

```
exten=>_[0-9].,1,Dial(Zap/g1/${EXTEN:1})
```

```
exten=>_[0-9].,2,hangup()
```

```
[sipoutgoing]
```

```
# If you have a digital interface use the lines below
```

```
exten=_[0-9].,1,Answer()
```

```
exten=_[0-9].,2,dial(SIP/\${EXTEN}@sipproxy)
```

```
exten=_[0-9].,3,Hangup()
```

```
#If you have analog FXO interfaces use the lines below.
```

```
exten=s,1,Answer()
```

```
exten=s,2,dial(SIP/${EXTEN}@sipproxy)
```

```
exten=s,3,Hangup()
```

```
Cisco 2601 网关 ( Cisco 2601 Gateway )
```

下面的解释可以帮的上忙，但是要完成这个配置，Cisco 网关的一些知识也是需要的。Cisco 网关的通话路由是由指令拨号点（instruction dial peer）完成的。由拨号点 voice1 和 2 两条 POTS（plain old telephone system）线路的说明可以看到，任何以 9 打头，其他数字跟随的拨号通话都将会被前转到 PSTN 的 1/0 口或 1/1 口。而在拨号点 voice123 voip 线路的说明中，以 1—9 开始，任何数字跟随的拨号通话都将被直接转到 IP 地址为 10.1.3.22 的 SIP 代理上。

```
voice class codec 1
```

```
codec preference 2 g711ulaw
```

```
!
```

```
interface Ethernet0/0
```

```
ip address 10.1.30.38 255.255.0.0
```

```
half-duplex
```

```
!
```

```
ip classless
```

```
ip route 0.0.0.0 0.0.0.0 10.1.0.1
```

```
no ip http server
```

```
ip pim bidir-enable
```

```
!
```

```
voice-port 1/0
```

```
!
```

```
voice-port 1/1
```

```
!
```

```
mgcp profile default
```

```
!
```

! The dial-peer pots commands will handle the calls coming from SIP

!dial-peers. Any call matching 9 followed by any number of digits will

be forwarded to the PSTN with the 9 striped.

```
dial-peer voice 1 pots
```

```
destination-pattern 9T
```

```
port 1/0
```

```
!
```

```
dial-peer voice 2 pots
```

```
destination-pattern 9T
```

```
port 1/1
```

```
!
```

!The dial-peer voip commands will handle the calls coming from the

pots !dial peers (PSTN). You can prefix a number (80 in this example)

and send the DID number ahead.

!

```
dial-peer voice 123 voip
```

```
destination-pattern ....T
```

```
prefix 80
```

```
forward all
```

```
session protocol sipv2
```

```
session target ipv4:10.1.30.22
```

```
dtmf-relay sip-notify
```

步骤 6——测试配置，打出和接受通话

使用 LCR (Using LCR [Least Cost Routes])

上面的配置是好的，如果你只有一些网关，那么路由通常是不用改变的。然而，大多数的 SIP 服务提供者会频繁的改变路由。除此之外，他们中的大多数都有很多的网关并且会链接到更大的 VoIP 服务提供商那里。如果每次路由发生改变，那么修改脚本是比较麻烦的。那么 LCR 模块就有用武之地了。它允许你可以将路由器和网关插入数据库并且可以动态的更改他们以适应你所需求的系统要求。

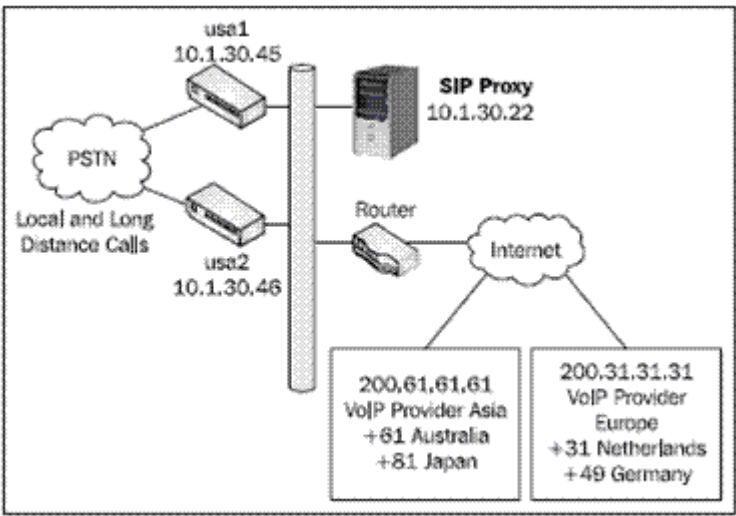
想象一下下面这种情况：你有两个大的服务商，一个在欧洲，一个在亚洲。现在，你想将本地和长途通话送到你自己的网关，到荷兰，德国和法国的通话送到欧洲的服务提供商，到日本和澳大利亚的通话送到亚洲的服务提供商。如果亚洲的服务停了，那么要求能够将通话转向你自己的网关。

LCR 模块 (The LCR Module)

LCR 模块有两种应用功能。最重要的是将一个请求连续的前转到一个或多个网关 (load_gws()和 next_gw())。这些函数用来将通话送到网关上，当然你也可以使用 failure_route 和 next_gw () 将故障转移到其他网关上。你可以给网关分配优先级以决定到底哪一个网关会被优先选择。

你也可以使用 LCR 模块中的 load_contacts()和 next_contacts()函数前转基于 q 值的 contacts。

配置图表（ **Configuration Diagram** ）



要实现上面的这幅图中的内容，我们需要了解三张表：lcr，gw 和 gw_grp。

VoIP 服务提供商拨号方案（ VoIP Provider Dial Plan ）

是时候开始为我们的 VoIP 服务提供商仔细的作一份拨号方案的时候了。我们将实现 E.164 编号方案（ numbering scheme ），将通话送到正确的网关上。当客户拨打本地号码时，我们在使用 load_gw（ ）函数选择网关之前，先使用正则表达式将其号码转换成它们相应的 E.164 地址。除了订阅者，这个服务提供商不会接受其他用户的任何非 E.164 格式的号码。我们假设用户接口能够帮用户拨出 E.164 号码。

Destination	Numbering	Description
VoIP provider callers	"8[0-9][5]"	Six digits starting with 8
Long distance calls	"1[2-9][0-9][9]"	1+Area Code+Subscriber(7 digits)
International Calls	"+[0-9]*"	Any number starting with +
International calls	"011[0-9]*"	Any number starting with 011

LCR 表（ The LCR Table ）

在 lcr 表中，我们将实现路由。lcr 的各个字段描述如下。本地通话和长途通话将以号码 +1305 作为前缀。

Prefix	From_uri	Grp_id	Priority
+31		3	1
+49		3	1
+61		2	1
+81		2	1
+1		1	1

- Prefix——这个前缀要和 URI（phone number）的用户部分（user part）相符合
- From_uri——From_uri 可能会包含需要匹配的 URI。有时候当你想要用主叫用户信息来进行路由的时候使用这个字段。
- Grp_id——Grp_id 标识网关组（gateway group）
- Priority——网关优先级（Gateway priority）

