SIPp 3.4中文参考文档

翻译人员：**李明禄**

日期:2014.4.23

SIP VoIP 测试交流群：323827101

SIPp参考文档（http://sipp.sourceforge.net/doc/reference.html）

由Richard GAYRAUD[初始代码]，Olivier JACQUES[代码/文档]，Robert Day[[代码/文档]，Charles P. Wright[代码]，许多贡献者[代码]

目录列表

1. Foreword（前言）

 警告：

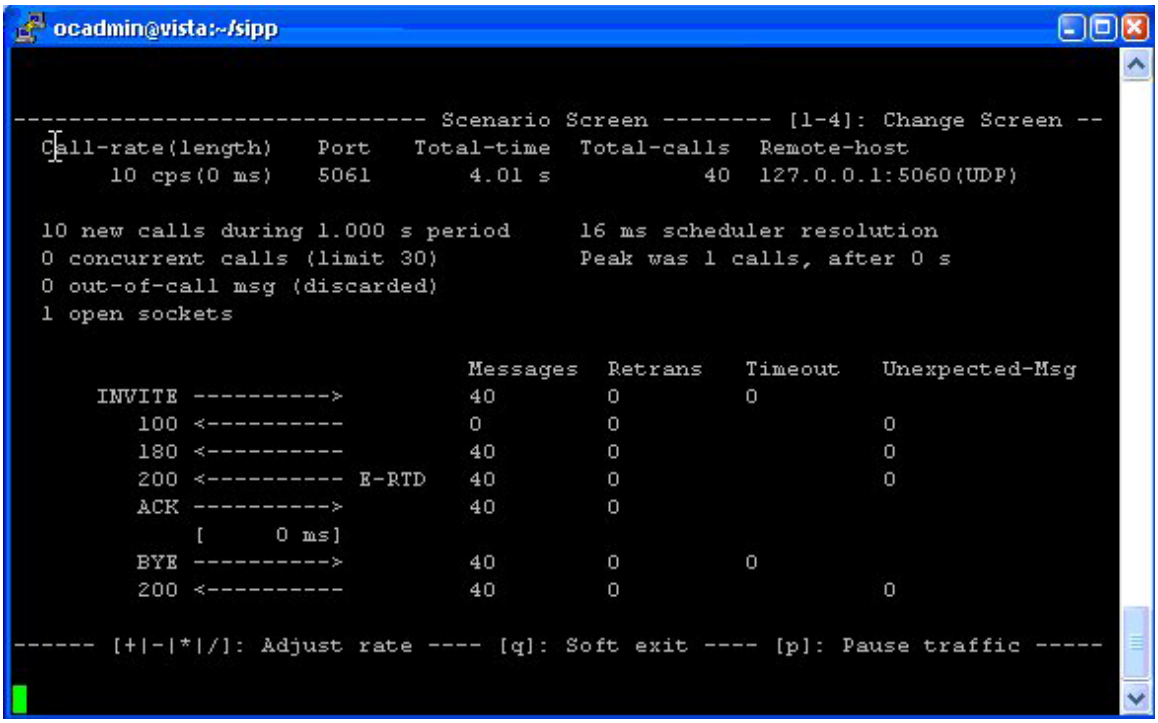
这个版本的文档是为 SIPp 3.4 而写的，描述了一些以前版本不存在的特性。

SIPp是一个性能测试工具,用于SIP协议。它包括几个基本SipStone用户代理场景(UAC和UAS)并且可以用个INVITE和BYE建立和释放多个呼叫。它还可以读取XML场景文件描述任何性能测试配置文件。它具有动态显示运行测试的统计数据(称为呼叫速率、往返延迟以及消息统计)，在场景文件中周期性统计CSV数据、TCP和UDP多个socket或传输管理中多路复用、正则表达式和变量和动态可调呼叫速率。

SIPp可以用来测试许多真正的SIP设备像SIP代理、B2B用户代理、SIP媒体服务器、SIP／x网关、SIP专用小交换机,……它也通常用于模仿成千上万的用户代理呼叫你的SIP系统。

想要了解它吗?

这里有一个SIPp页面截图



这是SIPp执行的过程中的一个视频(Windows媒体播放器9编解码器或以上要求):

sipp-01.wmv

2. Installation (安装)

2.1. 获取 SIPp

SIPp在GNU GPL license(http://www.gnu.org/copyleft/gpl.html)下发布。license应用于所有条款。最初由惠普(http://www.hp.com)工程师提供给SIP社区。

现在惠普公司只接收一些有关这个自由工具信息,但惠普并不提供有关支持SIPp的任何帮助。

2.2. Stable release（发布版本）

像许多其他的“开源”项目一样, SIPp有两个版本: stable(稳定版本)和unstable release（不稳定版本）。Stable release（稳定发布版）:在被标记“stable”以前,会进行一次彻底测试。所以上面提到的所有功能都会正常运行)

> **注意:**
>
> 使用stable release为您的日常使用,如果你不是被一个特定的功能在当下的“unstable release”(见下文)。

SIPp stable release的下载页面 (http://sourceforge.net/project/showfiles.php?group_id=104305)

2.3. Unstable release（不稳定的发布版本）

Unstable release: 尽可能快地修复 SIP SVN(http://sipp.svn.sourceforge.net/viewvc/sipp/sipp/trunk/)库中所有新功能和确认的 bug。

> **注意:**
>
> 使用不稳定的版本仅仅是在你需要不稳定版本中存在的功能，建议还是用稳定版本。

2.4. Available platforms（可用平台）

SIPp有效运行在Linux和Cygwin上。在其它Unix分发版本或许正常工作，但并没有全部测试过。

> **注意:**
>
> 因为支持IPv6，只能在Windows XP系统和最新的版本安装Cygwin后运行SIPp，在Windows2000中无法运行。

2.5. Installing SIPp（安装SIPp）

- 在Linux中,SIPp以源代码的形式提供。你需要编译后才能使用。
- 编译SIPp的条件，参见编译技巧 (http://sipp.sourceforge.net/wiki/index.php/Compilation)):

  - C++编译器
  - curses 或ncurses 库
  - 支持TLS：OpenSSL >= 0.9.8
  - 支持pcap播放：libpcap和libnet
  - 支持SCTP：lksctp-tools
  - 对分布式停顿支持: Gnu Scientific库 (http://www.gnu.org/software/gsl/)

    （原文: For distributed pauses: Gnu Scientific Libraries (http://www.gnu.org/software/gsl/)）

- 四个选项编译SIPp：

  - **不支持TLS（传输层安全协议），SCTP或 PCAP 支持**：这是推荐的设置如果你不需要处理SCTP、TLS或PCAP。

```
# tar -xvzf sipp-xxx.tar

# cd sipp

# ./configure

# make
```

  - **支持TLS:你**必须安装OpenSSL库(http://www.openssl.org/)

(>=0.9.8))(你的系统可能已经有了)。只有在命令上添加"--with-openssl"选项来

构造SIPp:

```
# tar -xvzf sipp-xxx.tar.gz
# cd sipp
# ./configure --with-openssl
# make
```

- 支持PCAP播放：

```
# tar -xvzf sipp-xxx.tar.gz
# cd sipp
# ./configure --with-pcap
# make
```

- 支持SCTP ：

```
# tar -xvzf sipp-xxx.tar.gz
# cd sipp
# ./configure --with-sctp
# make
```

- 你也能绑定这些选项，如：

```
# tar -xvzf sipp-xxx.tar.gz

# cd sipp

# ./configure --with-sctp --with-pcap --with-openssl

# make
```

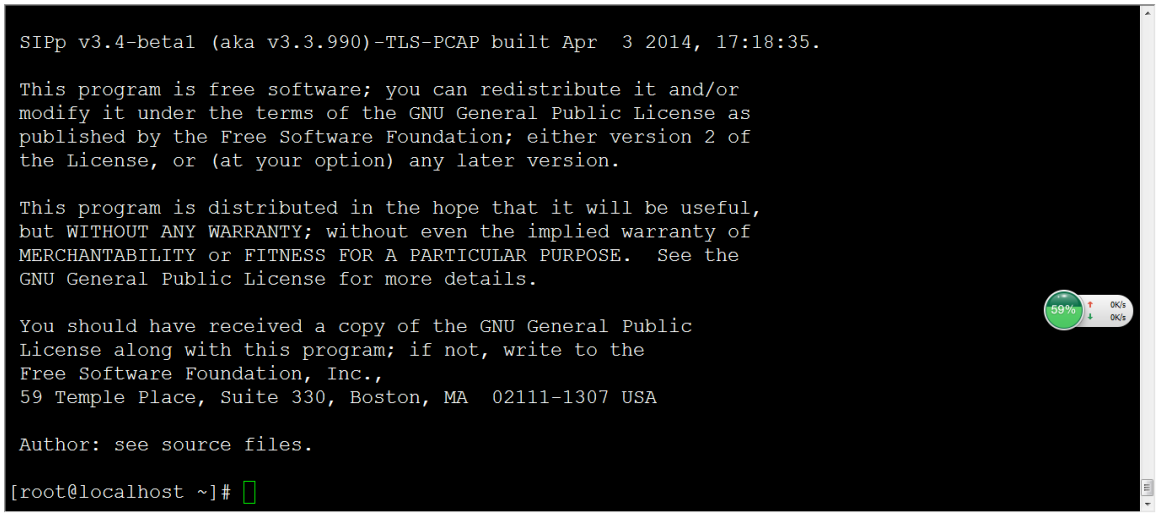
我写了一个shell脚本,将下面的内容拷贝到一个记事本中,然后保存为install.sh后再在linux下执行

```
./install.sh
```

Shell脚本如下：

```
#!/bin/sh

yum install make gcc gcc-c++ ncurses ncurses.x86_64 ncurses-devel ncurses-devel.x86_64 openssl libnet libpcap libpcap-devel libpcap.x86_64 libpcap-devel.x86_64 gsl gsl-devel

cd /root/

wget http://sourceforge.net/projects/sipp/files/sipp/3.4/sipp-3.3.990.tar.gz/download

tar -zxvf sipp-3.3.990.tar.gz

cd sipp-3.3.990/

./configure  --with-pcap --with-openssl

make && make install

sipp -v
```

当出现下面的信息后说明安装完成：



警告：

在Windows XP中CYGWIN下编译SIPp， CYGWIN（http://win6.jp/Cygwin/)提供安装IPv6扩展，和libncurses（可选OpenSSL andWinPcap）一样。当前不支持SCTP。

- 在windows下编译带有pcap（支持媒体流）功能的SIPp源文件，你必须：

■ 复制WinPcap开发者包(http://www.winpcap.org/devel.htm)到“C:\ cygwin \ lib \ WpdPack”下。

■ 在“C:\cygwin\lib\WpdPack\Include”中删除或重命名“pthread.h”,否则在执行SIPp的时候会干扰cygwin运行。

■ 在cygwin中编译的时候一定要用命令"make pcapplay_cygwin"或"pcapplay_ossl_cygwin"

2.6. 增加文件描述符限制

如果您的系统不支持足够的文件描述符,你在使用许多同步呼叫的TCP/TLS模式时可能会遇到问题。

你有两种方法来克服这一限制:要么使用-max_socket命令行选项或者更改你的系统的限制：

根据您使用的操作系统,不同的程序允许你增加最大数量的文件描述符:

■ 在Linux 2.4内核增加默认的文件描述符数量可以修改/etc/security/limits.conf和/etc/pam.d/login这两个文件。

打开/etc/security/limits.conf文件并添加以下两行代码:

```
soft nofile 1024

hard nofile 65535
```

打开/etc/pam.d/login文件添加下列代码：

```
session required /lib/security/pam_limits.so
```

在/proc/sys/fs/file-max文件设置系统的文件描述符限制。下面的命令将增加文件描述符限制:

```
echo 65535> /proc/sys/fs/file-max
```

为了增加文件描述符的数量到最大限度(65535)，在/etc/security/limits.conf文件中输入：

```
ulimit -s unlimited
```

```
[root@localhost ~]# ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 16384
max locked memory       (kbytes, -l) 32
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) unlimited
cpu time               (seconds, -t) unlimited
max user processes              (-u) 16384
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
[root@localhost ~]#
```

注销然后再登录,使改变生效。

■ 在HP-UX系统，可能通过带有sam程序系统配置去增加默认的文件描述符数量。在内核配置菜单,选择可配置参数,更改以下属性:

```
maxfiles : 4096

maxfiles_lim : 4096
```

```
nfiles : 4096

ninode : 4096

max_thread_proc : 4096

nkthread : 4096
```

3. Using SIPp（使用SIPp）

3.1. Main features（主要功能）

SIPp可以生成1个或多个SIP呼叫至一个远程系统，这个工具用命令行启动，在下面的例子里，在前端启动两个SIPp来演示SIPp的功能。

运行SIPp嵌入式服务（uas）场景：

```
# ./sipp -sn uas
```

在同样的主机中，运行sipp内嵌的客户端（uac）场景：

```
# ./sipp -sn uac 127.0.0.1
```

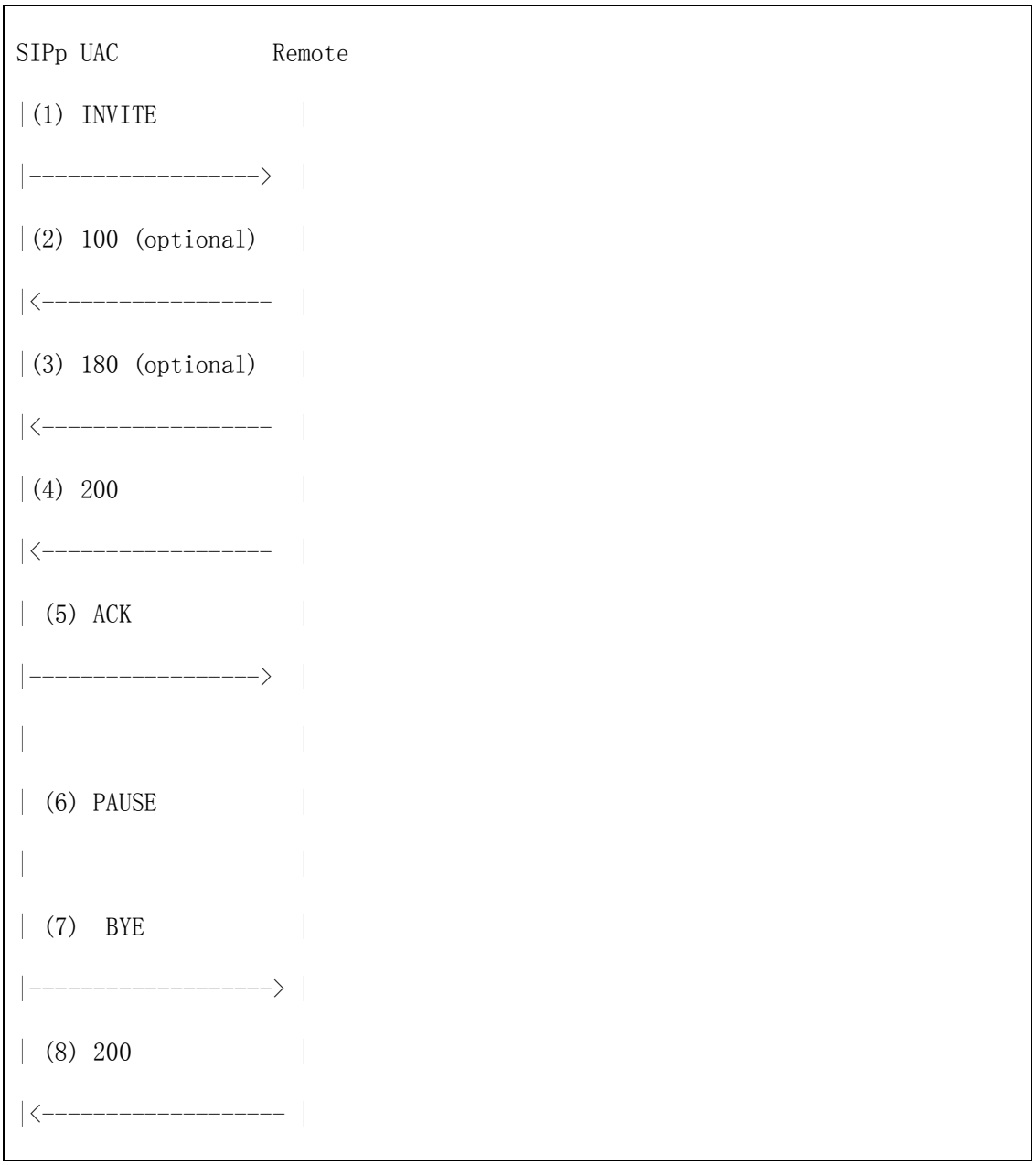
3.2. Integrated scenarios（集成场景）

集成的场景？没错，场景文件被嵌入到了SIPp可执行脚本中，当你创建自己定义的SIPp场景时（参考“how to create yourown XML scenarios（怎样创建自己的XML场景）”），在SIPp中可执行一些基本的(但其中有相当有用的)场景。

3.2.1. UAC

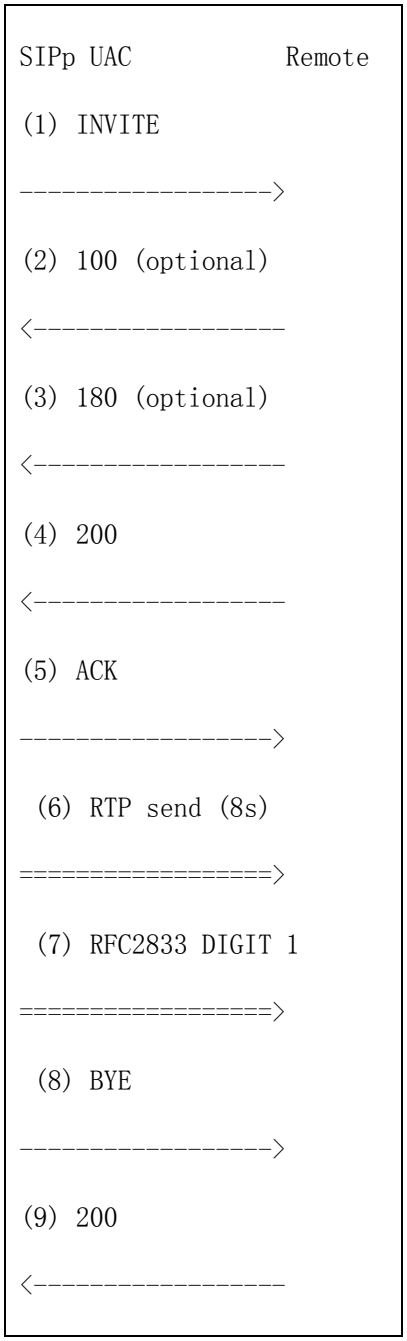
场景文件：uac.xml(uac.xml.html)(original XML 文件（uac.xml）