网关表（The Gateways Table）

gw_name	grp_id	ip_addr	port	uri_scheme	transport	strip	prefix
Usa1	1	10.1.30.45	5060	1	1	2	
Usa2	1	10.1.30.46	5060	1	1	2	
Asia1	2	200.61.61.61	5060	1	1	0	
Europe1	3	200.31.31.31	5060	1	1	0	

- gw_name——网关名称（Gateway name）
- grp_id——网关组标识（Gateway group identification）
- ip_addr——网关 IP 地址（IP address of the gateway（对于 1.2.x 一下的版本，使用反转的十进制记法））
- port——（UDP/TCP 端口）
- uri_scheme——sip（1），sips（2）
- transport——udp（1），tcp（2），tls（3）
- strip——删除的字符的数量
- prefix——发往网关前应用的前缀

网关组表（ The Gateway Groups Table ）

网关组表只是用来进行管理之用。与网关的名字息息相关。

添加，删除和显示 LCR 和网关（ Adding, Removing, and Showing LCR and Gateways ）

你可以使用 openserctl 或 SerMyAdmin 来添加，删除，和吸纳是 LCR 表和网关表。你也可以手动进行该项工作。

如果你正使用 OpenSer1.0.x 或 1.1.x，需要观察，这个字段的 IP 地址要是反转的十进制记法。

要将 IP 地址转换成十进制反转记法。需要做如下计算：

- IP Address=a.b.c.d
- 反转十进制记法的 IP 地址 = $d*2^{24}+c*2^{16}+b*2^8+a$

如果你不想计算，那么就使用 openserctl 工具。它对于所有的版本都工作良好。在 OpenSER 的 1.2.x 版本中，你可以将 IP 地址以通常的十进制点记法的格式直接插入数据库。

Openserctl LCR 相关的命令（ Openserctl LCR-Related Commands ）

Command	Description
lcr show	Show routes, gateways, and groups
lcr reload	Reload lcr gateways
lcr addgw_grp <grp_name>	Add gateway group, autocreate grp_id
lcr addgw_grp <grp_name> <grp_id>	Add gateway group with grp_id
lcr rmgw_grp <grp_id>	Delete the gw_grp
lcr addgw <gw_name> <ip> <port> <scheme> <transport> <grp_id>	Add a gateway
lcr addgw <gw_name> <ip> <port> <scheme> <transport> <grp_id> <prefix>	Add a gateway with prefix
lcr addgw <gw_name> <ip> <port> <scheme> <transport> <grp_id> <prefix> <strip>	Add a gateway with prefix and strip
lcr rmgw <gw_name>	Delete a gateway
lcr addroute <prefix> <from> <grp_id> <prio>	Add a route
lcr rmroute <prefix> <from> <grp_id> <prio>	Delete a route

小贴士（Notes）：

- IP 地址必须以点记法的格式键入，如 1.2.3.4
- <uri_scheme> 和 <transport> 必须以整数或文本的方式键入：
 - Transport ‘2’ 等同于 transport ‘tcp’
 - Scheme：1=sip, 2=sips；transport：1=udp, 2=tcp, 3=tls

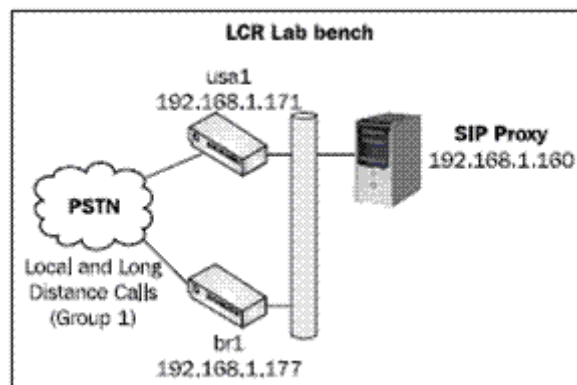
例子（Examples）：

```
lcr addgw_grp usa 1
```

```
lcr addgw_level3 1.2.3.4 5080 sip tcp 1
```

```
lcr addroute +1 '1 1
```

实验——使用 LCR 特性（Lab——using the LCR Feature）



让我们使用 LCR 来执行一个简单的实验。对于这个实验，我们需要一个 OpenSER 服务器，两个网关和一个 IP 电话。你可以使用 asterisk 作为网关和虚拟机很容易的模拟这个实验环境。

步骤 1：使用网关构建 LAB。配置一台名为 usa1 的网关来接收以 +1 作为前缀的通话，另配置一台名为 br1 的网关来接收以 +55 作为前缀的通话。在网关中，你可以在将通话发送到 PSTN 之前添加前缀或删除一个数字。可以使用 strip（）和 prefix（）核心函数来加前缀和删除数字。要了解更多的细节，请登录 www.openser.org。

步骤 2：从 <http://www.sermyadmin.org/openser/openser.lcr> 下载配置文件并复制到

[openser.cfg](#).

```
cd /etc/openser
wget http://www.sermyadmin.org/openser/openser.lcr
cp openser.lcr openser.cfg
The script can be seen below with the modifications highlighted.
# ----- module loading -----
#set module path
mpath="//lib/openser/modules/"
loadmodule "mysql.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "uri.so"
loadmodule "uri_db.so"
loadmodule "domain.so"
loadmodule "permissions.so"
loadmodule "group.so"
loadmodule "mi_fifo.so"
loadmodule "lcr.so"
# Uncomment this if you want digest authentication
# mysql.so must be loaded !
loadmodule "auth.so"
loadmodule "auth_db.so"
# ----- setting module-specific parameters -----
modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
modparam("usrloc", "db_mode", 2)
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
modparam("rr", "enable_full_lr", 1)
modparam("auth_db|permissions|uri_db|
usrloc", "db_url", "mysql://openser:openserrw@localhost/openser")
modparam("permissions", "db_mode", 1)
modparam("permissions", "trusted_table", "trusted")
# ----- request routing logic -----
# main routing logic
route{
```

```

#
# -- 1 -- Request Validation
#
if (!mf_process_maxfwd_header("10")) {
    sl_send_reply("483", "Too Many Hops");
    exit;
};
if (msg:len >= 2048 ) {
    sl_send_reply("513", "Message too big");
    exit;
};
#
# -- 2 -- Routing Preprocessing
#
## Record-route all except Register
if (!method=="REGISTER") record_route();
##Loose_route packets
if (loose_route()) {
    # marca a logica de roteamento no pedido
    append_hf("P-hint: roteado por loose_route\r\n");
    route(1);
};
#
# -- 3 -- Determine Request Target
#
if (method=="REGISTER") {
    route(2);
} else {
    route(3);
};
}
    route[1] {
#
# -- 4 -- Forward request to target
#
## Forward statefully
t_on_failure("1");
if (!t_relay()) {
    sl_reply_error();
};
exit;

```

```

}
route[2] {
## Register request handler
if (is_uri_host_local()) {
if (!www_authorize("", "subscriber")) {
www_challenge("", "1");
exit;
};
if (!check_to()) {
sl_send_reply("403", "Forbidden");
exit;
};
    save("location");
exit;
} else if {
sl_send_reply("403", "Forbidden");
};
}

    route[3] {
## INVITE request handler
if (is_from_local()){
# From an internal domain -> check the credentials and the FROM
if(!allow_trusted()){
if (!proxy_authorize("", "subscriber")) {
proxy_challenge("", "1");
exit;
} else if (!check_from()) {
sl_send_reply("403", "Forbidden, use From=ID");
exit;
};
} else {
log("Request bypassed the auth.using allow_trusted");
};
consume_credentials();
#Verify aliases, if found replace R-URI.
lookup("aliases");
if (is_uri_host_local()) {
# -- Inbound to Inbound
route(10);
} else {
# -- Inbound to outbound

```

```

route(11);
};
} else {
#From an external domain ->do not check credentials
#Verify aliases, if found replace R-URI.
lookup("aliases");
    if (is_uri_host_local()) {
#-- Outbound to inbound
route(12);
    } else {
# -- Outbound to outbound
route(13);
    };
};
}

    route[4] {
# routing to the public network
if (!load_gws()) {
sl_send_reply("503", "Unable to load gateways");
exit;
}

    if(!next_gw()){
sl_send_reply("503", "Unable to find a gateway");
exit;
}
}
route(1);
exit;
}

    route[10] {
#from an internal domain -> inbound
#Native SIP destinations are handled using the location table
#Gateway destinations are handled by regular expressions
#In our example we will normalize the number to e164 +1305XXXXXX
#to facilitate the posterior billing.
append_hf("P-hint: inbound->inbound \r\n");
if (uri=~"^sip:[2-9][0-9]{6}@") {
if (is_user_in("credentials","local")) {
# Assuming your country is USA (+1) and area code (305)
prefix("+1305");
route(4);
exit;

```

```

} else {
sl_send_reply("403", "No permissions for local calls");
exit;
};
};
if (uri=~"^sip:1[2-9][0-9]{9}@") {
if (is_user_in("credentials","ld")) {
prefix("+");
route(4);
exit;
} else {
sl_send_reply("403", "No permissions for long distance");
exit;
};
};
if (uri=~"^sip:011[0-9]*@") {
if (is_user_in("credentials","int")) {
strip(2);
prefix("+");
route(4);
exit;
} else {
sl_send_reply("403", "No permissions for international calls");
};
};
if (!lookup("location")) {
sl_send_reply("404", "Not Found");
exit;
};
route(1);
}
route[11] {
# from an internal domain -> outbound
# Simply route the call outbound using DNS search
append_hf("P-hint: inbound->outbound \r\n");
route(1);
}
route[12] {
# From an external domain -> inbound
# Verify aliases, if found replace R-URI.
lookup("aliases");

```

```

if (!lookup("location")) {
    sl_send_reply("404", "Not Found");
    exit;
};
route(1);
}
route[13] {
    #From an external domain outbound
    #we are not accepting these calls
    append_hf("P-hint: outbound->inbound \r\n");
    sl_send_reply("403", "Forbidden");
    exit;
}
failure_route[1] {
    if(!next_gw()) {
        t_reply("503", "Service not available, no more gateways");
        exit;
    }
    t_on_failure("1");
    t_relay();
}

```

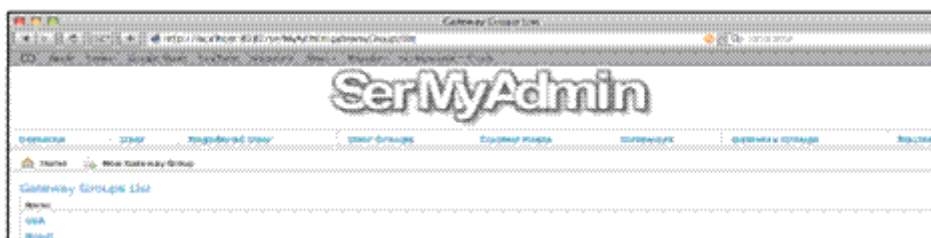
步骤 3：使用 ngrep 抓包并确保包都能够到达正确的目的地。

步骤 4：按照下面的表，使用 `openserctl lcr` 命令添加路由和网关

Icr 网关组（Icr Gateway Groups）

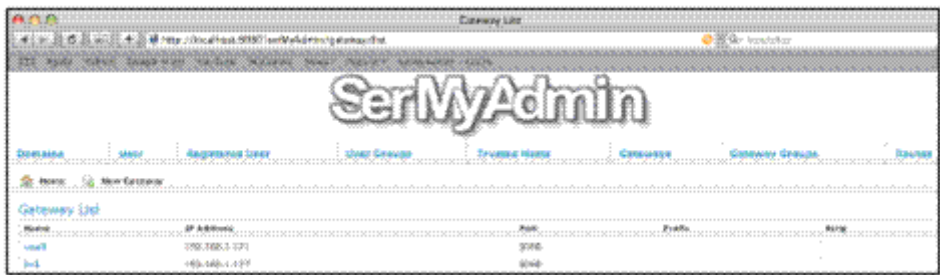
grp_id	grp_name
1	Usa
2	Br

你可以使用 SerMyAdmin 来插入网关组。



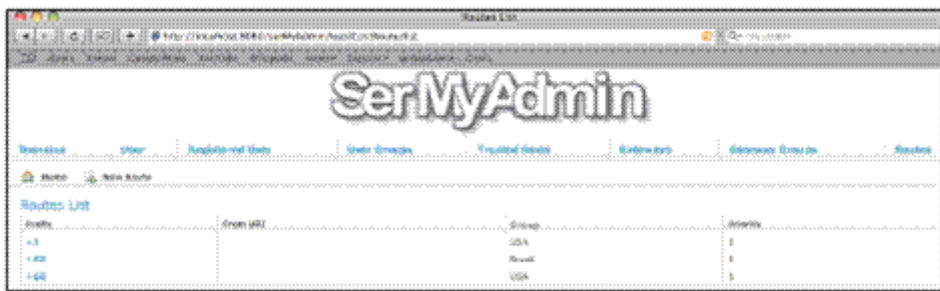
Lcr 网关（lcr Gateways）

gw_name	Ip	port	uri_scheme	transport	grp_id	strip	Prefix
usa1	192.168.1.171	5060	1	1	1	0	
br1	192.168.1.177	5060	1	1	2	0	