Scenario file: uac.xml (original XML file)



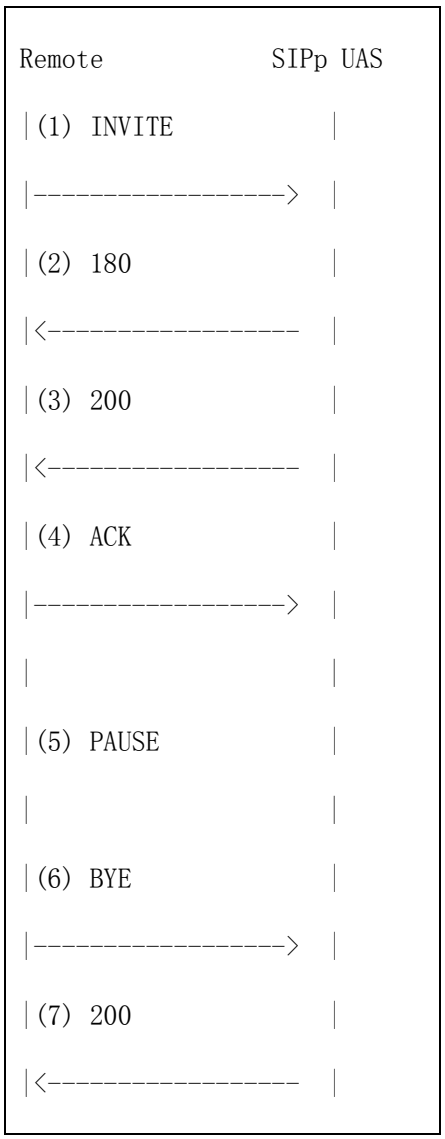
3.2.2. UAC with media(带有媒体流的UAC)

场景文件：uac_pcap.xml (original XML file)



3.2.3. UAS

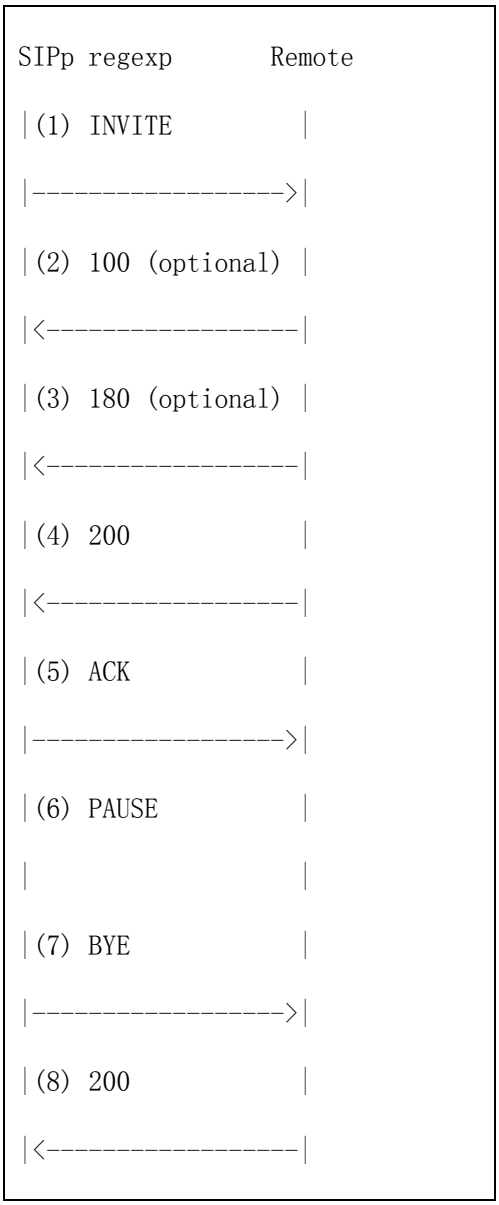
场景文件：uas.xml (original XML file)



3.2.4. regexp

场景文件：regexp.xml (original XML file))

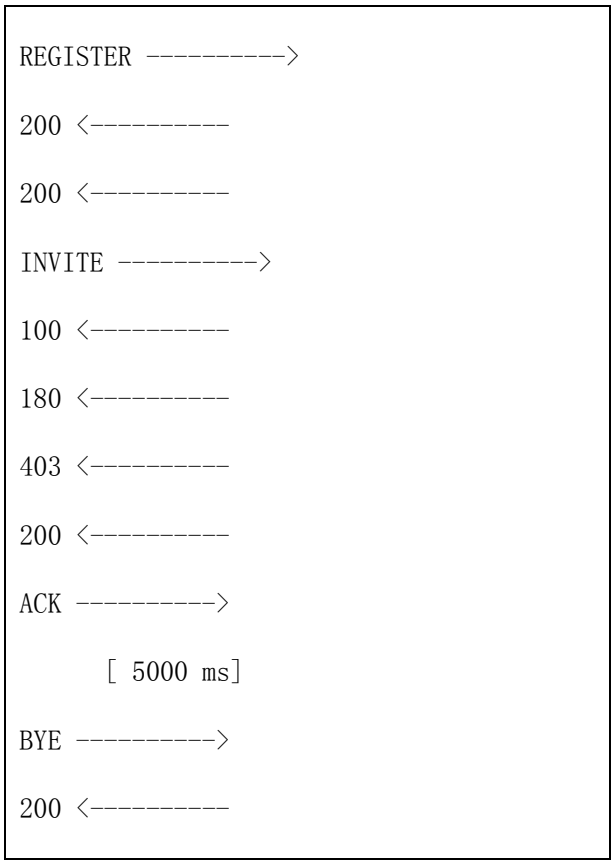
在此部分详细说明了UAC场景的操作



3.2.5. branch（分支）

场景文件： branchc.xml (original XML file) and branchs.xml (original XML file)

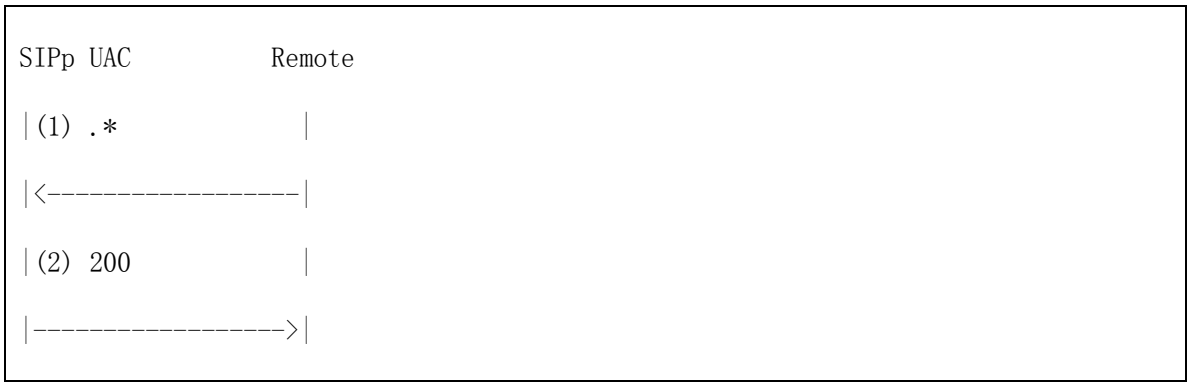
这些场景文件，协调工作(branchc为客户端，branchs为服务端)，这里有详细的介绍。



3.2.6. UAC Out-of-call Messages(UAC呼出消息)

场景文件：ooc_default.xml (original XML file)

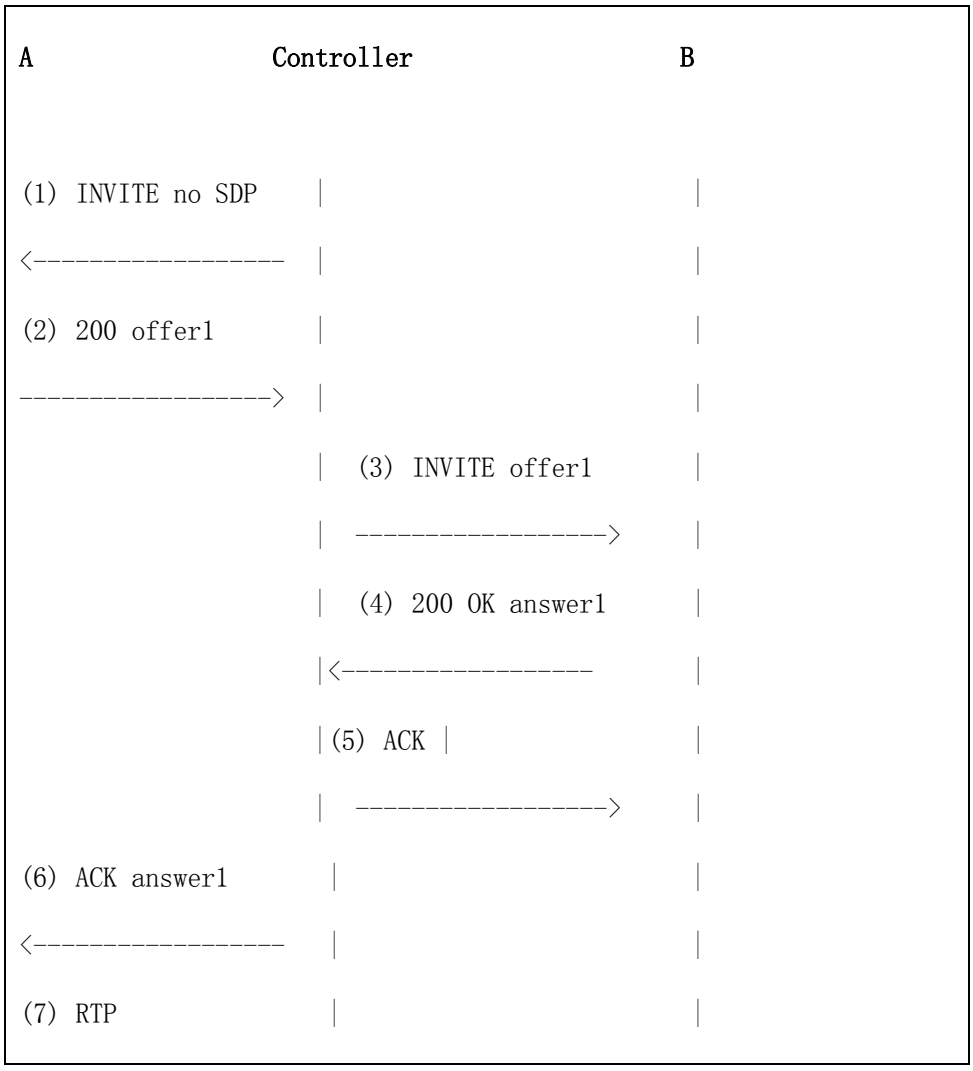
当SIPp UAC收到一个呼出的请求消息，它实例化一个out-of-call场景。通常情况下用200 OK简单的响应。通过用-oocsf或-oocsn命令行选项改写这个场景。

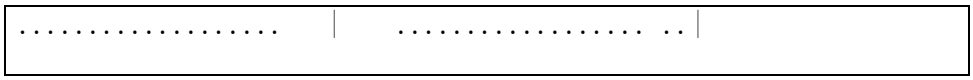


3.2.7. 3PCC

3PCC代表第3方呼叫控制，3PCC 描述请见 RFC 3725 (http://www.ietf.org/rfc/rfc3725.txt)，虽然第一次开发这个功能的时候应用3PCC场景，但是它也能用于各种情况,你仅需要一个SIPp跟几个远端通讯即可。

为了让SIPp更加简单（记住，它只是一个测试工具！），在3PCC呼叫流中有一个问题,那就是一个SIPp实例仅能和一个远端通讯,就像只能呼叫我一样(原文:one SIPp instance can only talk to one remote. Which is an issue in 3PCC call flows, like call flow I）。（SIPp只是作为一个控制器使用）





场景文件：3pcc-A.xml (original XML file)

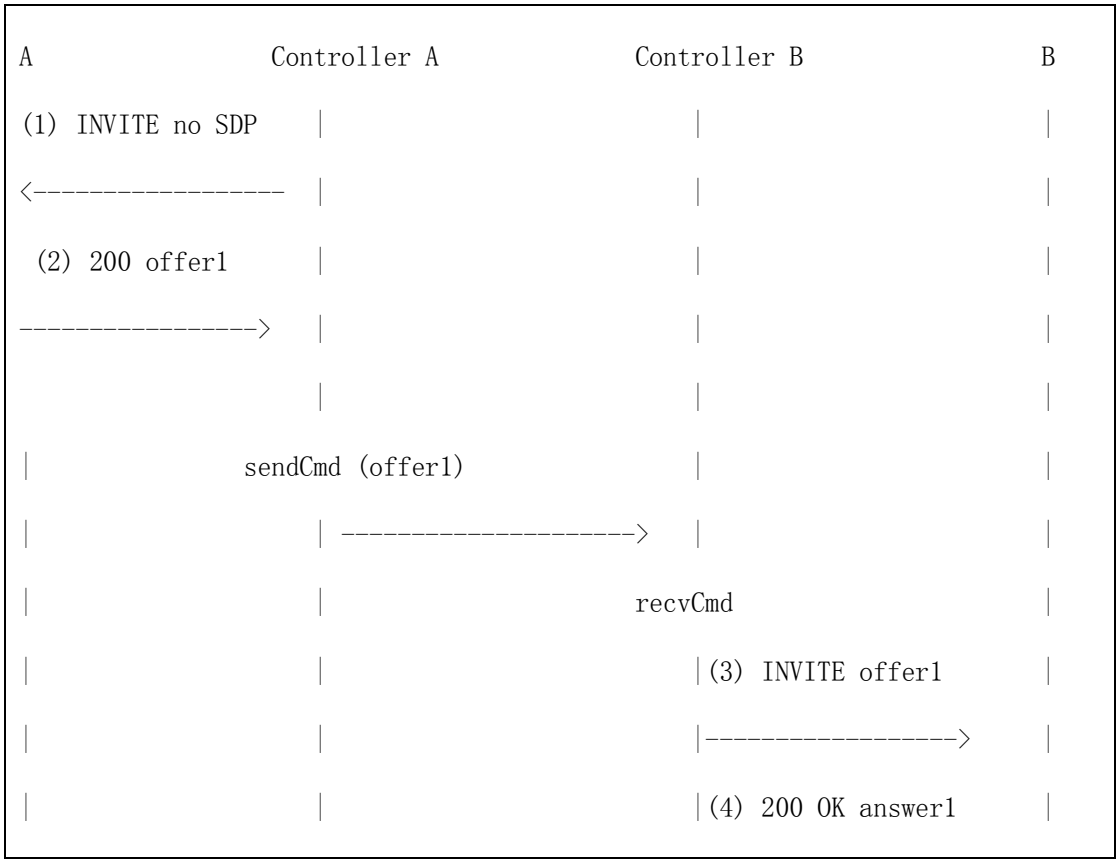
场景文件 3pcc-B.xml (original XML file)

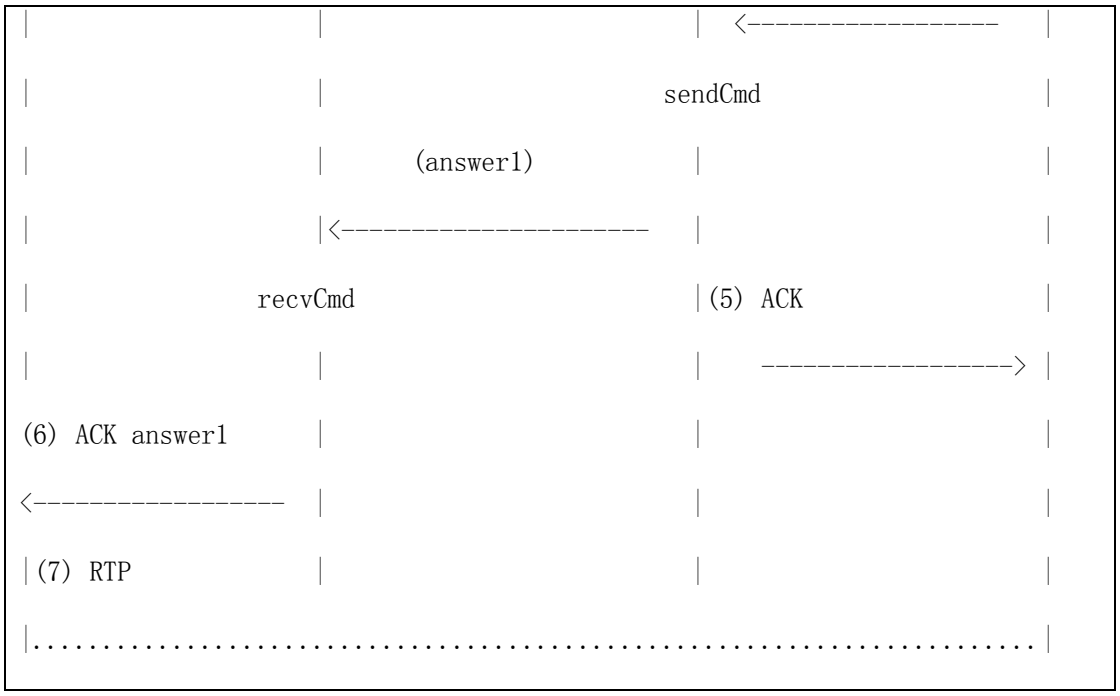
场景文件： 3pcc-C-A.xml (original XML file)