Lcr 路由（lcr Routes）

Prefix	From_uri	Grp_id	Priority
+1		1	1
+55		2	1
+55		1	2



步骤 5：对任何以+1 或+55 开头的号码进行测试

步骤 6：关闭网关 br1，并在此测试+55 的通话。通话此时应该走另一个可供选择的网关，因为现在的 br1 已经关闭了。

保证 re-INVITES 的安全（Securing re-INVITES）

既然我们连上了 PSTN，那么对于安全性的考虑就很重要了。Re-INVITES 在松散路由区段中被处理。这些 re-INVITES 不会被要求他们的凭据。为了增强安全性，需要添加下面脚本的内容到你的 loose_route 区段中。如果是接续请求（has_totag()），需要其拥有 ROUTE 头。如果它没有该头（由函数 loose_route() 检查得到），我们将丢弃请求并给出错误“404, not here”。如果你有疑问，请查看文件 opener. 章节 7-3。

```
if (has_totag()) {
    # sequential request withing a dialog should
    # take the path determined by record-routing
    if (loose_route()) {
        #Check authentication of re-invites
        if (method=="INVITE" && (!allow_trusted())) {
            if (!proxy_authorize("", "subscriber")) {
                proxy_challenge("", "1");
                exit;

            } else if (!check_from()) {

                sl_send_reply("403", "Forbidden, use From=ID");
                exit;
            };
        };
        route(1);
    } else {
        sl_send_reply("404", "Not here");
    }
    exit;
}
```

黑名单和“473/Filtered Destination”消息 (Blacklists and “473/Filtered Destination messages”)

DNS 黑名单是用来进行 DNS 故障转移的特性。如果你发送一通电话给一个网关，并且这个网关不能被访问或是返回 5××或 6××错误码，OpenSER 使用了一个叫做“dns blacklist”的资源，并将你的网关插入黑名单。在能够继续发送前，你的网关在此黑名单将持续 4 分钟（这个数字可以在编译时在 blacklists.h 中被修改）。当网关在黑名单中时，如果你试图继续向该网关发送通话，那么你将收到“473/Filtered Destination”消息。要关闭该特性（默认是打开的），可以使用下面的语句：

```
disable_dns_blacklist=yes
```

你也可以创建你自己的永久的网关黑名单或暂时的服务之外的名单。

概要（ **Summary** ）

在本章中，你已经学会如何配置 OpenSER 来将通话前转到网关。注意安全性是很重要的。使用 permissions 模块你能够允许网关避开摘要认证（digest authentication）并允许他们只验证 IP 地址。组模块对于控制由 UAC 到来的访问尤其重要。确认 re-INVITES 的凭据也是很有趣的工作。从你连上 PSTN 的那一刻起，就要特别注意通行欺骗（toll fraud）。我建议你找一个安全专家定期检查你的环境。还要经常的对你的通话记录进行分析以检查出不正常的通话活动。

第八章：通话前转和语音邮件（**call forward and voice mail**）

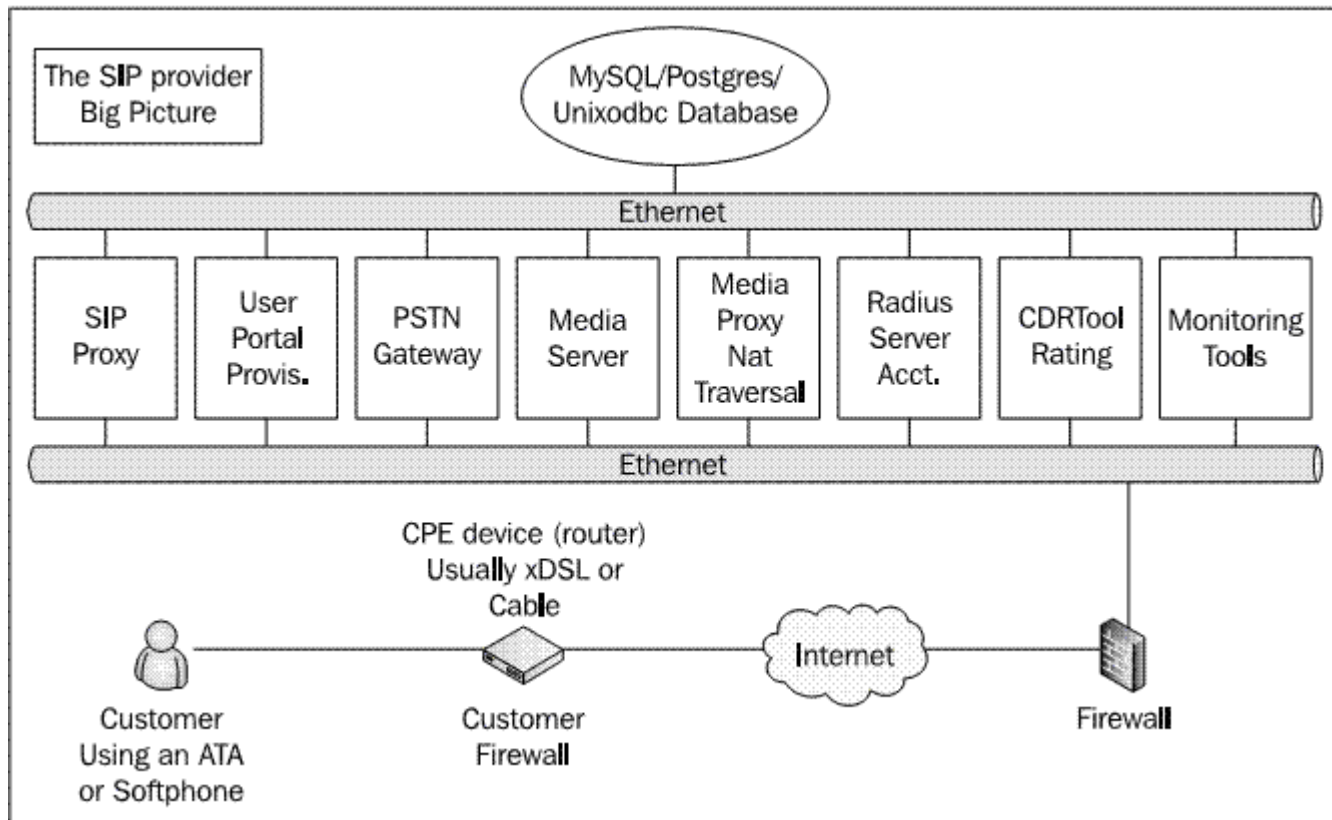
通话前转是 VoIP 服务提供商提供的重要的特性。它的实现有两种方式，一种是派生（forking），一种是重定向。当使用派生时，新的通话分支将会被创建，在首选的目的地失败时（busy 或 timeout），它会向新的目的地发送新的 INVITE 消息。而使用重定向，代理只是会向发起者发送应答信息，并将新的重定向的通话地址告诉发起者。

本章结束你将能够：

- 描述诸如派生和重定向的概念
- 实现通话前转
- 实现忙线时的通话前转
- 使用 AVP 资源存储通话前转数据
- 使用失败路由块（failure route）前转没有应答和忙线的通话

在本材料中，我们只适用派生，因为它比重定向更加安全，而且还能够允许我们对通话进行计费。重定向的方法对于 VoIP 服务提供者来说几乎是无用的，因为代理不在通话的路径当中，在这种情况下根本无法计费。

让我们来检验一下我们的进度。VoIP 服务提供商的解决方案中有许多部件。为了避免失去对整体的把握，我们在每一章中都会展示下面的这幅图。在这一章中，我们将同媒体服务器合作（采用 Asterisk Voice Mail）。还有许多其他的媒体服务器可供选择，如 SEMS（SIP Express Media Server www.ipstel.org/sems），Yate，FreeSwitch 等等。我们之所以选择 Asterisk 是因为它非常流行。“前转”对于在某些情况下将通话送到媒体服务器是非常重要的特性。本章我们会介绍语音邮件（voicemail）的使用。媒体服务器还可以被很多应用使用，如：IVRs，播放提示信息，播放语音文本，和声音识别等。请记住，SIP 代理从不会去处理媒体，所以在需要对媒体进行的处理的情况下，你需要媒体服务器。



本章学习完后，VoIP 服务提供商将能够使用连续派生将无应答和忙线的通话发送到语音邮箱服务器上。

通话前转（**Call Forwarding**）

本章中，我们将实现三种通话前转。此前转对于语音邮箱的操作很重要。

- 盲转（Blind call forwarding）——所有打给某一电话号码的通话的 INVITE 请求都会被立即重新指向存储在 user_preferences 表中的电话号码。SIP 路由器将派生出一路指向新地址的通话。配置了前转的话机无论是否注册上，都不会振铃。
- 忙线前转（Forward on busy）——在这种情况下，我们将使用 failure_route 特性来拦截“486 Busy”消息，并创建一个新的通话将 INVITE 请求发送到最终的目的地。
- 无应答前转（Forward on no answer）——如果话机对于 INVITE 请求的回复是“408 Request Timeout”，那么 OpenSER 也会使用 failure_route 特性来拦截该消息并创建新的通话将 INVITE 消息发送到最终的目的地。

所有的呼叫前转的目的地址都被存储在叫做 user_preferences 的表中。我们在这一章将介绍如 AVPs（Attribute-Value Pairs）和伪变量（pseudo-variables）的一些新的概念。AVPs

是 OpenSER 的核心使用。AVPOPS (Attribute-Value Pairs Operations) 模块提供了几个操作 AVPs (如, 同 SIP 请求和数据库进行交互, 使用字符串和正则表达式进行操作。) 的函数。

伪变量 (**Pseudo-Variables**)

伪变量是系统变量, 你可以在你的脚本中将其作为参数或内部函数 (inside functions) 使用。在脚本执行之前, 这些变量会被相应的值替换掉。有些模块可以接受伪变量, 如:

- ACC
- AVPOPS
- TEXTOPS
- UAC
- XLOG

伪变量的名字总是以“\$”开头。如果你想要在你的脚本中使用\$字符, 那么你必须像这样进行换码——\$\$. 有一个伪变量的预定义集合。OpenSER1.1 中使用的 OpenSER 伪变量如下:

\$ar	Auth realm
\$au	Auth username
\$br	Request's first branch
\$bR	All Request's branches
\$ci	Call-ID header
\$cl	Content length
\$cs	Cseq
\$ct	Contact Header
\$cT	Content Type
\$dd	Domain of destination URI
\$di	Diversion header URI
\$dp	Port of destination URI
\$dP	Transport protocol of destination URI
\$ds	Destination set

\$du	Destination URI
\$fd	From URI domain
\$fn	From display name
\$ft	From Tag
\$fu	From URI
\$fU	From URI username
\$mb	SIP message buffer
\$mf	Flags
\$mF	Flags in hexadecimal
\$mi	SIP message ID
\$ml	SIP message length
\$od	Domain in SIP request's original URI
\$op	Port of Sip request's original URI
\$oP	Transport protocol of SIP request's original URI
\$ou	Request's original URI
\$oU	Username in SIP request's original URI
\$pp	Process id
\$rd	Domain in SIP request's URI
\$rb	Body of request/reply
\$rc	Returned code
\$rm	SIP request's method
\$rp	SIP request's port of R-URI
\$rP	Transport protocol of SIP request URI
\$rr	SIP reply reason
\$rs	SIP reply status
\$rt	Refer-to URI
\$ru	SIP request's URI
\$rU	Username in SIP request's URI
\$Ri	Received IP address
\$Rp	Received Port
\$si	IP source address
\$sp	Source port
\$td	To URI domain
\$tn	To display name
\$tt	To tag
\$tu	To URI

\$tU	To URI username
\$Tf	String formatted time
\$Ts	Unix time stamp
\$ua	User agent header
\$re	Remote-Party-ID header URI

AVP (Attribute-Value Pair) 概览 (AVP Overview)

属性-值对的操作相当于是允许了对用户的首选项 (user preferences) 进行访问和操作 。 AVP 可以看作是与标识 (字符串或整数) 相关联的一个值。在 OpenSER 的处理过程中, AVP 与事务捆绑在一起。当事务开始时, AVP 被分配, 当其结束时, 则被释放。

AVPs 的出现创造了一些服务实现和用户或域名的用户首选项处理的新的可能性。它们可以在配置脚本中被直接使用并从 MySQL 数据库中加载数据。

属性-值对的引用与变量的引用非常相似。

`$avp(id[N])`

Where ID is:

- `si : name` —— AVP 标识名称。“s”和“i”分别表示字符串和整数。
- `name` —— 别名 AVP 的名称。可以是字符串, 也可以是整数。

例子:

`$avp (i: 700)`

`$avp (s: blacklist)`

对于了解 Asterisk 的人来说, AVPOPS 模块之于 OpenSER 就相当于 AstDB 函数之于 Asterisk。然而, 实现方式非常不同, AVPs 更加强大, 允许一些更加高级的特性, 如数据库的查询和直接将数据插入 SIP 包等。有许多与 AVPs 相联系的函数如下:

- `avp_db_load`: 将 AVPs 从数据库加载至内存
- `avp_db_store`: 将 AVPs 存进数据库
- `avp_db_delete`: 从数据库中删除 AVPs
- `avp_db_query`: 进行数据库查询并将结果存进 AVP 中

- avp_delete: 从内存中删除 AVPs
- avp_push: 将 AVP 的值插入 sip 消息
- avp_check: 使用一个操作符和一个值来检查 AVP 的值
- avp_copy: 拷贝 AVP 到另一个
- avp_printf: 格式化一个字符串到 AVP
- avp_subst: 查找并替换一个值到 AVP
- avp_op: 允许在 AVPs 上进行算术操作
- is_avp_set: 检查这个 AVP 名字是否被设置
- avp_print: 打印内存中的所有 AVPs (为了 debug)

由于文档中的这些函数，你可以对句法进行检查。现在，我们必须要了解如何使用 avp_db_load 和 avp_push，它们将用在我们的脚本中。关于 AVPs 在 <http://www.voice-system.ro/docs> 上有一篇非常棒的教程。

AVPs 不是很简单。但是如果你把它想象成一对简单的属性和值，那么它们其实也不是那么的复杂。然而，数据库中的 AVPs 的加载是非常令人费解的。默认的表是 usr_preference。有时候我们想要的值并不是和一个特定的用户相关，而是同一个域相关。不管怎样，所有从数据库中被加载的 AVPs 都来自 usr_preference 表。