场景文件： 3pcc-C-B.xml (original XML file)

在SIPp中，使用3PC允许你有两个SIPp实例同步发送消息，如果我们拿呼叫流程为例，一个SIPp实例去和远端A通话（对3PCC-Controller-A端来说这个实体叫做3PCC-C-A），另一个SIPp实例和远端B通话（对3PCC-Controller-B端来说这个实体叫做3PCC-C-B）。

在现实中，3PCC呼叫流程如下（在两个SIPp实例中，控制器被分开）：





如你们所看到的，我们需要去手动配置控制器两端的信息，A端消息消息（2）提供的SDP“offer1”发送到B端的消息(3)中，在场景中通过<sendCmd>命令去实施这种机制，如下：

```
<sendCmd>
<![CDATA[
    Call-ID: [call_id]
    [$1]


]]>
</sendCmd>
```

这个命令将要发送给两个相同的SIPp实例,注意包含的Call-ID主要是为了和实际的

呼叫相关联，类似如下：

```
<recvCmd>
<action
    <ereg regexp="Content-Type:.*"
    search_in="msg"
    assign_to="2"/>
</action>
</recvCmd>
```

然后从两个SIPp实例中接收一个命令，我们用正则表达式机制重新提取上面的内容储存到变量（本例中为$2）中，

```
<send>
<![CDATA[
    ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0
    Via: SIP/2.0/[transport] [local_ip]:[local_port]
    From: sipp
    <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
    To: sut
    <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
    Call-ID: [call_id]
    CSeq: 1 ACK
    Contact: sip:sipp@[local_ip]:[local_port]
```

```
    Max-Forwards: 70

    Subject: Performance Test

    [$2]

]]>

</send>
```

换句话说，sendCmd和recvCmd可以被看作是两个SIPp实例中的同步点。具有携带相互传递参数的能力。

以下是据报道可行3PCC特性的另一个场景：

- A呼叫B，B应答，B和A通话
- B呼叫C，C应答，C和B通话
- B把A介绍给C，请求替换A-B呼叫与B-C的呼叫
- A接受，A和C通话，B退出。

3.3. 3PCC Extended(3PCC扩展)

在SIPp中可以实现3PCC模式的扩展，这个功能允许SIPp实例的任何号码相互通信，他们中的每一个都能连接到一个远端的机器。

初始化“主”模式下的SIPp实例发起呼叫，在“从”模式下发起其余的呼叫，“从SIPp实例”都有名称，名称在命令行中被赋值（例如：在从模式下为s1,s2,…sn，在主模式下为m),在一个文件中两个实例的名称和存储的地址必须一致(-slave_cfg命令行提供验证）,如下格式：

```
s1;127.0.0.1:8080

s2;127.0.0.1:7080
```

```
m;127.0.0.1:6080
```

每个SIPp实例必须访问一个该文件的副本。

sendCmd 和 recvCmd命令可以添加属性：

```
<sendCmd dest="s1">
  <![CDATA[
      Call-ID: [call_id]
      From: m
      [$1]
  ]]>
</sendCmd>
```

每个实例中都要给“s1”发送一个命令，可以是从模式也可以是主模式，依据命令行参数，场景必须始终一致:“从实例”在没有接收到任何recvCmd之前不能有一个sendCmd操作，注意，消息中必须包含一个“From”字段，内有发送者的名字。

```
<recvCmd src="m">
    <action
        <ereg regexp="Content-Type:.*"
        search_in="msg"
        assign_to="2"/>
```

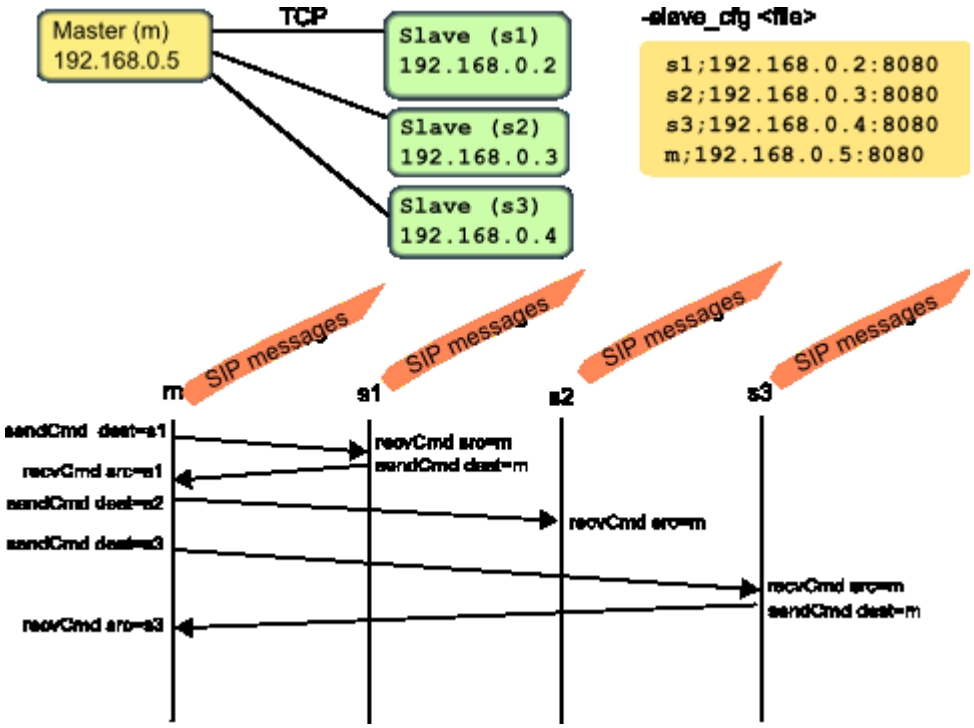
```
    </action>

</recvCmd>
```

这段代码表示期望从对端实例“m”中收到两个相同命令。

注意，主实例必须最后启动。

虽然这里没有为3PCC扩展模式集成场景，但你能很容易的配置。

例如：下面的图表画出了整个步骤，箭头展示SIPp主与从实例在两个不同的SIPp实例之间的同时进行命令交换，就像平常的SIP消息交换一样。



3.4. 控制SIPp

通过键盘或由UDP socket来与SIPp进行交互控制。SIPp同时支持热键和一些简单命令行模式，热键可以随时被使用，使用的热键有：

键值	操作
+	1倍于rate_scale的呼叫速率增加
*	10倍于rate_scale的呼叫速率增加
-	1倍于rate_scale的呼叫速率减少
/	10倍于rate_scale的呼叫速率减少
c	进入命令模式
q	退出SIPp(所有的呼叫完成后才退出)
Q	立即退出SIPp
s	储存屏幕信息至log文件中（仅当用了-trace_screen参数）
p	暂停
1	显示场景信息
2	显示统计信息
3	显示重发信息
4	显示变量信息
5	显示TDM信息
6-9	显示第2到第5次重发的信息

在命令模式中，你能输入一行单独的命令来指示SIPp去执行一些操作。命令模式比热键更加灵活。但需要花费更多的时间来输入操作。下面是有效的命令：

命令	描述	例子
dump tasks	打印操作任务列表（大多数任务是呼叫）到错误日志中	dump tasks
set rate X	设置呼叫速率	set rate 10
set rate-scale X	设置速度步长，通过'+', '-', '*', 和 '/'	set rate-scale 10
set users X	设置用户数量（仅当指定 -users时有效）	set users 10
set limit X	设置打开呼叫限制数量（等同于 -l 选项）	set limit 100
set hide <true|false>	是否显示隐藏的XML属性	set hide false
set index <true|false>	在场景屏幕中显示消息索引	set index true
set display <main|ooc>	改变显示页面，在主体或呼出的任意一个场景进行切换	set display main set display ooc
trace <log> <on|off>	在运行时打开或关闭日志，日志的有效值有"error"、"logs"、"messages"、和"shortmessages"	trace error on

表2：交互命令列表

3.4.1. Traffic control（流量控制）

SIPp 根据指定的场景生成 SIP 流量。在启动的时候你能控制每秒呼叫的数量。如果你用了 -users 选项，你需要控制实例化用户的数量。你能控制速率，通过用：

- 交互热键（以前的部分介绍过）
- 交互命令
- 启动时指定的参数

有两个命令控制速率：set rate X 设置当前呼叫速率为 X。另外一个是 set rate-scale X 设置速率步长（rate-scale）参数为 X。

这样可以更加快速的用'+'，'-'，'*'，和 '/' 来设置速率。例如，如果你设置了 set rate-scale 100，每次按'+'键，以 100 个呼叫的速率增加，每次按‘*’键，以 1000 个呼叫的速率增加。类似地，以一个用户数量为基准，你能运行 set users X 命令。在启动的时候，你能在命令行中指定参数来控制速率：

- “-r”来指定每秒呼叫的数量
- “-rp” 以毫秒为单位指定“速率周期”的呼叫（默认是1000毫秒=1秒），这样就允许每m毫秒内有n个呼叫（使用命令“-r n -rp m”）。

> **注意：**
>
> 例如：在运行 SIPp 时，每 2 秒 7 个呼叫（每秒 3.5 个呼叫）

```
./sipp -sn uac -r 7 -rp 2000 127.0.0.1
```

可以通过按‘p’键来暂停呼叫，SIPp会停止增加新的呼叫，并且等待直到当前所有的呼叫都发起，你可以再次按‘p’键来恢复。

按‘q’键来退出SIPp，SIPp会停止新的呼叫，并且等到当前所有的呼叫都结束，然后SIPp才退出。

你也可能通过按‘Q’键来强制退出SIPp，当前的呼叫会发送一个BYE或CANCEL消息来立即结束（这依赖呼叫是否建立还是没有建立）。按两次‘q’可以达到相同的功能。

> **注意：**
>
> 小提示：你可以用-m选项的命令来放置一个已定义的呼叫数，当到达此数后，SIPp自动退出。

3.4.2. Remote control（远程控制）

通过一个 UDP socket 可以实现远程控制 SIPp，如下面的例子：

- 为了自动完成一系列的操作，像平缓的增加呼叫速率，等待10秒增长一些，一分钟后循环；
- 一个反馈回路，被测试程序可以遥控SIPp来进行降低负载，暂停呼叫，．．

一个SIPp实例去监听一个UDP socket,从8888端口开始，以后每个SIPp实例（最多60个）会监听base_port + 1(8889,8890…)。

控制SIPp类似下面：

```
echo p >/dev/udp/x.y.z.t/8888 -> put SIPp in pause state (相当于p键)

echo q >/dev/udp/x.y.z.t/8888 -> quit SIPp (相当于q键)
```

> **注意：**

在远程控制接口端可以用键盘来输入有效的关键字也生效。

你也可以用一个小的SHELL脚本来自动完成一系列操作，例如，这个脚本展示每5秒来增加10个呼叫，保持此呼叫速率1分钟，然后退出SIPp：

```
#!/bin/sh
echo "*" >/dev/udp/127.0.0.1/8889
sleep 5
echo "*" >/dev/udp/127.0.0.1/8889
sleep 5
echo "*" >/dev/udp/127.0.0.1/8889
sleep 5
echo "*" >/dev/udp/127.0.0.1/8889
sleep 60
echo "q" >/dev/udp/127.0.0.1/8889
```

为了给SIPp发送一个命令，以‘c’开始。例如：

```
echo "cset rate 100" >/dev/udp/127.0.0.1/8888
```

设置呼叫速率到100。

3.5. Running SIPp in background（后台运行SIPp）

SIPp可以在后台模式下运行（加上 -bg 命令行选项）

通过这样做，SIPp可以从当前终端分离出来，以后台模式运行，SIPp进程提供PID，如果你不添加 -m选项来指定执行呼叫的次数，SIPp会一直运行。

有一种机制来平缓地停止SIPp,运行命令 kill -SIGUSER1 [SIPp PID] 来通知SIPp停止一切新的呼叫，并且在退出以前完成所有现在正在进行的呼叫。

当用背景模式时，主SIPp实例停止时，子进程会继续工作。因此，日志文件名称会包含另一个PID，而不是实际的sipp实例的PID。

3.6. Create your own XML scenarios（创建自定义XML场景）

当然，内置的场景是远远这够的，有时需要去自定义你自己的场景，一个SIPp场景写在XML文件中，（一个存在的DTD也许会帮助你写SIPp场景，它用jEdit测试过。在下面的代码中有描述），一个场景文件通常以

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<scenario name="Basic Sipstone UAC">
```

开始，以