举例：对于呼叫前转，我们有一个与用户相关的前转呼叫。确实是一个 usr_preference (用户优先)。让我们来查看 usr_preference 表的结构。

id	uuid	username	domain	attribute	type	value	Last_modified
	1001			callfwd	0	sip:1004@yourdomain	

Id 是一个自增域。

- uuid 是一个唯一的用户标识
- username 是用户名
- domain 代表域
- attribute (AVP 名称)
- type (0-AVP str|Val Str, 1-AVP str|Val Int, 2-AVP int|Val Str, 3-AVP int|Val int)

- value (AVP 值)
- last modified (最后修改的日期)

AVPs 可以与一个用户相关联也可以与一个域相关联。所以，你能够使用这两个参数中的任何一个来加载 AVPs。你可以将 AVP 与 uuid (唯一的用户 ID) 相关联，与一个用户名相关联 (单域设置)，或者与用户名和域建立多域设置关系。

函数 `avp_db_load` 的第一个参数是源 (source)，第二个是 `avp_name`。所以，下面的函数将以字符串的形式加载与被请求用户名中的 URI 相符合的用户的 AVP 名称为 `callfwd` 的 AVP 值。

```
(avp_db_load("$ruri/username","s:callfwd")
```

之后，我们将把这个 AVP 插进 SIP 包中，以将原来的 `$ruri` 进行改造。

```
avp_push("s:callfwd");
```

换句话说，如果这个用户设置了对应的呼叫前转号码，那么这通电话将不会呼给原来的用户，而是呼给存储在 AVP `s:callfwd` 中的用户。呼叫前转的特别之处就在于将呼叫前转号码插入了 `usr_preference` 表中。

AVPOPS 模块假造和参数 (AVPOPS Module Loading and Parameters)

在模块加载中，我们指定数据库地址，访问参数和 AVP 表。

```
loadmodule "/usr/lib/openser/modules/avpops.so"
```

```
modparam("avpops", "avp_url", "mysql://openser:openserrw@localhost/openser")
```

```
modparam("avpops", "avp_table", "usr_preferences")
```

实现呼叫盲转 (Implementing Blind Call Forwarding)

首先，让我们来实现呼叫盲转服务。在 INVITE 请求的处理过程中，我们将从数据库中的用户优先表中加载名称为 `callfwd` 的 AVP。如果 `callfwd` 被设置给了指定的用户，那么我们在对该请求进行前转前将它“推进” (push) 到 R-URI 中。

```
if(avp_db_load("$ruri/username","s:callfwd")){
```

```
#Check the existence of the callfwd attribute on the

#usr_preferences table. If found, assign the vaue to the AVP

# and push the value to the ruri of the SIP header.

avp_push("ruri","s:callfwd");

route(1);

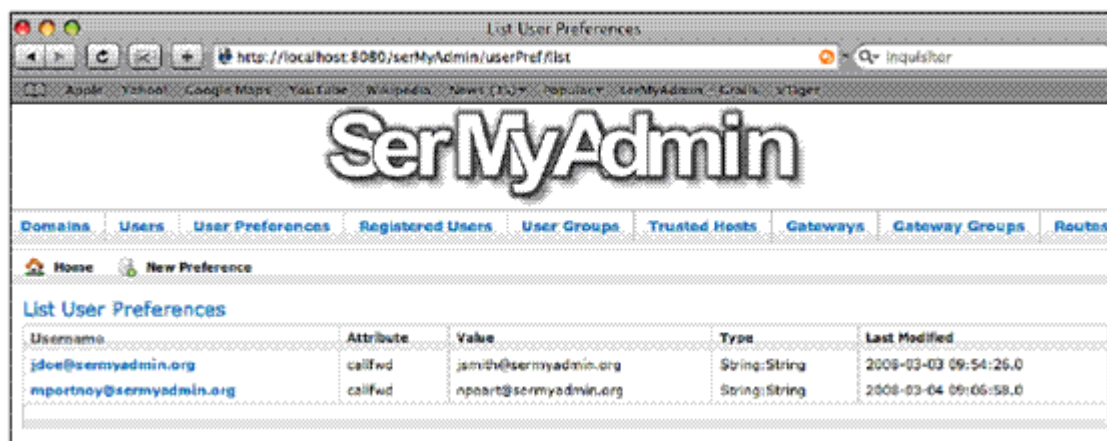
exit;

};
```

为了能让这个特性得以工作，重点在于在数据库中插入正确的入口点。AVPs 使用的表叫做 `usr_preferences`。

Username	Type	Attribute	Value
1001	0	callfwd	sip:1004@yourdomain

你可以在 SerMyAdmin 的帮助下来对用户的喜好进行修改；只需要浏览菜单中“user preferences”即可。在那个选单中你可以查看所有的用户喜好，编辑它们，还可以创建新的。



如果你工作在多域环境中，请将模块 AVPOPS 的多域参数功能打开，并用域名来填充数据库。

在上面的记录中，我们告诉系统将任何呼给 1001 的呼叫都呼给 URI：`sip:1004@yourdomain`。

Lab——实现呼叫盲转（ Lab——Implementing Blind Call Forwarding ）

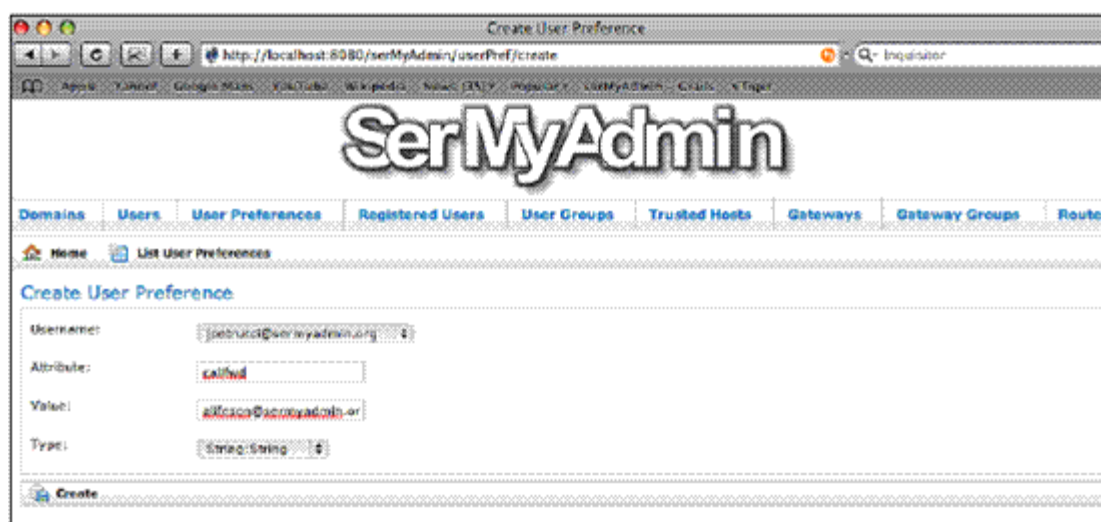
步骤 1：让我们使用在第六章里第一次见到的 SerMyAdmin 接口来插入 AVP 对。

你的浏览器中访问 SerWEB 的管理接口为：

`http://<your-ip-server-address>:8080/serMyAdmin`

步骤 2：使用一个具有“全局管理员”角色的用户登录接口。

步骤 3：点击用户喜好（ User preferences ）标签。在这个菜单中，点击“New Preference”按钮，为你想要前转通话的用户来创建 AVP；在这个例子中，应该是 1000@sermyadmin.org。它的属性比较取名为 callfwd，值是你前转通话的 URI；这里应该设置成 1004@sermyadmin.org。



步骤 4：编辑 openser.cfg 以包含上面做的那些说明解释。文件应该以下面的代码结束。在 openser.cfg 文件中包含下面的代码行。或是简单地从 <http://www.sermayadmin.org/openser> 拷贝 openser.callfwd1 到 openser.cfg。

在模块加载部分：

```
loadmodule "avpops.so"
```

```
loadmodule "xlog.so"
```

在模块参数部分：

```
modparam("avpops", "avp_url", "mysql://openser:openserrw@localhost/openser")
```

```
modparam("avpops", "avp_table", "usr_preferences")
```

在 route[3]部分：

```
route[3] {
```

```
#
```

```
# -- INVITE request handler --
```

```
#
```

```
if (is_from_local()) {
```

```
# From an internal domain -> check the credentials and the FROM
```

```
if (!allow_trusted()) {
```

```
    if (!proxy_authorize("", "subscriber")) {
```

```
        proxy_challenge("", "1");
```

```
        exit;
```

```
    } else if (!check_from()) {
```

```
        sl_send_reply("403", "Forbidden, use From=ID");
```

```
        exit;
```

```
    };
```

```
} else {
```

```
    log("Request bypassed the auth. using allow_trusted");
```

```
};
```

```
if (avp_db_load("$ru/username", "$avp(s:callfwd)") {
```

```
    avp_push_to("$ru", "$avp(s:callfwd)");
```

```

xlog("forwarded to: $avp(s:callfwd)");

route(1);

exit;

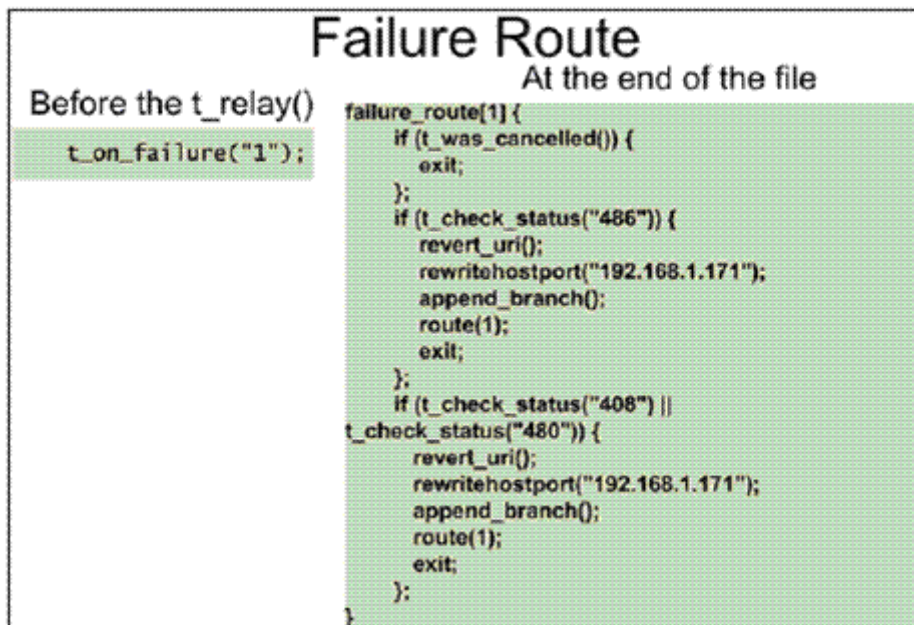
}

consume_credentials();

```

步骤 5：注册分机 1001 和 1004。使用 1001 打给 1000，这时候这通电话就应该按照 `usr_preferences` 表中说明被前转到 1004。

在忙线或无人接听的时候实现呼叫前转（Implementing Call Forward on Busy or Unanswered）



这是这一章的第二部分。现在我们将介绍两个新的重要概念。第一个是 `failure_route`，第二个则是 `append_branch`，其被用来派生通话（fork the call）。我们将处理下面的出错情况：

- 408 – Timeout
- 486 – Busy Here
- 487 – Request Cancelled

为了实现在忙线和无人接听的情况下的通话前转，我们将使用出错路由（`failure route`）的概念。

下面的逻辑中，在发送 INVITE 到标准处理过程前，我们将调用函数 `t_on_failure ("1")`。这允许我们在 `failure_route[1]` 中处理 SIP 出错消息（返回的消息高于 299 的，也叫做负应答 `negative replies`）

当在这种情况下收到一通通话时，我们将其前转到一个语音邮箱系统。Asterisk 能够充当一个相当好的语音邮箱系统。让我们来看看如何整合 asterisk 来记录语音邮箱消息。我们将在 URI 使用 `b (busy)` 前缀来告诉 asterisk 服务器播放 busy 消息，使用 `u (unanswered)` 来播放无应答消息。Asterisk 将分别使用应用 `voicemail(b${EXTEN})` 和 `voicemail(u${EXTEN})` 来处理忙线和无人接听的语音邮箱请求。

下面是一个完整的脚本，修改部分使用高亮显示。

```
#
#
# $Id: opener.cfg 1676 2007-02-21 13:16:34Z bogdan_iancu $
#
# simple quick-start config script
# Please refer to the Core CookBook at http://www.openser.org/dokuwiki/doku.php
# for a explanation of possible statements, functions and parameters.
#
# ----- global configuration parameters -----
debug=3 # debug level (cmd line: -ddddddddd)
fork=yes
log_stderr=no # (cmd line: -E)
children=4
port=5060

# ----- module loading -----

#set module path
mpath="//lib/openser/modules/"
# Uncomment this if you want to use SQL database
#loadmodule "mysql.so"
loadmodule "mysql.so"
loadmodule "sl.so"
loadmodule "tm.so"
loadmodule "rr.so"
loadmodule "maxfwd.so"
loadmodule "usrloc.so"
loadmodule "registrar.so"
loadmodule "textops.so"
loadmodule "uri.so"
```

```

loadmodule "uri_db.so"
loadmodule "domain.so"
loadmodule "permissions.so"
loadmodule "group.so"
loadmodule "mi_fifo.so"
loadmodule "lcr.so"
loadmodule "avpops.so"
loadmodule "xlog.so"
# Uncomment this if you want digest authentication
# mysql.so must be loaded !
loadmodule "auth.so"
loadmodule "auth_db.so"
# ----- setting module-specific parameters -----
modparam("mi_fifo", "fifo_name", "/tmp/openser_fifo")
modparam("usrloc", "db_mode", 2)
modparam("auth_db", "calculate_ha1", yes)
modparam("auth_db", "password_column", "password")
modparam("rr", "enable_full_lr", 1)
modparam("auth_db|permissions|uri_db|
usrloc", "db_url", "mysql://openser:openserrw@localhost/openser")
modparam("permissions", "db_mode", 1)
modparam("permissions", "trusted_table", "trusted")
modparam("avpops", "avp_url", "mysql://openser:openserrw@localhost/openser")

modparam("avpops", "avp_table", "usr_preferences")

# ----- request routing logic -----
# main routing logic
route{
#
# -- 1 -- Request Validation
#
if (!mf_process_maxfwd_header("10")) {
sl_send_reply("483", "Too Many Hops");
exit;
};
if (msg:len >= 2048 ) {
sl_send_reply("513", "Message too big");
exit;
};
#