```
</scenario>
```

结束。

容易吧！嗯？好吧，现在让我们来看看里面可以放些什么，你不必现在就读整个表，仅仅看代码中的例子就行了。

有许多公共属性用于呼叫流量控制和统计数据,这些属性可用于所有的消息命令(例如, <send>, <recv>, <nop>, <pause>,<sendCmd>, and <recvCmd>)。

属性	描述	例子
start_rtd	启动"Response Time Duration"计时器中的一个(查看statistics section)	<send start_rtd="invite">: 将开始发送名为"invite"的定时器消息
rtd	停止5个"Response Time Duration"中的一个	<send rtd="2">: 当发送了消息后，计时器2号停止计时。
repeat_rtd	使用携带的rtd属性，它允许每个呼叫不止一次计算相应的"Response Time Duration"计时器。	<send rtd="1" repeat_rtd="true">: 打印计时器号码1的值，但计时器不会停止。
crlf	在主SIPp屏幕消息的箭头后面显示一个空行。	<send crlf="true">
next	当发送消息时，你能在任意命令中放置“next”去跳转至脚本的另一部分。如果仅收到了为next的消息，为可选接收。参考conditional branching获得更多信息。	发送ACK后跳到标签“12”的例子： <send next="12"> <![CDATA[ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0 Via: ...

		From: ... To: ... Call-ID: ... Cseq: ... Contact: ... Max-Forwards: ... Subject: ... Content-Length: 0]]> </send> 收到一条403消息时跳到标签“5”的例子： <recv response="100" optional="true"> </recv> <recv response="180" optional="true"> </recv> <recv response="403" optional="true" next="5"> </recv> <recv response="200"> </recv>
test	如果在“test”设置了指定变量，你能在靠近“next”属性旁放置“test”来指出你仅能跳转指定了	仅当设置了变量4时，发送一个ACK后才跳转于标签“6”的例子：

	携带“next”的标签处。 查看 conditional branching部分获得更多信息	<send next="6" test="4"> <![CDATA[ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0 Via: ... From: ... To: ... Call-ID: ... Cseq: ... Contact: ... Max-Forwards: ... Subject: ... Content-Length: 0]]> </send>
chance	与”test”结合使用，实际上是转到场景中其它位置的概率，chance是0（never）与1（always）。参考 conditional branching获得更多信息。	<recv response="403" optional="true" next="5" test="3" chance="0.90"> </recv>

		如果已经设置了变量“3“，90%的机率会跳转到标签“5”。
condexec	仅当设置在condexec属性中设置了变量时执行一个元素。这个属性允许你在复杂的XML场景有更少的属性和标签。	<nop condexec="executethis">
condexec_inverse	如果设置了condexec，condexec_inverse 与condexec相反。仅当没有设置变量时执行一个元素	<nop condexec="skipthis" condexec_inverse="true">
counter	当发送一条消息时，增加的计数器作为参数，计数器保存在statistic file	<send counter="MsgA">: 当发送消息时，计数器"MsgA"加1。

表 1：常用的所有命令属性列表

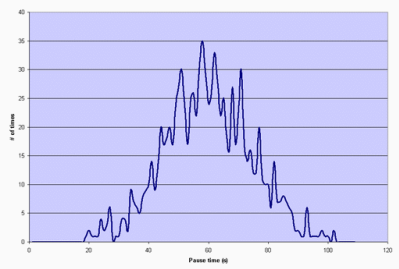
每一个命令也有它自己唯一属性，如下列表：

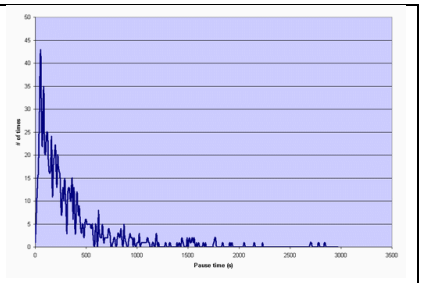
命令	属性	描述	范例
<send>	retrans	仅仅UDP传输：在SIP RFC 3261 17.1.1.2.章中描述，它指定T1计时器的值	<send retrans="500"> 初始化T1计数器为500毫秒。(RFC3261 default).
	lost	用百分比模仿丢失包	<recv lost="10">:消息中的10%没有发送。
	start_txn	记录本次发送消息的branch ID，以便正确的匹配响应消息。	<send start_txn="invite">: 在名为"invite"的事物中储存这条消息的branch ID。

		（如果没有此元素，则用Cseq方式完成事务匹配，但不精确）	
	ack_txn	标明通过start_txn属性开始发送ACK事物响应。每一个带有start_txn传记的INVITE必须与一个带ack_txn属性的ACK相匹配。	<send ack_txn="invite">: 参考名为"invite"事物的branch ID
<recv>	response	期望收到的sip代码	<recv response="200">: SIPp期望收到一个带有“200”代码的SIP消息
	request	期望收到请求的SIP消息	<recv request="ACK">: SIPp期望收到带有"ACK"的SIP消息.
	optional	收到的消息是可选的，万一如果没有收到或实际上收到了这样的一条消息，视为未知情况下所做的处理功能。	<recv response="100" optional="true">: 可以作为在预期没有考虑到的情况下收到了100 SIP 消息。
	rrs	记录路由设置，如果属性设置为“true”，用[router]关键字重呼和保存接收到“Record-Route:“消息头	<recv response="100" rrs="true">.
	auth	认证，如果此属性设置为”true”， 会用	<recv

		[authentication]关键字重呼并保存收到的"Proxy-Authenticate:"消息头	response="407" auth="true">.
	lost	模拟丢包率：用百分比指定一个值	<recv lost="10">: 丢掉收到消息的10%
	timeout	指定等待一条消息的超时时间。如果没有收到消息，停止呼叫，除非定义一个ontimeout 标签。	<recv timeout="100000">
	ontimeout	如果在收到消息之前已经超时，跳到指定的标签。	例子：当100秒生还没有收到100消息，跳到标签“5”处： <recv response="100" timeout="100000" ontimeout="5"> </recv>
	action	当收到消息时指定一个操作。请参考Actions section	正则表达式的操作的例子： <recv response="200"> <action> <ereg regexp="([0-9]{1,3}\.){3}[0-9]{1,3}:[0-9]*" search_in="msg" check_it="true" assign_to="1,2"/>

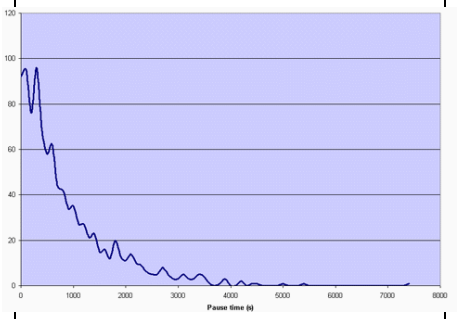
			</action> </recv>
	regexp_match	布尔型。表明如果请求是一个正则表达式（响应不起作用），那么recv命令匹配正则表达式，这样允许在相同的接收命令中捕获几个事件。	一个接收命令中匹配MESSAGE或PUBLISH 或SUBSCRIBE请求的例子： <recv request="MESSAGE|PUBLISH|SUBSCRIBE" crlf="true" regexp_match="true"> </recv>
	response_txn	表明这是一个启动原来事物的响应。为了与之匹配，第一个via消息头的branch ID必须与储存的事物ID相匹配。	<recv response="200" response_txn="invite" />: 仅与带有start_txn="invite"属性的发送响应消息相匹配
<pause>	milliseconds	指定停顿延迟，以毫秒为单位。 当没有设置延迟时间时，使用-d命令行参数的值。	<pause milliseconds="5000"/>: 暂停5秒的场景。
	variable	声明决定暂停的呼叫变量	<pause variable="1" /> 用呼叫变量1指定暂停的毫秒数
	distribution	表示统计分布，用来决定暂时的时间。不使用GSL，你可以用统一的或固定的值，如果使用GSL,可选用标准，指数，函数、匿名函数、对数正态分布、负二项分布、帕雷托和威布尔（韦伯）	以下例子展示了各种暂停分布的类型： ● <pause distribution="fixed" value="1000" />

		（原文：With GSL, normal, exponential, gamma, lambda, lognormal, negbin, (negative binomial), pareto, and weibull are available.），依赖于你选择分布模式，你也必须用参数指定支持的分布。	暂停1秒。 ● <pause distribution="uniform" min="2000" max="5000"/> 在2至5秒内暂停 剩下的分布要求有GSL。一般情况下，选择的参数名称与维基百科的分布描述页面一致。 ● <pause distribution="normal" mean="60000" stdev="15000"/> 提供了一个正常的停顿，即平均60秒暂停一次。（例如：60000 ms）和15秒的标准偏差。平均值和标准偏差被指定为整数毫秒。分布图类似下图：  ● <pause lognormal="true" mean="12.28" stdev="1" /> 创建一个分布的自然对数的平均值12.28,标准偏差为1。平均值和标准偏差被指定为双值（用毫秒），分布图类似下图：



- <pause exponential="true" mean="900000"/>

创建一个15分钟的指数分布暂停分布，分布图类似下图：



- <pause weibull="true" lambda="3" k="4"/>

创建一个尺度参数为3和形状参数为4的威布尔分布暂停。

（参考Weibull on Wikipedia (http://en.wikipedia.org/wiki/Weibull)分布的例子）

- <pause distribution="gamma" k="3" theta="2"/>

分别创建一个带有K、9和2

			时间递耗的伽马分布(在Wikipedia上查看伽马分布(http://en.wikipedia.org/wiki/Gamma_distribution) 的描述信息). ● <pause distribution="negbin" p="0.1" n="2"/> 分别创建一个带有p、0.1和2的n负二项分布(在Wikipedia上查看负二项分布(http://en.wikipedia.org/wiki/Negative_binomial_distribution))的描述）
	sanity_check	默认情况下，完整性检查,统计分布暂停，以确保他们的第99百分位值小于INT_MAX。设置**sanity_check为false**时不启用此功能。	<pause distribution="lognormal" mean="10" stdev="10" sanity_check="false"/> 不启用lognormal distribution（对数正态分布）检查
<nop>	action	nop指令在SIP级别不做任何事情。仅指定一个行动来执行，请参考Actions section。	执行play_pcap_audio/video操作： <nop> <action> <execplay_pcap_audio="pcap/g711a.pcap"/> </action> </nop>

<sendCmd>	<![CDATA[]]>	内容发送到双3PCC SIPp实例。在CDATA必须包含Call-ID。在3 pcc扩展模式下,必须包括From。	<sendCmd> <![CDATA[Call-ID: [call_id] [$1]]]> </sendCmd>
	dest	仅用在3 pcc扩展模式，命令被发送到两个sipp实例中	<sendCmd dest="s1">: 命令将被送到“s1”两个实例
<recvCmd>	action	当接收到一个命令时指定一个操作，参考Actions Section。	通过sendCmd 命令发送正则表达式检索： <recvCmd> <action <ereg regexp="Content-Type:.*" search_in="msg" assign_to="2"/> </action> </recvCmd>
	src	仅用在3 pcc扩展模式，表明两个SIPP实例从预期中接收命令。	<recvCmd src ="s1">: 期望从s1两个实例中接收命令

<label>	id	当你想到跳转到场景的指定部分，你可以设置一个标签，’id’属性是一个整型，最大值为19，查看conditional branching部分获得更多信息	设置标签号为13的例子： <label id="13"/>
<Response Time Repartition>	value	指定间隔，用于分发的响应时间的值，用毫秒表示。	<ResponseTimeRepartition value="10, 20,30"/>: 响应时间值分布介于0和10 ms,10和20 ms,20和30 ms,30ms之外。
<Call Length Repartition>	value	指定间隔，用于测量分布呼叫长度的值。	<CallLengthRepartition value="10, 20,30"/>: 呼叫长度值分布介于0和10 ms,10和20 ms,20和30 ms,30ms之外。
<Globals>	variables	规定全局范围变量的名称	<Globals variables="foo,bar" />.
<User>	variables	指定user-scoped变量的名称	<User variables="foo,bar" />
<Reference>	variables	抑制未使用警告的变量	<Reference variables="dummy" />

表1:带有属性的命令列表

没有什么比一个例子更加清晰表达send, recv, sendCmd, recvCmd, pause, ResponseTimeRepartition and CallLengthRepartition这些命令了……

3.6.1. Structure of client (UAC like) XML scenarios（客户端（如UAC）XML场景结构）

一个客户端场景用"send"命令开始，如下：

```
<scenario name="Basic Sipstone UAC">
<send>
<![CDATA[
        INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
        Via: SIP/2.0/[transport] [local_ip]:[local_port]
        From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
        To: sut <sip:[service]@[remote_ip]:[remote_port]>
        Call-ID: [call_id]
        Cseq: 1 INVITE
        Contact: sip:sipp@[local_ip]:[local_port]
        Max-Forwards: 70
        Subject: Performance Test
        Content-Type: application/sdp
        Content-Length: [len]


        v=0
        o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]
        s=-
        t=0 0
        c=IN IP[media_ip_type] [media_ip]
        m=audio [media_port] RTP/AVP 0
        a=rtpmap:0 PCMU/8000
```

```
]]>

</send>
```

在"send"命令内部，你必须用"<![CDATA"和"]]>"标记包围SIP消息。那些标记之间的内容被发送到远端，你可能已经注意到，在SIP消息中有一些奇怪的关键字，像：[service], [remote_ip], ...，那些关键字用于表示SIPp用它来做些什么。

列表如下：

关键字	默认	描述
[service]	service	service字段，通过用-s service_name命令
[remote_ip]	-	远程IP地址，通常在命令行中使用
[remote_port]	5060	远程主机端口，你可以向这个值中添加一个计算偏移量[remote_port+3]
[transport]	UDP	使用-t参数传值来使用"UDP" or "TCP"
[local_ip]	本地IP地址（原文：Primary host IP address）	使用”-i”参数传值
[local_ip_type]	-	依赖于-i参数(IPv4 or IPv6),的地址类型，local_ip_type值为4时类型为IPv4或值为6时类型为IPv6
[local_port]	随机值(原文：Random)	携带-p参数的值，你可以向这个值中添加一个计算偏移量[local_port+3]。

[len]		计算SIP body的长度，通常用在"Content-Length"消息头。你可以向这个值中添加一个计算偏移量[len+3]。
[call_number]	-	指数。call_number从1开始，随着呼叫数量的递增加1
[cseq]	-	自动生成Cseq号码，默认初始值为1，可以用“-base_cseq”命令行选项改变默认值。
[call_id]	-	一个call_id识别一个呼叫，SIPp为每一个新呼叫生成一个call_id，在客户端模式中， SIPp在Call-ID"消息头强制生成。除非，SIPp没有识别出已经存在呼叫的部分消息发送的应答， 注意：[call_id]可以用’///’预置加上任意字符串，如：Call-ID: ABCDEFGHIJ///[call_id] SIPp仍然会识别相同的呼叫
[media_ip]	-	依赖于-mi参数的值，它是RTP流的本地地址。
[media_ip_type]	-	依赖于-mi参数（IPv4 or IPv6）的值，media_ip_type值为4时类型为IPv4或值为6时类型为IPv6，构造有效的SDP与媒体类型无关。
[media_port]	-	依赖于-mp参数的值，设置本机地址RTP的端口号，默认为没有设置，在这个端口上接收RTP/UDP包，你可以向这个值中添加一个计算偏移量[local_port+3]。
[auto_media_port]	-	仅为PCAP,从-mp参数的设置音频和视频端口号的值开始，周期性的为每个呼叫改变，以10000为模（为pcap_play限制10000个并发RTP会话）

[last_*]	-	如果在上一次收到的消息中存在，'[last_*]'关键字通过特定的头被自动替换（除非被转发），如果消息头不存在或没有收到任何消息，将会丢弃'[last_*]'关键字，至行的最后所有字节也被丢弃，如果在消息中出现了几次规定的消息头，所有出现'[last_*]'关键字的地方被用于连接(CRLF分割)起来。
[field0-n]	-	通常从外部CSV文件中插入值，参考"Injecting values from anexternal CSV during calls"部分
[$n]	-	通常插入呼叫变量号码n的值，参考"Actions"部分
[authentication]	-	通常放在认证头，这个字段有参数，用下面的格式：[authentication username=myusername password=mypassword]，**如果没有提供username，用-s命令行参数（服务原文为service）的值，如果没有提供password，用**-ap命令行参数的值，参考"Authentication"部分
[pid]	-	在主SIPp线程中进程ID（pid）
[routes]	-	如果在recv命令中”rrs”属性设置为"true"，会存储收到消息的"Record-Route:"头，并用[routes]关键字重新呼叫；
[next_url]	-	如果在recv命令中”rrs”属性设置为"true"，[next_url]会包含Contact头的内容（例如Contact在'<' 和 '>'）
[branch]	-	在场景中提供一个分支值，此值用魔法cookie(z9hG4bK) +呼叫号码+消息索引串联起来。
[msg_index]	-	在场景中提供消息号码
[cseq]	-	提供上一次收到请求的Cseq值，这个值会增加（例如，[cseq+1]加1至上一次请求的Cseq值中）

表1：关键字列表

现在发送INVITE消息，SIPp用"recv"命令等待一个应答

```
<recv response="100"> optional="true"
</recv>
<recv response="180"> optional="true"
</recv>
<recv response="200">
</recv>
```

100和180消息是可选的，200是必须存在的，**在"recv"顺序中，必须有一个必选消息。**

现在，让我们发送ACK:

```
<send>
  <![CDATA[
      ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port]
      From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
      To: sut <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
      Call-ID: [call_id]
      Cseq: 1 ACK
```

```
      Contact: sip:sipp@[local_ip]:[local_port]
      Max-Forwards: 70
      Subject: Performance Test
      Content-Length: 0
  ]]>
</send>
```

我们也能插入暂停。场景会在这个点上等待5秒钟。

```
<pause milliseconds="5000"/>
```

发送一个BYE并期望收到2000K来完成一人呼叫：

```
<send retrans="500">
<![CDATA[
BYE sip:[service]@[remote_ip]:[remote_port] SIP/2.0
Via: SIP/2.0/[transport] [local_ip]:[local_port]
From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
To: sut <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]
Call-ID: [call_id]
Cseq: 2 BYE
Contact: sip:sipp@[local_ip]:[local_port]
```

```
Max-Forwards: 70

Subject: Performance Test

Content-Length: 0

]]>

</send>

<recv response="200">

</recv>
```

这个场景的结束：

```
</scenario>
```

创建你自己的SIPp场景没有什么大不了的。如果你想参考其它例子，在命令行中用-sd参数去显示嵌入的场景。

注意：现在SIPp支持短格式的消息头（例如："Call-id:"能用"i:"代替）。

3.6.2. Structure of server (UAS like) XML scenarios（构造服务端(如UAS)XML场景）

一个服务器场景用”recv”命令开始，它的语法和命令的列表和“客户端”场景是的相同的。但是在服务端场景中你可能更多的用[last_*]关键字，一个UAS的例子看起来像：

```
<recv request=” INVITE” >
```

```
</recv>

<send>

   <![CDATA[

      SIP/2.0 180 Ringing

      [last_Via:]

      [last_From:]

      [last_To:];tag=[call_number]

      [last_Call-ID:]

      [last_Cseq:]

      Contact: <sip:[local_ip]:[local_port];transport=[transport]>

      Content-Length: 0

   ]]>

</send>
```

在这个例子中，在收到INVITE消息头的内容中，180响铃是一个应答消息。

3.6.3. Actions（操作）

在"recv"或"recvCmd"命令中，你有可能执行一个操作。有几种操作:

- 正则表达式(ereg)
- 在aa日志文件中记录一些日志（log）
- 执行外部(系统)、内部(int_cmd)或pcap_play_audio / pcap_play_video命令(exec)
- 使用算法操作双精度变量
- 将字符串值分配给一个变量

- 比较双精度变量
- 跳转到一个特定的场景索引
- 把当前时间存储到变量中
- 在一个有索引的注入文件中查找一个值
- 验证授权证书
- 改变一个呼叫的目的地址

3.6.3.1. 正则表达式（Regular expressions）

在SIPp中使用正则表达式能够：

- 提取SIP消息或SIP头的内容并存储供以后使用，（回呼：called re-injection）
- 检查一部分SIP消息或消息头是否匹配一个期望的表达式

在SIPp中按照Posix Extended standard (POSIX 1003.2) (http://www.opengroup.org/onlinepubs/007908799/xbd/re.html) 定义正则表达式。如果你想学习如何写正则表达式，我推荐regexp资料 (http://analyser.oli.tudelft.nl/regex/index.html.en)，（注：反正我是在网上没有搜索到此网址。）

这是regexp的语法：

关键字	默认	描述
regexp	none	用于匹配收到的消息或头 必选项
search_in	msg	有两个值："msg"（尝试匹配整个消息）或"hdr"（尝试匹配一个指定的SIP头）

header	None	尝试匹配消息头，只用在hdr中设置search_in标记时，如果search_in等于hdr时必选
case_indep	false	寻找忽略的消息头，仅用于在hdr中设置了search_in标记。
occurence	1	找到第n个出现的消息头，仅用于在hdr中设置了search_in标记
start_line	false	仅看唯一的起始行，仅用于在hdr中设置了search_in标记
check_it	false	如果设置为true，如果regexp不匹配，呼叫标记为失败。
assign_to	None	包含变量id（整型）或变量id的列表，它被用来去储存正则表达式和消息匹配的结果。这个变量在以后场景中可以被重用，在消息中通过'[$n]'插入变量的值或变量的内容被用于有条件的分支，assign_to变量列表的第一个变量了包含整个正则表达式匹配，例如： <ereg regexp="o=([[:alnum:]]*) ([[:alnum:]]*) ([[:alnum:]]*)" search_in="msg" check_it=i"true" assign_to="3,4,5,8"/> 如果SIP消息中包含了行： o=user1 53655765 2353687637 IN IP4 127.0.0.1 变量3包含： "o=user1 53655765 2353687637",

		变量4包含： "user1", 变量5包含： "53655765" 变量8包含："2353687637".

表1：正则表达式操作语法

注意：在一个操作中可以有几个正则表达式。

下面的例子经常会被用到：

- 第一个操作：
- 提取收到SIP消息的首个Ipv4地址
- 检查提取到的IP地址（除非呼叫标记为失败）
- 把提取到的IP地址分配到呼叫变量1和2

- 第二个操作：
- 提取Contact:收到IP消息头
- 分配提取的Contract:设置变量6为标题

```
<recv response="200" start_rtd="true">

  <action>

    <ereg regexp="([0-9]{1,3}\.){3}[0-9]{1,3}:[0-9]*" search_in="msg"
    check_it="true" assign_to="1,2" />

    <ereg regexp=".*" search_in="hdr" header="Contact:" check_it="true"
    assign_to="6" />

  </action>
```

```
</recv>
```

3.6.3.2. Log a message（记录消息）

“log”操作让你自定义自己的轨迹，消息打印在<scenario file name>_<pid>_logs.log文件中。任何扩展的关键字来反映实际使用的值。

> **警告：**
>
> 只有在命令行中设置了-trace_logs才会生成日志

例子：

```
<recv request="INVITE" crlf="true" rrs="true">

   <action>

      <ereg regexp=".*" search_in="hdr" header="Some-New-Header:"
      assign_to="1" />

      <log message="From is [last_From]. Custom header is [$1]"/>

   </action>

</recv>
```

3.6.3.3. Execute a command（执行命令）

"exec"操作允许你执行"internal", "external", "play_pcap_audio"或"play_pcap_video"命令。

内部命令

内部命令（规定用int_cmd 属性)来停止呼叫，缓慢停止（类似于按’q’键），立即停止（类似于按Ctrl+C键）。

当收到“603”响应时停止脚本执行的例子：

```
<recv response="603" optional="true">
  <action>
    <exec int_cmd="stop_now"/>
  </action>
</recv>
```

外部命令

外部命令（用命令属性指定）是在本机中用一个 shell 执行任何操作。

执行一个每收到一个 INVITE 便在系统回显的例子:

```
<recv request="INVITE">
  <action>
    <exec command="echo [last_From] is the from header received >> from_list.log"/>
  </action>
</recv>
```

3.6.3.4. Media/RTP commands(媒体/RTP命令)

RTP streaming（RTP流）允许你用PCMA、PCMU、G729编码音频文件（如a.wav文件）去播放音频，这些用"rtp_stream"操作来控制。

- <exec rtp_stream="file.wav" />会播放file.wav中的音频流。假设它是一个PCMA格式的文件。

- <exec rtp_stream="[filename],[loopcount],[payloadtype]" />会播放[filename]中的音频流，重复播放[loopcount]次（默认为1，-1代表不重复播放），以[payloadtype]类型播放音频（默认为8代表PCMA,0代表PCMU，18代表G729）。

- <exec rtp_stream="pause" />暂停当前所有的回放/播放。

- <exec rtp_stream="resume" />恢复当前所有的回放/播放。

PCAP play命令（规定用play_pcap_audio / play_pcap_video属性）允许你用[pcap library](http://www.tcpdump.org/pcap3_man.html)发送一个预录的RTP流(http://www.tcpdump.org/pcap3_man.html)

用“m =audio”SIP / SDP线路端口作为基础重播选择**play_pcap_audio**发送预录的RTP流。

用“m =video”SIP / SDP线路端口作为基础来选择**play_pcap_video**发送预录的RTP流

play_pcap_audio/video命令有以下格式：play_pcap_audio="[file_to_play]带有

- file_to_play：播放预录的pcap文件

-

> **注意：**
>
> 操作无阻塞，SIPp开启一个高亮线程去立即播放文件和场景，如果有必要，你需要添加一个暂停操作直到pcap播放完。

> 警告：
>
> 一个已知的BUG是在启动pcap_play_audio命令会结束所有的pcap_play_video命令，反之亦然，你不能一次同时播放音频和视频流。

播放预录RTP流的例子：

```
<nop>
  <action>
        <exec play_pcap_audio="pcap/g711a.pcap"/>
   </action>
</nop>
```

3.6.3.5. Variable Manipulation（变量操作）

你也可以在浮点支点中执行一个简单的算法（加、减、乘、除）。"assign_to"属性包含在第一种运算中。这也是产生值的目的。第二个操作数是立即值或存储在一个变量中，分别由"value" 和"variable" 属性代表。

SIPp支持调用变量双精度浮点值。修改双变量的操作都通过**assign_to参数**写到引用的变量中。这些变量可以使用三种分配操作：assign, sample, 或todouble（赋值、抽样或方法转换）。对于assign，双精度值存放在"value"参数中。sample操作基于统计分布赋值，用相同的参数作为一个统计分布式停顿（statistically distributed pauses）。最后，todouble命令转换“变量”属性的变量所引用，设定给它之前的两倍。

例如，给$1分配值1.0，到正态分布$2中抽样：

```
<nop>
  <action>
     <assign assign_to="1" value="1" />
     <sample assign_to="2" distribution="normal" mean="0" stdev="1"/>
     <!-- Stores the first field in the injection file into string variable $3.
     You may also use regular expressions to store string variables. -->
```

```
    <assignstr assign_to="3" value="[field0]" />

    <!-- Converts the string value in $3 to a double-precision value stored in
    $4. -->

    <todouble assign_to="4" variable="3" />

  </action>

</nop>
```

也可能用<add>, <subtract>, <multiply>和<divide>进行一些简单的算法，在value中通过assign_to算法进行加、减、乘、除变量引用。例如，下面的操作会修改变量1：

```
<nop>

  <action>

    <assign assign_to="1" value="0" /> <!-- $1 == 0 -->

    <add assign_to="1" value="2" /> <!-- $1 == 2 -->

    <subtract assign_to="1" value="3" /> <!-- $1 == -1 -->

    <multiply assign_to="1" value="4" /> <!-- $1 == -4 -->

    <divide assign_to="1" value="5" /> <!-- $1 == -0.8 -->

  </action>
```

而不是一个确定的值，你也可以从一个变量中取回第二个运算，用<variable>参数，例如：

```
<nop>
   <action>
      <!-- Multiplies $1 by itself -->
      <multiply assign_to="1" variable="1" />
      <!-- Divides $1 by $2, Note that $2 must not be zero -->
      <multiply assign_to="1" variable="2" />
   </action>
</nop>
```

3.6.3.6. String Variables（字符串变量）

用<assignstr>命令来创建字符串变量，字符串变量接受两个参数：assign_to 和 value。消息中的值可能包含一些被其它相同的替换值，如下：

例如：

```
</nop>
   <action>
       <!-- Assign the value in field0 of the CSV file to a $1. -->
       <assignstr assign_to="1" value="[field0]" />
   </action>
</nop>
```

用<strcmp>可以比较一个字符器变量和一个值，结果是双精度型，比较的结果是小

于、等于、或大于 0 的值。参数有 assign_to、variable 和 value。例如：

```
<nop>
  <action>
    <!-- Compare the value of $strvar to "Hello" and assign it to $result.. -->
    <strcmp assign_to="result" variable="strvar" value="Hello" />
  </action>
</nop>
```

3.6.3.7. Variable Testing（变量测试）

变量测试允许你调用变量来构造循环和控制流程。test操作携带四个参数：variable是和value比较的一个变量，assign_to调用存储测试结果的变量，是一个布尔型。比较可以是以下测试中的一个：equal,not_equal、 greater_than、less_than、greater_than_equal、或less_than_equal。

如果$1小于10时，设置$2为true的例子：

```
<nop>
  <action>
    <test assign_to="2" variable="1" compare="less_than" value="10" />
  </action>
</nop>
```

3.6.3.8. lookup（查找）

lookup 操作用于注入索引文件（查看 indexed injection files）。lookup 操作携带的字段和关键字作为输入，产生的整型行号作为输入。例如，下面的操作是从认

证的消息头中提取用户名，然后在 user.csv 中找到相应的行。

```
<recv request="REGISTER">

  <action>

    <ereg regexp="Digest .*username=\"([^\"]*)\"" search_in="hdr"
    header="Authorization:" assign_to="junk,username" />

    <lookup assign_to="line" file="users.csv" key="[$username]" />

  </action>

</nop>
```

3.6.3.9. Updating In-Memory Injection files（更新内存的字段）

注入字段，特别是当一个已定义的索引，可以为你的 SIPp 场景储存一个内存中的数据。<insert> 和 <replace>操作提供一种方法以编程方式更新当前版本的内存中注射文件的版本（目前没有办法更新基于硬盘盘的版本）。插入操作携带两个参数：**file** 和 **value**，替换操作携带额外的行值。例如：插入一的新行可以这样实现：

```
<nop display="Insert User">

  <action>

    <insert file="usersdb.conf" value="[$user];[$calltype]" />

  </action>

</nop>
```

替换一行是相似的，但必须指定行号。你可能想用查找操作来获得替换的行号，如下操作：

```
<nop display="Update User">

  <action>

    <lookup assign_to="index" file="usersdb.conf" key="[$user]" />

    <!-- Note: This assumes that the lookup always succeeds. -->

    <replace file="usersdb.conf" line="[$index]" value="[$user];[$calltype]"
    />

  </action>

</nop>
```

3.6.3.10. Jumping to an Index（跳转到一个索引）

用<jump>操作能跳转到任意场景索引处。这可以用来创建基本的子例程。主叫能用[msg_index]保存他们的索引。被叫用这个操作跳转到相同的地方。如果在场景中有一个名为"_unexp.main"的标签时，SIPp 无论何时都会跳转到意外消息的标签处，并在名为"_unexp.retaddr"的变量中存放原先的地址。

下面的例子是跳转到索引 5：

```
<nop>

  <action>

  <jump value="5" />

  </action>

</nop>
```

跳转到索引中，包含变量名为_unexp.retadd：

```
<nop>
  <action>
    <jump variable="_unexp.retaddr" />
  </action>
</nop>
```

3.6.3.11. gettimeofday

gettimeofday 操作允许你用秒和毫秒得到从新纪元时间当前的时间，例如：

```
<nop>
  <action>
      <gettimeofday assign_to="seconds,microseconds" />
  </action>
</nop>
```

3.6.3.12. setdest

setdest 操作允许您更改调用的远程服务器。参数是协议、主机和调用的端口。在 SIPp 的设计存在一些局限性：你不能为一个呼叫改变协议；如果你用 TCP 模式，必须选择支持的 multi-socket（例如：必须指定-t tn）。也就是说，频繁使用 setdest 可能由于块操作而减少 SIPp 名称解析能力（这样可能会拖延 SIPp 查找主机名的时间）。这个例子用[next_url]关键字来连接指定的值。

```
<nop>

  <action>

    <assignstr assign_to="url" value="[next_url]" />

    <ereg   regexp="sip:.*@([0-9A-Za-z\.]+):([0-9]+);transport=([A-Z]+)"
    search_in="var" check_it="true" assign_to="dummy,host,port,transport"
    variable="url" />

    <setdest host="[$host]" port="[$port]" protocol="[$transport]" />

  </action>

</nop>
```

警告：

如果在 IPv6 下用 setdest，必须不使用方括号括起地址。对 SIPp 来说有特殊的含义，它将尝试作为变量解析你的 IPv6 地址。

分别指定端口后,就没有必要指定方括号了。

3.6.3.13. verifyauth

verifyauth 操作检查传入消息的授权头，主要针对用户名和密码，检查的结果用一个布尔变量存放。这样就可以模仿请求认证的服务端。目前仅支持简单的 MD5.以前用 **verifyauth** 操作，你必须发送一个消息，例如：

```
<recv request="REGISTER" />

<send><![CDATA[

  SIP/2.0 401 Authorization Required

  [last_Via:]
```

```
   [last_To:];tag=[pid]SIPpTag01[call_number]

   [last_Call-ID:]

   [last_CSeq:]

   Contact: <sip:[local_ip]:[local_port];transport=[transport]>

   WWW-Authenticate:      Digest      realm="test.example.com",
   nonce="47ebe028cda119c35d4877b383027d28da013815"

   Content-Length: [len]

]]>

</send>
```

收到第二次请求之后，你能提取提供的用户名，并作出比较，针对提供的用户名和密码的列表作为注入文件，并根据结果采取适当的操作：

```
<recv request="REGISTER" />

  <action>

     <ereg regexp="Digest .*username=\"([^\"]*)\"" search_in="hdr"
     header="Authorization:" assign_to="junk,username" />

     <lookup assign_to="line" file="users.conf" key="[$username]" />

     <verifyauth      assign_to="authvalid"      username="[field0
     line=\"[$line]\"]" password="[field3 line=\"[$line]\"]" />

  </action>

</recv>

<nop hide="true" test="authvalid" next="goodauth" />

<nop hide="true" next="badauth" />
```

3.6.4. Variables（变量）

由于复杂的场景，你需要存储信息位，用于消息或呼叫中。就像其它的编程语言，定义的 XML 场景允许你用变量。通过字母或数字名引用变量。在以前的版本里，变量名仅为数值型；在这个文档和默认的场景中，虽然创建新场景时，鼓励你去给变量分配有意义的名字，但你还是会看到很多"1"，"2"等等的变量。

除了名称外，SIPp 的变量也是一种宽松的类型。并不明确的声明变量的类型，而是从操作推断来设置。有四种类型的变量：字符串、正则表达式匹配、双精度型和布尔型。所有的运算双精度型的。**<test>** 和 **<verifyauth>**操作生成布尔值。字符器变量与正则表达式匹配相似。可以用正则表达式变量来代替调用的字符器变量。主要不同在于 **test** 属性（查看 Conditional Branching）有关。如果定义了一个字符串，test 为 true，不管怎样，设置正则表达式的变量 test 必须为真。可以用 <assignstr>操作把值转换成字符串。用<todouble>操作把值转换成双精度型。

变量也有一个范围，这是全局中所有的呼叫、每个用户或者默认每个呼叫中一个。也可以用一个全局变量。 例如存储场景配置参数或者保存一个全局计数器。绑定 -users 选项的用户变量允许你保存每一个用户呼叫的状态（例如：如果用户已经注册）。最后，默认情况下，将每一个呼叫变量从一个 SIP 消息复制到下一条消息或者控制分支有帮助的。变量可以用下面的语法声明为全局或每一个用户：

```
<Global variables="foo,bar" />

<User variables="baz,quux" />
```

没有必要声明局部变量。为了预防程序设计错误，在场景中，SIPp 执行基本的检查，以确保多次使用每个变量（这有助于防止一些错误变成很难调试的错误），不幸的是，这样携带正则表达式时会导致一些并发症。正则表达式的操作必须将整个匹配表达式分配给一个变量。如果你只是对检查表达式的有效性感兴趣（例如：设置了 check_it 属性）或者捕获子表达式，你必须仍然将整个匹配表达式分配给一个变量，这个变量可能只引用一次，你必须告诉 SIPp，用一个参考有意的使用一次这个变量。例如：

```
<recv request="INVITE">
```

```
   <action>

      <ereg  regexp="<sip:([^;@]*)"  search_in="hdr"  header="To:"
      assign_to="dummy,uri" />

   </action>

</recv>

<Reference variables="dummy" />
```

3.6.5. Injecting values from an external CSV during calls（在呼叫过程中从外部CSV文件中插入值）

可以用"-inf file_name"作为一个命令行参数把文件的值插入到场景中。文件的第一行应当说明是使用顺序(SEQUENTIAL)命令还是随机(RANDOM)命令。每一行相当于一个呼叫，每一行或多行用';'分隔数据段，他们在XML场景文件中被称为[field0], [field1], ...。例如：

```
SEQUENTIAL

#This line will be ignored

Sarah;sipphone32

Bob;sipphone12

#This line too

Fred;sipphone94
```

文件会按照顺序执行（第一个呼叫用第一行，第二个呼叫用第二行）。关键字"[field0]"出现在场景的任何地方，"Sarah", "Bob" 或"Fred"会替换"[field0]"。关键字"[field1]"出现在场景的任何地方，"sipphone32"或"sipphone12"或"sipphone94"会替换"[field1]"。在文件的末尾,SIPp将从一开始就再次呼叫。文件的大小没有限制。

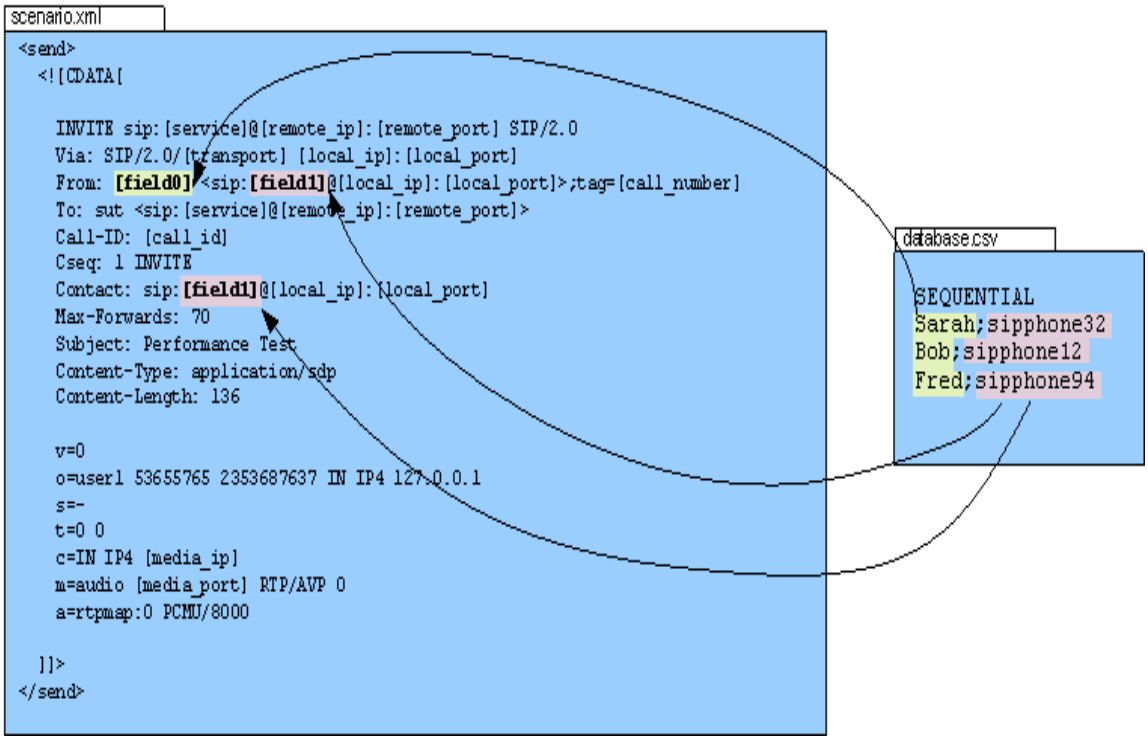
您可以用可选的行参数覆盖默认的行，例如：

```
[field0 line=1]
```

在文件中选择第二行（第一行是0行。行参数支持关键字，因此与lookup操作一直使用，可以基于一个关键字来选择值。）

CSV文件可以包含注释行。一个注释行以“#”开头。

一个图片胜过1000句话，如下图：



难以想象，这个功能真是太强大了。

有时个可能用多个注入文件。当用不同的方法时会用到不同类型的数据。例如,当运行基于用户的脚本时，可以将”USER”放在caller.csv的第一行，将“RANDOM”放在callee.csv的第一行。为了区别使用哪一个CSV文件，加上关键字file=参数，例如：