```

```

# -- 2 -- Routing Preprocessing
#
## Record-route all except Register
if (!method=="REGISTER") record_route();
##Loose_route packets
if (has_totag()) {
#sequential request withing a dialog should
# take the path determined by record-routing
if (loose_route()) {
#Check authentication of re-invites
if(method=="INVITE" && (!allow_trusted())) {
if (!proxy_authorize("", "subscriber")) {
proxy_challenge("", "1");
exit;
} else if (!check_from()) {
sl_send_reply("403", "Forbidden, use From=ID");
exit;
};
};
route(1);
} else {
sl_send_reply("404", "Not here");

}

exit;
}
#CANCEL processing
if (is_method("CANCEL")) {
if (t_check_trans()) t_relay();
exit;
};
t_check_trans();
#
# -- 3 -- Determine Request Target
#
if (method=="REGISTER") {
route(2);
} else {
route(3);
};

```



```

}
    route[1] {
#
# -- 4 -- Forward request to target
#
## Forward statefully
t_on_failure("1");
if (!t_relay()) {
    sl_reply_error();
};
exit;
}
    route[2] {
## Register request handler
if (is_uri_host_local()) {
    if (!www_authorize("", "subscriber")) {
        www_challenge("", "1");
        exit;
    };
    if (!check_to()) {

        sl_send_reply("401", "Unauthorized");

        exit;
    };
    save("location");
    exit;
} else if {
    sl_send_reply("401", "Unauthorized");
};

}

    route[3] {
## Non-Register request handler
if (is_from_local()){
# From an internal domain -> check the credentials and FROM
if(!allow_trusted()){
    if (!proxy_authorize("", "subscriber")) {
        proxy_challenge("", "1");
        exit;
    } else if (!check_from()) {

```

```

sl_send_reply("403", "Forbidden, use From=ID");
exit;
};
} else {
log("Request bypassed the auth.using allow_trusted");

};

if(avp_db_load("$ru/username","$avp(s:callfwd)") {
avp_pushto("$ru", "$avp(s:callfwd)");
route(1);
exit;
}
consume_credentials();
#Verify aliases, if found replace R-URI.
lookup("aliases");
if (is_uri_host_local()) {
# -- Inbound to Inbound
route(10);
} else {
# -- Inbound to outbound
route(11);

};

} else {
#From an external domain ->do not check credentials
#Verify aliases, if found replace R-URI.
lookup("aliases");
if (is_uri_host_local()) {
#-- Outbound to inbound
route(12);
} else {
# -- Outbound to outbound
route(13);
};
};

}

route[4] {

```

```

# routing to the public network
if (!load_gws()) {
sl_send_reply("503", "Unable to load gateways");
exit;
}
if(!next_gw()){
sl_send_reply("503", "Unable to find a gateway");
exit;
}
route(5);
exit;

}

route[5] {
#
# -- 4 -- T_relay for gateways
#
## Forward statefully, if failure load other gateways
t_on_failure("2");
if (!t_relay()) {
sl_reply_error();
};
exit;

}

route[10] {
#from an internal domain -> inbound

#Native SIP destinations are handled using the location table

#Gateway destinations are handled by regular expressions
append_hf("P-hint: inbound->inbound \r\n");
if (uri=~"^sip:[2-9][0-9]{6}@") {
if (is_user_in("credentials","local")) {
prefix("+1");
route(4);
exit;
} else {
sl_send_reply("403", "No permissions for local calls");
exit;
}
}
}

```

```

};
};
if (uri=~"^sip:1[2-9][0-9]{9}@") {
if (is_user_in("credentials","ld")) {
strip(1);
prefix("+1");
route(4);
exit;
} else {
sl_send_reply("403", "No permissions for long distance");
exit;
};
};
if (uri=~"^sip:011[0-9]*@") {
if (is_user_in("credentials","int")) {
strip(3);
prefix("+");
route(4);
exit;
} else {
sl_send_reply("403", "No perm. for internat.calls");
};
};
if (!lookup("location")) {
if (does_uri_exist()) {
## User not registered at this time.
## Use the IP Address of your e-mail server
revert_uri();
prefix("u");

rewritehostport("192.168.1.171"); #Use the voicemail IP

route(1);
} else {
sl_send_reply("404", "Not Found");
exit;
}
sl_send_reply("404", "Not Found");
exit;
};
route(1);

```

```

}

route[11] {
# from an internal domain -> outbound
# Simply route the call outbound using DNS search
append_hf("P-hint: inbound->outbound \r\n");
route(1);

}

route[12] {
# From an external domain -> inbound
# Verify aliases, if found replace R-URI.
lookup("aliases");
if (!lookup("location")) {
sl_send_reply("404", "Not Found");
exit;
};
route(1);

}

route[13] {
#From an external domain outbound
#we are not accepting these calls
append_hf("P-hint: outbound->inbound \r\n");
sl_send_reply("403", "Forbidden");
exit;

}

failure_route[1] {
##--
##-- If cancelled, exit.
##--
if (t_check_status("487")) {
exit;

};

##--
##-- If busy send to the e-mail server, prefix the "b"
##-- character to indicate busy.

```

```

##--
if (t_check_status("486")) {
    revert_uri();
    prefix("b");
    xlog("L_ERR","Stepped into the 486 ruri=<$ru>");
    rewritehostport("192.168.1.171");
    append_branch();
    route(1);
    exit;
};

##--
##-- If timeout (408) or unavailable temporarily (480),
##-- prefix the uri with the "u" character to indicate
##-- unanswered and send to the e-mail
##-- sever
##--
if (t_check_status("408") || t_check_status("480")) {
    revert_uri();
    prefix("u");
    xlog("L_ERR","Stepped into the 480 ruri=<$ru>");
    rewritehostport("192.168.1.171");
    append_branch();
    route(1);
    exit;
};

}

failure_route[2] {
    if(!next_gw()) {
        t_reply("503", "Service not available, no more gateways");
        exit;
    }
    t_on_failure("1");
    t_relay();
}

```