```
INVITE sip:[field0 file="callee.csv"] SIP/2.0

From: sipp user <[field0 file="caller.csv"]>;tag=[pid]SIPpTag00[call_number]

To: sut user <[field0 file="callee.csv"]>

...
```

会从callee.csv选择目标user并发送出去。如果没有指定参数，默认用命令行中的第一个输入文件。

3.6.5.1. PRINTF Injection files（PRINTF注入文件）

一个标准的扩展注入文件是"PRINTF"文件。通常一个输入文件像这样有一个重复的特征：

```
USERS

user000;password000

user001;password001

...

user999;password999
```

SIPp必须在内存中维护这种结构，这样由于在调用非常大的注入文件时会减少机器性能。为了消除这个问题，SIPp可以自动生成这样基于一个或多个模板行结构化文件。例如：

```
USERS,PRINTF=999

user%03d;password%03d
```

与原来的例子有相同的逻辑,SIPp仅在内存里存一个条目就可以了。每次使用一行；SIPp将用请求的行号(从0行开始)取代%d。可以用标准输出函数格式十进制说明符。当使用多个模板时，会循环的使用它们，例如：

```
USERS,PRINTF=4

user%03d;password%03d;Foo

user%03d;password%03d;Bar
```

等同于下面的注入文件：

```
USERS

user000;password000;Foo

user001;password001;Bar

user002;password002;Foo

user003;password003;Bar
```

使用下面的参数来控制打印输出注入文件的操作：

参数	描述	例子
PRINTF	在文件中有多少虚拟线	PRINTF=10，创建10条虚拟线
PRINTFMULTIPLE	设置行增长的步长	PRINTF=10,PRINTFMULTIPLE=2，创建10条虚拟线，编号为0，2，4…18。
PRINTFOFFSET	设置初始值	PRINTF=10,PRINTFOFFSET=100，创建号码为100至109的虚拟线。 PRINTF=10,PRINTFMULTIPLE=2,PRINTFOFFSET=10 创建10条虚拟线，编号为10，12，14，….28。

表1：输出函数参数

3.6.5.2. Indexing Injection files（为注入文件加入索引）

-infindex 选项为注入文件生成一个索引。-infindex 参数为注入文件的字段号增加索引，例如，如果你有一个包含用户名和密码的注入文件(例如)：

```
USERS
alice,pass_A
bob,pass_B
carol,pass_C
```

你可以在文件中用选定的用户提取相应的密码。为了高效的达到此功能，SIPp必须为第一个字段（0）创建一个索引。这样你会传递-infindex users.csv 0参数（假设文件名为user.csv）。SIPp将包含的逻辑{"alice" => 0, "bob" => 1,"carol" => 2}创建一个索引条目，为的提取指定的密码，用lookup操作把行号存到一个变量（如

$line）然后用关键字[field1 line="[$line]"]。

3.6.6. Conditional branching（条件分支）

3.6.6.1. Conditional branching in scenarios（场景中条件分支）

可以以非线性的方式执行一个场景，你可以从一个场景的一部分,跳到另一个部分，例如当收到一条消息或设置了一个呼叫变量。

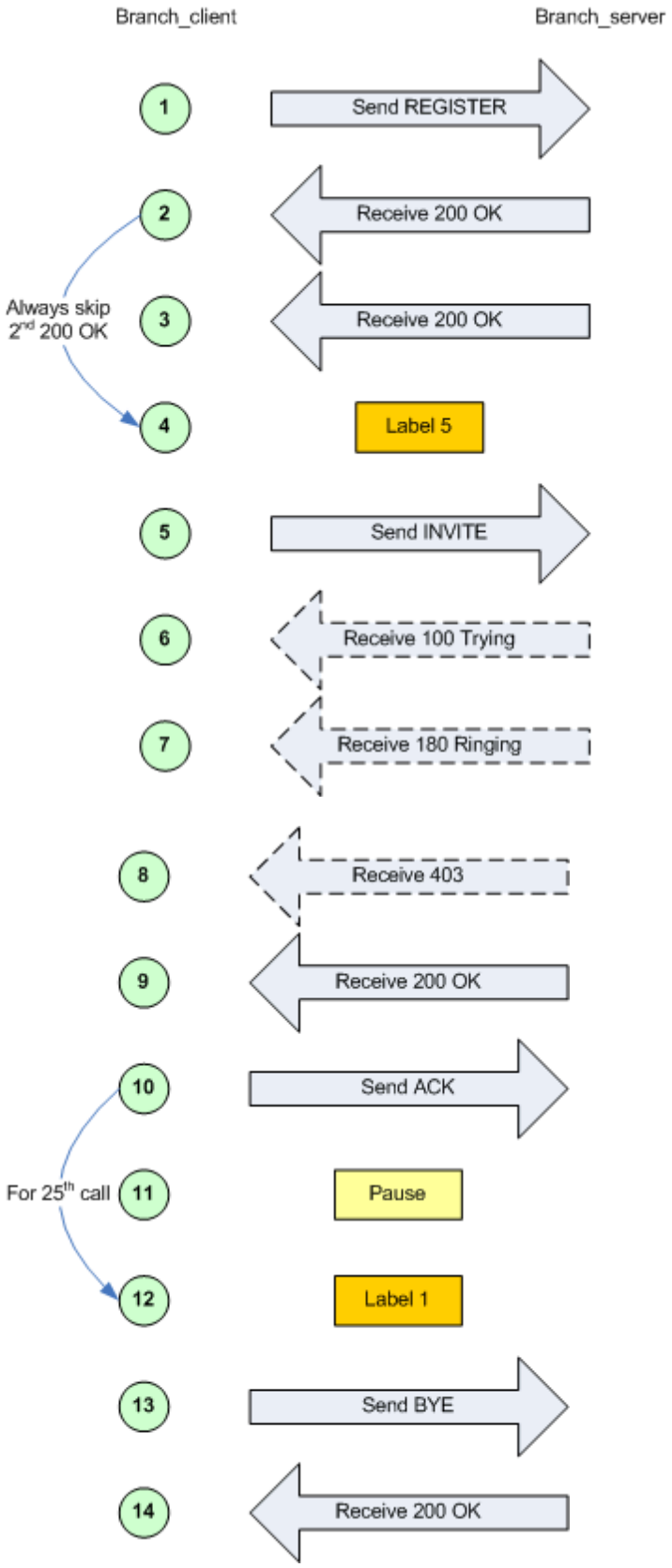
在XML文件中定义一个标签<label id="n"/>，n是1到19之间的任何数字（如果有必要，我们可以设置更多），用标签命令可以去主场景的任何地方，你可以在任何操作(send, receive, pause等等)之前添加next="n"参数，n匹配标签的id，一旦执行next="n"命令,将会从标签处的地方继续执行场景。这项功能用于收到可选的消息，如403消息，根据不同的脚本回复不同的消息。

作为一种选择，如果你添加test="m"参数到next,它会跳到设置变量为[$m]的标签处，这个允许你在收到的包和改变流或在以后的脚本查找一些字符串。测试变化的评估基于呼叫变量的类型。对于正则表达式来说，至少匹配一项；对于布尔变量来说，值必须为true；对其它来说，必须设定一个值（当前仅是双精度型）。对于更复杂的测试，查看<test> 操作

> 警告：
>
> 如果在脚本末尾添加特例，不要忘记在实例中放一个标签，跳到正常的流程中。

例如：

下面的例子对应嵌入“branchc”(客户端)的场景。它必须运行在嵌入式“branchs”(服务器端)的场景。



3.6.6.2. 随机性的条件分机（Randomness in conditional branching）

SIPp更像一个“普通”的SIP客户端为人类所用，可以使用“统计分支”（"statistical branching"）。无论在什么地方你可以在设置一个变量（test=”4”）地方有一个条件分支，你也能用属性“chance”（例如：chance=”0.90”）统计决定分支。Chance有0（从不）和1（一直）中的一个值。"test"和"chance"能捆邦使用，例如，仅当测试成功且chance是良好的情况下才能分支（原文：i.e. only branching when the test succeeds and the chance is good.）

利用这个特点，你能在给定的场景中用一个变量重新执行（例如：应答一个呼叫或因线路忙拒接1路通话），或者循环运行（如注册）且迭代一些随机号码后跳出循环。

3.6.7. SIP认证（SIP authentication）

SIPp支持sip认证。支持两种认证算法：Digest/MD5 ("algorithm="MD5"") 和 Digest/AKA ("algorithm="AKAv1-MD5"", IMS通过3GPP规定其用法).

> **注意：**
>
> 为了启用认证支持，必须用指定的方法编译SIPp，参考SIPp installation获得更多信息

启用认证很简单，当收到401（未认证）或者一个407（请求代理认证），你必须在<recv>命令添加 auth=”true”，去挑战帐号，这时，在下一条消息中通过用[authentication]关键字认证头消息能被重新拒绝。

通过[authentication]关键字的使用来处理认证头消息。依据算法("MD5" or "AKAv1-MD5")，不同的参数必须挨着authentication关键字：

- Digest/MD5 （例如：[authentication username=joe password=schmo]）
  - **username**:名称：如果没有指定用户名， 用’-s’ (service)命令行参数中携带用户名。

■ **passwlrd**:密码：如果没有指定密码，用'-ap'(认证密码)命令行参数中携带密码

- Digest/AKA(例如：[authentication username=HappyFeet aka_OP=0xCDC202D5123E20F62B6D676AC72CB318 aka_K=0x465B5CE8B199B49FAA5F0A2EE238A6BC aka_AMF=0xB9B9])

■ **username**:用户名：如果没有指定用户名，用'-s' (service)命令行参数中携带用户名。

■ **aka_K**: 永久加密关键字，如果没有提供aka_K，用"password"属性作为aka_K

■ **aka_OP**:操作变量关键字；

■ **aka_AMF**:认证管理字段（在使用中声明算法和密钥）

每一个呼叫你想用一个带有不同的username/password或aka_K的认证，你可以这样做：

- 做一个CSV文件，如下:

```
SEQUENTIAL
User0001;[authentication username=joe password=schmo]
User0002;[authentication username=john password=smith]
User0003;[authentication username=betty password=boop]
```

- XML文件如下：（授权字符串作为一个新的关键字去替换[field1]字段）：

```
<send retrans="500">
```

```
    <![CDATA[
       REGISTER sip:[remote_ip] SIP/2.0
       Via: SIP/2.0/[transport] [local_ip]:[local_port]
       To: <sip:[field0]@sip.com:[remote_port]>
       From: <sip:[field0]@[remote_ip]:[remote_port]>
       Contact: <sip:[field0]@[local_ip]:[local_port]>;transport=[transport]
       [field1]
       Expires: 300
       Call-ID: [call_id]
       CSeq: 2 REGISTER
       Content-Length: 0
    ]]>
</send>
```

例子：

```
<recv response="407" auth="true">
</recv>
<send>
   <![CDATA[
      ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0
      Via: SIP/2.0/[transport] [local_ip]:[local_port]
      From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]
```

```
      To: sut <sip:[service]@[remote_ip]:[remote_port]>[peer_tag_param]

      Call-ID: [call_id]

      CSeq: 1 ACK

      Contact: sip:sipp@[local_ip]:[local_port]

      Max-Forwards: 70

      Subject: Performance Test

      Content-Length: 0

   ]]>

</send>

<send retrans="500">

   <![CDATA[

      INVITE sip:[service]@[remote_ip]:[remote_port] SIP/2.0

      Via: SIP/2.0/[transport] [local_ip]:[local_port]

      From: sipp <sip:sipp@[local_ip]:[local_port]>;tag=[call_number]

      To: sut <sip:[service]@[remote_ip]:[remote_port]>

      Call-ID: [call_id]

      CSeq: 2 INVITE

      Contact: sip:sipp@[local_ip]:[local_port]

      [authentication username=foouser]

      Max-Forwards: 70

      Subject: Performance Test

      Content-Type: application/sdp

      Content-Length: [len]

      v=0
```

```
      o=user1 53655765 2353687637 IN IP[local_ip_type] [local_ip]

      s=-

      t=0 0

      c=IN IP[media_ip_type] [media_ip]

      m=audio [media_port] RTP/AVP 0

      a=rtpmap:0 PCMU/8000

   ]]>

</send>
```

3.6.8. Initialization Stanza

在启动SIPp时，一些复杂的场景要求设置的全局变量。initialization stanza可以实现这项功能。为了创建一个initialization stanza，用带有<init>和</init>.包围<nop>和<label>命令，<nop>在启动SIPp的时候执行一次。除全局变量外，不用于呼叫，举个例子，如果$THINKTIME没有设置值，就把它设置为1.（用-set 命令行参数）

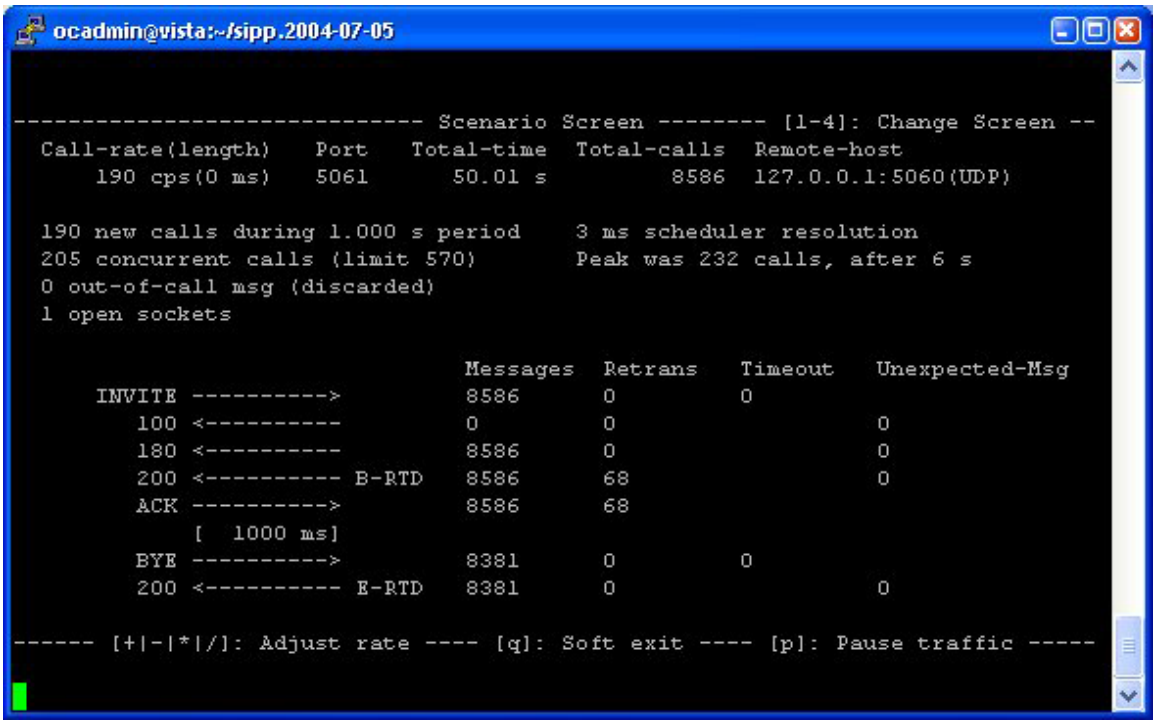
```
<init>

<!-- By Default THINKTIME is true. -->

    <nop>

        <action>

            <strcmp assign_to="empty" variable="THINKTIME" value="" />

            <test assign_to="empty" compare="equal" variable="empty" value="0"
            />

        </action>

    </nop>
```

```
<nop condexec="empty">

    <action>

        <assignstr assign_to="THINKTIME" value="1" />

    </action>

</nop>

</init>
```

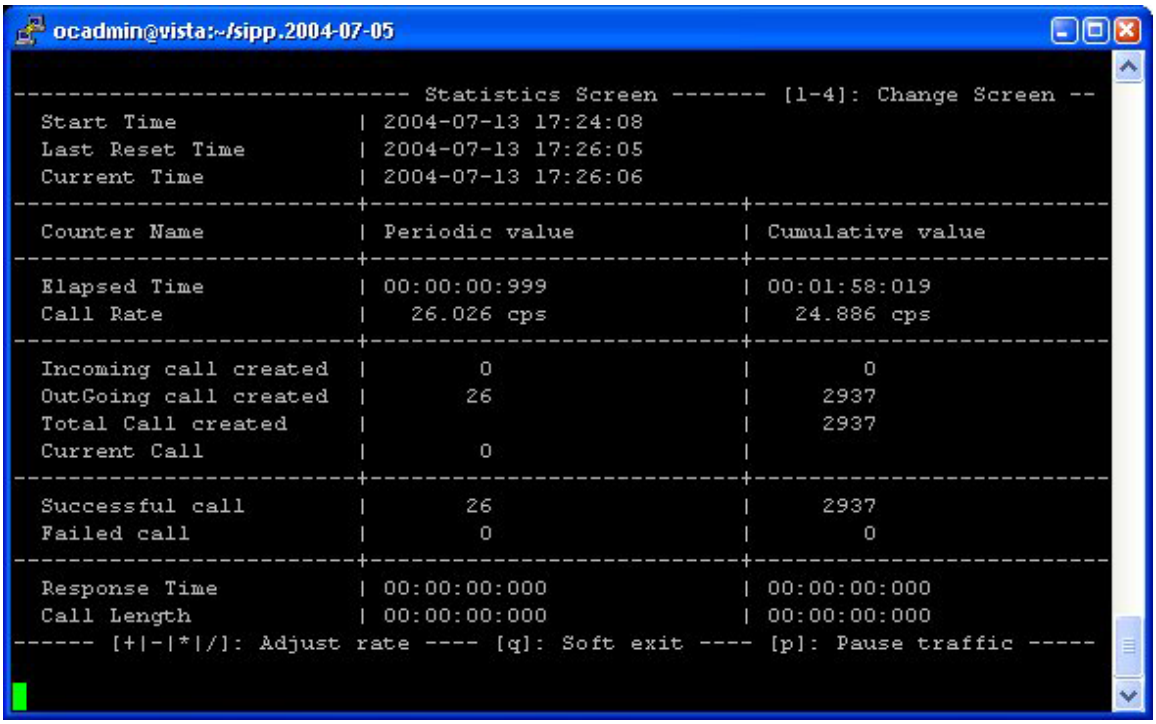
3.7. Screens（屏幕显示）

有几个屏幕显示监控的SIPp事件。你可以按键盘上的1到9数字键来观看不同的屏幕显示参数。

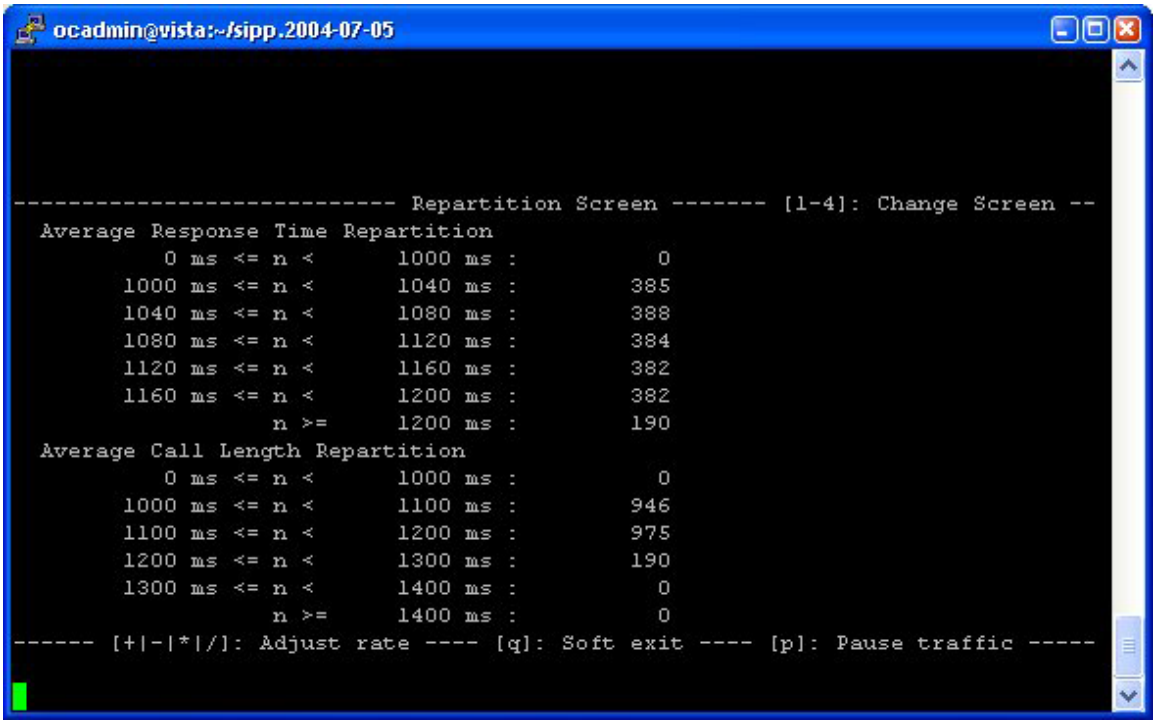
- 数字’1’键：场景屏幕。一个场景的呼叫流显示一些重要的信息。



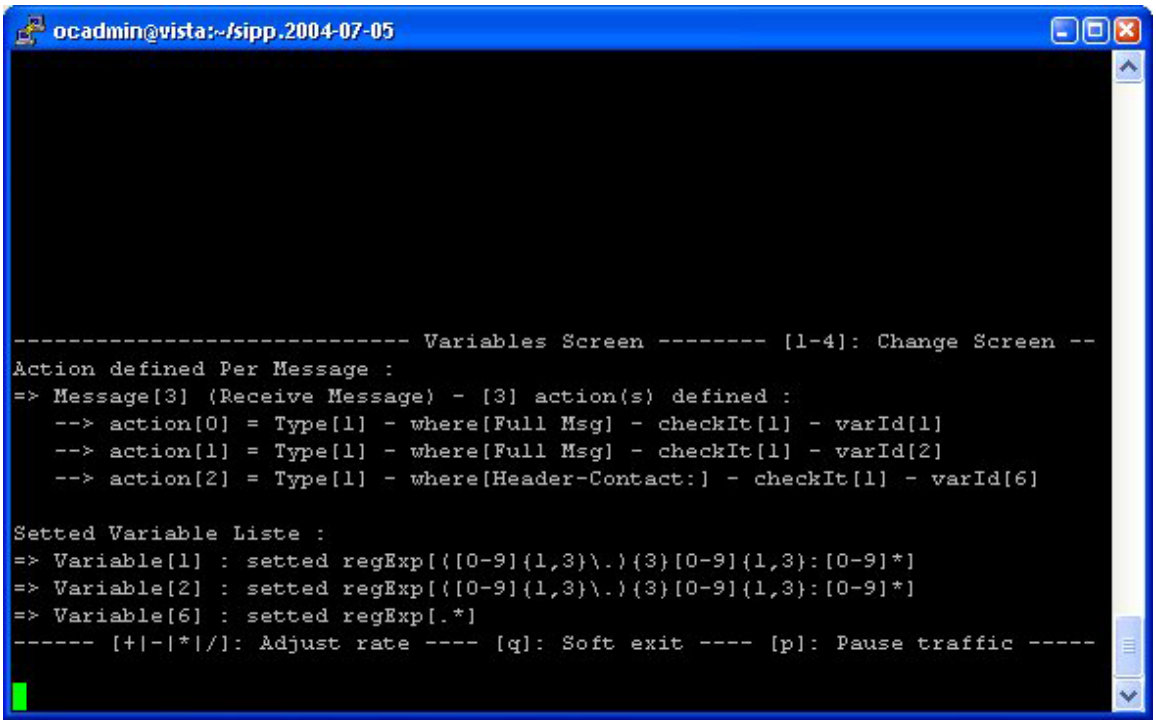
- 数字’2’键：统计屏幕。显示主要的统计数量。"Cumulative"列收集所有的统计。启动SIPp后，"Periodic"列给定了周期统计的值。(在命令行中通过**-f frequency**指定)



- 数字键‘3’：再分配屏幕。显示响应时间和呼叫长度的分布。在场景中指定：



- 数字键’4’:变量屏幕。显示在场景执行呼叫中操作的信息，与场景变量信息相似。



3.8. Transport modes（传输模式）

SIPp有几种传输模式，默认为“UDP 单通道socket”（"UDP mono socket"）。

3.8.1. UDP mono socket（UDP 单通道socket）

在UDP mono socket模式中（-t u1 命令行参数），在SIPp与远程控制中一个呼叫打开一个IP/UDP socket。

这种模式是通常用在模仿用户代理呼叫一个SIP服务器。

3.8.2. UDP multi socket

在UDP multi socket模式中（-t un 命令行参数），在SIPp与远程控制中一个呼叫打开一个IP/UDP socket。

这种模式是通常用在模仿用户代理呼叫一个SIP服务器。

3.8.3. UDP with one socket per IP address（UDP模式中一个IP地址带有一个socket）

在UDP模式中一个IP地址带有一个socket（用-t ui 命令行参数），在inf file中，一个IP地址打开一个IP/UDPsocket。

除了"-t ui"命令行参数，在inf file中为本地IP地址给定的呼叫必须指定命令行参数字段。用"-ip_field <nb>"提供字段号。

用这个功能有两个明显的情况：

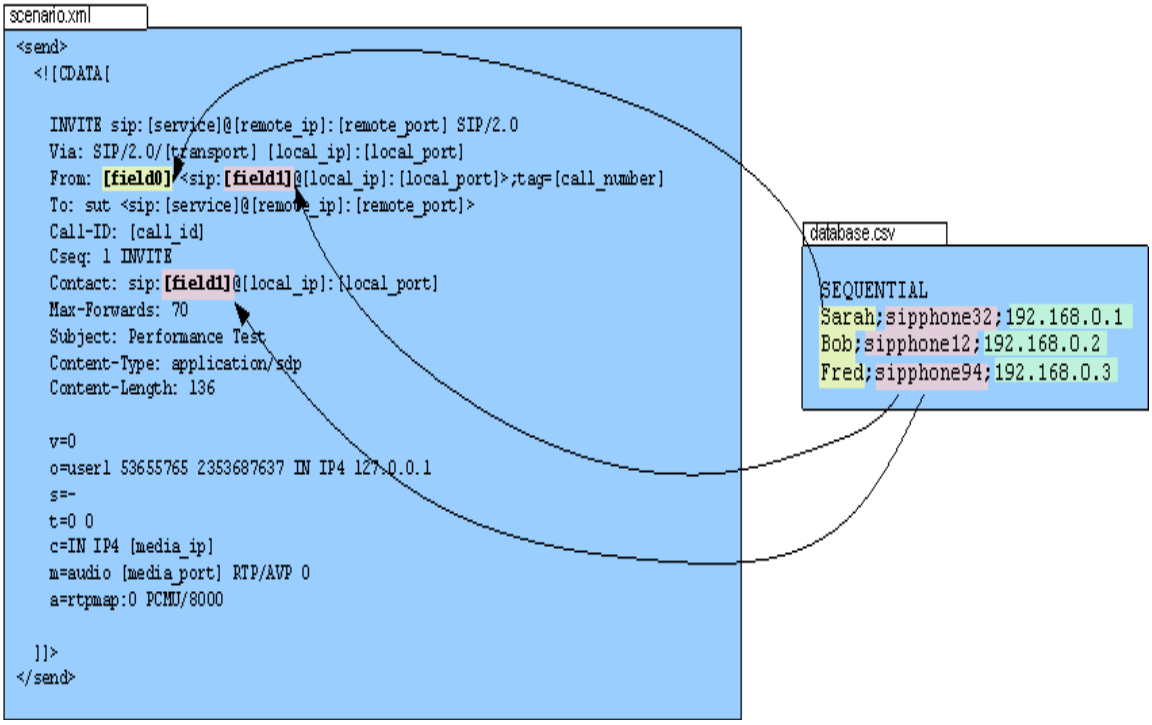
- 客户端：当客户端用“-t ui”，SIPp用一个不同的地址创建一个呼叫，如同在inf 文件中提供的一样，在这种情况下，当你的IP地址注入X字段中，那么在你的UAC XML场景文件中用[fieldX]代替[local_ip]。

- 服务端：当服务端用“-t ui”，SIPp在inf文件中绑定自己，用0.0.0.0代替至所有的IP地址列表。SIPp应答相同IP的请求产生影响，为了适当的Contact和Via字段，用[server_ip]关键字和收到的请求中提供的IP地址，因此当用到的时候，在收到的请求中通过[server_ip]提供的IP地址，在UAS XML场景中你替换[local_ip]。

在接着的图解中，一个客户端场景中的命令行看起来像：

```
./sipp -sf myscenario.xml -t ui -inf database.csv  -ip_field 2 192.168.1.1
```

通过这样做，每一个新的呼叫会循序的来自IP 192.168.0.1, 192.168.0.2, 192.168.0.3, 192.168.0.1, ...



这种模式一般用于模拟用户代理,用于在IP地址中一个用户代理呼叫一个SIP服务端。

3.8.4. TCP 单socket（ TCP mono socket）

在TCP单通道socket模式中（用(-t t1命令行参数)，SIPp与远程控制端打开一个IP/TCPsocket。这个socket存放所有的呼叫。这种模式一般用于模拟两个SIP服务器之间的关系时使用。

3.8.5. TCP multi socket（ TCP多socket）

在TCP多socket模式（用命令行参数 -t tn），SIPp与远程控制端之间为每一个呼叫打开1个IP/TCPsocket。这种模式一般用于模拟用户代理呼叫SIP服务器时使用。

3.8.6. TCP reconnections（TCP重连）

SIPp支持TCP重连。万一TCPsocket丢失，SIPp会重新连接。用以下参数在命令行中控制这种行为：

- -max_reconnect: 设置重新连接的次数

- -reconnect_close true/false:重新连接是否关闭？
- -reconnect_sleep int: 多久进行一次重新连接？

3.8.7. TLS mono socket（TLS单socket）

在TLS单socket模式（用命令行参数 -t l1表示）中，SIPp与远程控制端之间打开一个安全TLS（Transport Layer Security）socket。这个socket存放所有的呼叫。

这种模式一般用于模拟 两个SIP服务器之间的关系时使用。

> 警告：
>
> 当用TLS传输时，在当前目录中SIPp期望两个字段，一个certificate (cacert.pem) 和一个 key (cakey.pem)。如果用密码保护，SIPp会邀请它。

SIPp支持X509's CRL(证书撤回列表)。如果指定命令行-tls_crl读取CRL文件，CRL会被读取并利用。

3.8.8. TLS multi socket（TLS多socket）

TLS多socket模式（用命令行参数 -t ln表示），SIPp与远程远程控制端之间为为每一个新的打开一个安全的TLS (Transport Layer Security)socket。

这种模式一般用于模拟用户代理呼叫一个SIP服务器。

3.8.9. SCTP mono socket（SCTP单socket）

在SCTP mono socket模式中（用-t s1命令行参数),在SIPp与远端打开一个SCTP（流传输控制协议）socket。所有的呼叫用这个socket放置。

这种模式一般用于模仿两个SIP服务器之间的联系。

-multihome, -heartbeat, -assocmaxret, -pathmaxret, -pmtu和-gracefulclose命令行参数允许控制SCTP协议的指定功能，通常没有必要这样做。

3.8.10. SCTP multi socket

在SCTP multi socket 模式中（用 -t sn命令行参数），在SIPp和远端为每一个新的呼叫打开一个SCTP。这种模式一般用于模仿用户代理呼叫一个SIP服务器。

3.8.11. IPv6 support (支持Ipv6)

SIPp支持Ipv6。当用IPv6时，只需指定本机IP地址（用命令行参数 -i表示）是一个IPv6地址即可。

以下例子启动一个UAS服务端监听端口5063和UAC客户端发送IPv6至5063端口。

```
./sipp -sn uas -i [fe80::204:75ff:fe4d:19d9] -p 5063

./sipp -sn uac -i [fe80::204:75ff:fe4d:19d9] [fe80::204:75ff:fe4d:19d9]:5063
```

警告:

当前在IPv6下可能不支持PCAP播放功能

3.8.12. Multi-socket limit (多socket限制)

当用"multi-socket"其中一个传输时，可以在系统（查看how to increase file descriptors section 去修改这些限制）中打开（对应同时呼叫的数量）最大socket数量。你可以用命令行选项 -max_socket 来限制socket的数量。一旦达到打开socket的最大数量，通信会结束已经打开分发的socket。

3.9. Handling media with SIPp（用SIPp处理媒体流）

SIPp可以认为是一个信号通信生成器，有限的媒体流支持（RTP）.

3.9.1. RTP echo (RTP回送)

"RTP echo"功能允许SIPp监控RTP媒体流中的一个或两个本地IP地址或端口（用命令行参数-mi和-mp指定）。

在这个地址/端口中收到的所有内容都会回送给发送端。

来自port+2的RTP/UDP包也回送给发送端（用于语音与视频回声）。

3.9.2. RTP streaming

SIPp可以用RTP.播放PCMA, PCMU or G729编码的音频文件。

在action reference section中可以找到更多详细的内容。

3.9.3. PCAP Play（PCAP播放）

PCAP播放功能用PCAP library(http://www.tcpdump.org/pcap3_man.html)的功能去重播预录制的RTP流发给目标。用Wireshark(http://www.wireshark.org/)或tcpdump (http://www.tcpdump.org/)去录制RTP流。PCAP播放允许你：

- 播放任何RTP流（语音、视频、语音+视频，除此之外还有DTMFs/RFC 2833, T38 fax, ...）
- SIPp不需要任何编解码器；
- 精确的模拟任何SIP设备的行为，pcap播放会尝试播放录制的RTP流（受系统性能的限制）
- 准确的复现IP嗅探器捕获的内容，像Wireshark (http://www.wireshark.org/)

一个好的例子是带有媒体的UAC (uac_pcap)场景.

SIPp中可以用G711 alaw 预录制的pcap文件和在pcap/目录绑定(RFC 2833) DTMFs。

> 警告：
>
> PCAP播放功能从多线程库中用多线程设置调试参数。依赖于系统设置，你也许需要用root来设置，请检查"man 3 pthread_setschedparam"手册获得更多信息。

PCAP播放操作的更多信息请参考action reference section。

可以在SIPp wiki (http://sipp.sourceforge.net/wiki/index.php/Pcapplay)发现这项功能最新信息的提示和技巧

3.10. Exit codes（退出代码）

为了减轻测试的自动化，当出现致命错误或当到达了请求的呼叫数量时，SIPp用以下退出代码退出：

- 0：所有呼叫全部成功
- 1：至少一个呼叫失败
- 97：内部命令退出。呼叫可能被处理了；
- 99：没有呼叫处理的正常退出
- -1：致命错误

依赖于SIPp运行的系统，你能用“echo ?”命令打印出这个退出代码。

3.11. Statistics（统计）

3.11.1. Response times（响应时间）

SIPp能收集和记录响应时间。响应时间名称可以是任意的字符串，而是为了向后兼容,如果被命名为“1”被看作是“true”值，在两个SIPp命令(send, recv or nop)之间会有捕获每一次响应时间。你能用start_rtd属性启动一个计时器，用rtd属性停止。

在SIPp接口中通过按3、6、7、8、9，你能查看那些计时器的值。你也能用-trace_stat选项把这些值保存在一个CSV文件中（参考下面）。

如果设置了-trace_rtt选项，会把响应时间也存放在>scenario file name<_>pid<_rtt.csv中。

每一行代表一个RTD计量（通过接收rtd="n"属性触发RTD）,通过-rtt_freq参数调整存放的频率。

3.11.2. Available counters（有效计数器）

-trace_stat选项保存所有的统计信息至scenario_name_pid.csv文件中，所有带有计数器的一个消息头行启动时，频繁地（(-fd选项）将下面所有消息行的统计计数

器的’快照’整理成统计报告。当SIPp退出时，最近统计的值也保存在这个文件中。这个文件能被轻易的导入至像Excel的任何电子数据表中。

在计数器名称中，

（P）代表“周期”---从上一次统计之后。

（C）代表“累计”---从启动SIPp之后。

可用的统计有：

- StartTime:启动SIPp时的日期和时间
- LastResetTime:周期性计数器，最近一次清零的日期和时间
- CurrentTime:统计行的日期和时间
- ElapsedTime:测试时长
- CallRate:呼叫速率（每秒的呼叫数量）
- IncomingCall:呼入的数量
- OutgoingCall:呼出的数量
- TotalCallCreated：创建呼叫的数量
- CurrentCall:当前正在呼叫的数量
- SuccessfulCall:成功呼叫的数量
- FailedCall:失败呼叫的数量（包括所有的原因引起的）
- FailedCannotSendMessage:由于SIPp不能发送消息时失败呼叫的数量
- FailedMaxUDPRetrans:由于已经达到了UDP试图重发的最大数量而引起呼叫失败的数量
- FailedUnexpectedMessage:由于在场景中收到的不是期望的SIP消息而引起呼叫失败的数量
- FailedCallRejected:由于SIPp内部错误而引起呼叫失败的数量（一个场景中同步命令时没有识别或者一个场景操作失败或一个场景中变量分配失败）。

- FailedCmdNotSent:由于内部SIPp通信错误而引起呼叫失败的数量(场景发送同步命令失败)

- FailedRegexpDoesntMatch:正则表达式不匹配而引起呼叫失败的数量（也许在呼叫时几个正则表达式不匹配，但计数器会一个一个的递增）

- FailedRegexpHdrNotFound:由于带有hdr选项但不匹配消息头的正则表达式引起的呼叫失败的数量

- OutOfCallMsgs: 一个存在的呼叫不能收到的SIP消息关联的数量。

- AutoAnswered: 为新Call-ID收到未期望特定消息的数量。一般用200 OK会自动回应这条消息，仅仅为 了实现'PING'消息。

此外，还会收集另外两个统计数据:

- ResponseTime（查看以前的部分）

- CallLength:这是整个呼叫持续的时间

在场景中ResponseTime 和 CallLength统计可以用ResponseTimeRepartition 和 CallLengthRepartition命令调节。

在log stat文件中标准偏差（STDev）

也可以在这两个log stat文件中使用标准偏差(STDev)

3.11.3. Detailed Message Counts（详细的消息计算）

SIPp屏幕提供了消息的发送或接收、重传、消息丢失和意外消息的详细信息，在屏幕中分析远比在CSV文件分析简单得多。通过传递-trace_counts选项把主显示屏中的每一条信息存储至CSV文件里，文件中的每列代表一条消息（比如："1_INVITE_Sent"或"2_100_Unexp"），

每一行代表统计的间隔。

3.11.4. Importing statistics in spreadsheet applications（电子表格应用程序中导入数据）

3.11.4.1. Example: importation in Microsoft Excel（例子：在EXCEL中导入）

这里有一个视频（要求Windows Media Player 9解码或更高），在上面演示了怎样在Excel中导入CSV统计信息，且创建一个失败呼叫的图像。sipp-02.wmv

sipp-02.wmv

3.12. Error handling（错误处理）

SIPp拥有先进的特性来处理错误和意外事件。下面几节详细讨论。

3.12.1. Unexpected messages（意外消息）

- 当一个SIP消息与一个存在的呼叫有关（携带相同的Call-ID:头）但在场景中收到了不期望的消息，如果没有收到200 OK，SIPp会自动发送一个CANCEL消息或收到了200 OK，SIPp会自动发送一个BYE消息。呼叫也会标记为失败的呼叫。如果收到的意外消息为4XX or 5XX，SIPp会自动给这条消息回应ACK，关闭呼叫并标记此次呼叫为失败。

- 当一个SIP消息与一个收到存在的呼叫无关（携带不同的Call-ID:头），SIPp自动发送BYE消息，不会计算这个呼叫。

- 当收到一个SIP "PING" 消息，SIPp在响应的时候会自动发送一个ACK消息。这条消息作为一条意外的消息不计算在内。但是它计算在“AutoAnswered”统计计数器中。

- 收到的意外的消息且不是一个SIP消息，SIPp会简单的抛弃这条消息。

3.12.2. Retransmissions (UDP only)（重传（仅用于UDP））

只有在UDP传输模式中存在重传机制。为了激活重传机制，"send"命令必须包含"retrans"属性。

当发送SIP消息，收到的消息中没有收到ACK或其它响应时会激活重传机制，重新发送消息。

注意：

重传机制遵循RFC 3261,部分17.1.1.2。重传分为INVITE 和 non-INVITE 两种方式。

<send retrans="500">:将启动计数器T1为500毫秒重传。

即使在场景中指明了用retrans，你可以用-nr命令行选项来全部禁用重传机制。

3.12.3. Log files (error + log + screen)（日志文件）

有几种方法来追踪SIPp运行期间的情况：

- 你能用命令行参数 -trace_msg 生成 <name_of_the_scenario>_<pid>_messages.log来记录发送和接收SIP消息,记录的消息是时间戳形式的，这样你就可以方便追踪。

- 也可以用-trace_shortmsg参数来追踪消息，这个日志会把消息中最重要的值放入<scenario filename>_<pid>_shortmessages.log。

- 你能用命令行参数 -trace_err 生成 <name_of_the_scenario>_<pid>_errors.log 来追踪所有意外的消息和事件。

- 命令行参数-trace_error_codes 可以把意外消息的SIP 响应代码放入 <name_of_the_scenario>_<pid>_error_codes.log

- 用命令行参数-trace_counts主场景屏幕中消息数量放在 <name_of_the_scenario>_<pid>_counts.csv

- 用命令行参数-trace_calldebug把追踪的消息和呼叫失败的状态存放至 <name_of_the_scenario>_<pid>_calldebug.log，这个功能的用处在于比 -trace_msg耗费较少的资源，也允许你去调试没有完成的呼叫流。

- 你可以将统计屏幕的消息保存在一个文件中，和界面中显示的信息一样。当SIPp在背景模式下运行的时候特别有用。

- 有两种方法可以实现：

- 当SIPp退出时会得到一个最终状态报告（用 -trace_screen选项）

- 用USR2信号即刻生效(例如: kill -SIGUSR2 738)

- 按’s’键（如果设置了-trace_screen选项）

  如果设置了-trace_logs选项，你能用<log>操作把场景存在<scenario file name>_<pid>_logs.log文件里。查看Log action section

3.13. Online help (-h)（在线帮助）

在线帮助，为了方便，你可以复制下来，用-h选项显示以下信息：

参数	解释
用法: sipp remote_host[:remote_port] [options] 例子: 运行嵌入服务端(uas)场景: ./sipp -sn uas 在同一台主机中，运行客户端（uac）场景: ./sipp -sn uac 127.0.0.1 可用选项:	
*** 场景文件选项:	
-sd	启动默认场景
-sf	加载一个XML场景文件,为了学习更多的XML场景语法，用-sd 选项启用默认场景，它们包含所有必要的帮助

-oocsf	加载out-of-call场景
-oocsn	加载out-of-call场景
-sn	启动默认场景（嵌入在SIPp可执行文件中）。如果省略这个选项，会加载标准SipStone UAC场景。 这个版本中可用的值有： - 'uac' : 标准SipStone UAC(默认) - 'uas' : 简单UAS响应器 - 'regexp' :标准SipStone UAC---带有regexp和变量 - 'branchc' :在场景中分支和条件分支---客户端 - 'branchs' : 在场景中分支和条件分支---服务端 默认3pcc场景9（参考 -3pcc选项）： -'3pcc-C-A' : 控制器A端（必须在所有3pcc场景启动后再启动此项） - '3pcc-C-B' : 控制器B端 - '3pcc-A' : A 端 - '3pcc-B' : B 端.
***** IP, 端口和协议选项:**	
-t	设置传输模式： - u1:带有一个socket的UDP(默认) - un: UDP中每个呼叫用一个socket - ui: UDP中每个IP地址用一个socket.必须在文件中定义IP地址 - t1: TCP中用一个socket

	- tn: TCP中每个呼叫用一个socket - c1: u1 + 压缩（仅用于加载压缩插件时使用）， - cn: un + 压缩（仅用于加载压缩插件时使用）. SIPp不提供此插件。
-i	设置'Contact:','Via:','From:' 域的IP地址，默认为主机的主ip地址
-p	设置本地端口号，默认为随机端口号
-bind_local	绑定socket至本地IP地址，例如，用本地IP地址为源IP地址，如果SIPp运行在服务器下，它只会监听本地IP地址而不是所有IP地址
-ci	设置本机控制的IP地址
-cp	设置西南控制的端口号，默认为8888.
-max_socket	设置同时打开sockets的最大数。这个选项很有特点，如果每秒用一个socket，一旦达到限制，通信分布在已经打开的sockets上，默认值是50000。
-max_reconnect	设置重新连接的最大次数
-reconnect_close	在重连的时候关闭呼叫吗？
-reconnect_sleep	关闭和重连之间的时间间隔（用毫秒表示）
-rsa	为了发送消息设置远端发送地址到host:port
*** SIPp全部行为选项	
-v	显示版本和版权信息

-bg	后台运行模式
-nostdin	禁用stdin
-plugin	加载一个插件
-sleep	多长时间启动SIPp。默认单位是秒。
-skip_rlimit	不执行rlimit文件描述符限制的调优，默认：不开启
-buff_size buff_size	设置发送和接收缓冲区大小
-sendbuffer_warn	在SendBuffer失败时产生警告来代替错误
-lost	默认设置丢包数量（场景说明中可以重写这个值）
-key	关键字值
-set	变量值
-tdmmap	生成并处理一个TDM circuits表 一个回路必须为存在的呼叫有效
-dynamicStart	变量值 设置启动 dynamic_id变量的offset
-dynamicStep	变量值 设置-dynamicStep变量的增量

*** 呼叫行为选项:	
-aa	对INFO, UPDATE NOTIFY消息自动回200ok
-base_cseq	自定义每个呼叫的cseq的初始值
-cid_str	Call ID字符串（默认%u-%p@%s）。%u=call_number，%s=ip_address， %p=process_number, %%=%（在任何命令中）
-d duration	控制呼叫的长度（以毫秒为单位），更准确的来说，在场景中如果他们没有一个'milliseconds' 部分，这种控制指“pause”指令的持续时间， 默认值为0.
-deadcall_wait	设置一个Call-ID在sipp中的保持时间，默认单位是ms，若时间过长，Call-ID会在sipp中长时间保持，将会使sipp认为同一个Call-ID的sip消息为重发的报文而不予处理。
-auth_uri	强制用于认证URI的值 ，默认情况下，URI由remote_ip:remote_port组成。
-au	为认证设置授权username，默认从 -s中携带
-ap	为认证设置密码。默认是'password'
-s	设置请求URI的username部分，默认是'service'
-default_behaviors	设置SIPp用的默认行为。可能的值是： -所有使用所有默认行为。 -没有使用任何默认行为。 -为中止的呼叫发送bye -pingreply回复ping请求

	如果一个行为以a- 开头，那行就是关闭。例如： all,-bye
-nd	不是默认。禁用sipp的所有以下的默认配置： -当UDP重发超时时，通过发送一个BYE或一个CANCEL来中止呼叫 -当收到携带no ontimeout属性超时时，通过发送一个BYE或一个CANCEL来中止呼叫 -收到意外的BYE消息时，发送一个200 OK，关闭这个呼叫 -收到意外的CANCEL消息时，发送一个200 OK，关闭这个呼叫 -收到意外的PING消息时，发送一个200 OK，继续这个呼叫 -收到任何其它的意外消息时，通过发送一个BYE或一个CANCEL来中止呼叫
-pause_msg_ign	在暂停期间收到了定义的忽略消息
*** 插入文件选项	
-inf	从呼叫的时候从外部csv文件中注入值到场景。 第一行以顺序（用SEQUENTIAL）、随机（RANDOM）或用户(USER)、命令读取数据。 每一行对应一个呼叫，用一个或多个“;”划分数据字段，这些字段在XML场景文件中可以归类为[field0], [field1], ...，可以同时用好几个CSV文件。（语法：-inf f1.csv -inf f2.csv ...）
-infindex	文件字段 用字段创建一个文件索引，例如-inf users.csv -infindex users.csv 0 在第一个键中创建一个索引。
-ip_field	客户端会发送从包含IP地址消息中设置-ip_field字段。 如果省略这个选项，将用’-t ui’选项，假定field 0。

	结合'-t ui'用这个选项。
***RTP行为选项	
-mi	设置本地媒体的ip地址（默认：本地主要主机IP地址）
-rtp_echo	启用rtp回应，用-mp定义端口收到的RTP/UDP包回传给发送者
-mb	设置RTP回传buffer大小（默认：2048）
-mp	设置本地RTP回传端口号，默认为6000
-min_rtp_port	RTP socket 范围的最小端口号
-max_rtp_port	RTP socket 范围的最大端口号
-rtp_payload	RTP默认payload类型
-rtp_threadtasks	每线程回放任务的RTP号
-rtp_buffsize	设置 rtp socket send/receive buffer大小
***呼叫速率选项	
-r rate (cps)	设置呼叫速率（每秒呼叫的个数），在测试过程中这个值可以通过按'+','_','*'或 '/'来改变，默认为10。 按'+'键以1*速率量增加呼叫速率 按'-'键以1*速率量减少呼叫速率 按'*'键以10*速率量增加呼叫速率 按'+'键以10*速率量减少呼叫速率 如果用 -rp选项，呼叫速率通过用户给定的毫秒周期的计算

-rp period (ms)	为呼叫速率指定用毫秒为速率周期呼叫速率。 默认是1秒，默认单位是毫秒，这允许你每m毫秒有n个呼叫（通过用-r n -rp m命令）例如： -r 7 -rp 2000 ===> 每2秒7个呼叫 -r 10 -rp 5s ===>每5秒10个呼叫
-rate_scale	用the '+', '-', '*', and '/'的单位来控制
-rate_increase	指定每-fd秒的速率增加，这允许你为每个单独的日志周期增加压力。 例如：-rate_increase 10 -fd 10 =➔每10秒加上10个呼叫。
-rate_max	如果设置了-rate_increase，呼叫速率最大值为此参数的值。 例如：-rate_increase 10 -max_rate 100 =➔以10个呼叫速率增加，直达到值为100.
-no_rate_quit	如果设置了-rate_increase，达到-rate_max的速率不退出。
-l	设置同时呼叫最大的最大数量，一旦达到这个限制，直到打开呼叫开始下降后流量才会上升，默认(3 * 呼叫持续时间 * 速率)
-m	当'calls'处理呼叫时停止测试并退出
-users	不以一个固定的速率开始呼叫，以'users'呼叫数量开始，呼叫的数量优质不变。
*** 重发和超时选项	
-recv_timeout	全局接收超时。默认单位是毫秒。如果没有收到预期的消息，呼叫超时并中止呼叫

-send_timeout	全局发送超时。默认单位是毫秒。如果没有发送一条消息（由于堵塞），呼叫超时并中止呼叫。
-timeout	全局超时。默认单位是秒。如果设置了这个选项，SIPp会在nb单位后退出。(-timeout 20s;20秒之后退出).
-timeout_error	如果达到了设置全局超时时间，SIPp失败。（要求有-timeout option）
-max_retrans	在呼叫超时之前，UDP重发的最大数量，默认INVITE重发是5，其它是7。
-max_invite_retrans	在呼叫超时之前，UDP重发中invite重发的最大数量
-max_non_invite_retrans	在呼叫超时之前，UDP重发中非invite重发的最大数量，
-nr	在UDP模式中禁用重发机制
-rtcheck	选择重发检测方式：全部（默认）或松散。
-T2	全局T2-计数器，用毫秒表示
***第三方呼叫控制选项	
-3pcc	在3pcc模式下启动SIPp(“(″Third Party call control″”).IP地址依赖于在3pcc角色。 -当第一个双命令是’sendCmd’时，那么这是远程双socket的地址。SIPp会试图发送双命令去连接address:port（这个实例会在所有其它的3pcc场景后开始）。 例如：3PCC-C-A scenario -当第一个双命令是’recvCmd’时，那么这是本地双socket的地址。SIPp会为双命令打开这个address:port去监听。 例如：3PCC-C-B scenario.

-master	3pcc扩展模式：指明主号码
-slave	3pcc扩展模式：指明从号码
-slave_cfg	3pcc扩展模式：指明存储主和从地址的文件
*** 性能和看门狗选项	
-timer_resol	设置计数器决议。默认单位是毫秒。这个选项影响计数器精度。比较小的值有更精确的调度，但影响CPU使用率。如果打开压缩功能，设定值为50ms.默认值是10ms.
-max_recv_loops	设置每周期读取接收到消息的最大数量。由于大业务量值会增长。默认值是1000.
-max_sched_loops	设置每个事件循环呼叫运行的最大数量。由于大业务量值会增长。默认值是1000.
-watchdog_interval	设置watchdog计数器监视的间隔。默认是400
-watchdog_reset	如果此次时间周期多于watchdog计数器没有启动，那么会重新设置最大触发器的数量。默认是10分钟。
-watchdog_minor_threshold	如果在watchdog执行计数之间有一个副的差错时间长于这次周期。默认是500.
-watchdog_major_threshold	如果在watchdog执行计数之间有一个主的差错时间长于这次周期。默认是500.
-watchdog_major_maxtriggers	在测试结束之前，主watchdog计数器有多少次被绊倒。默认为10.
-watchdog_minor_maxtriggers	在测试结束之前，副watchdog计数器有多少次被绊倒。默认为120.

***追踪、日志和统计选项	
-f	在屏幕上设置统计报告的显示频率。默认是1，默认单位是秒。
-trace_stat	把所有的统计信息转储到<scenario_name>_<pid>.csv文件中，用'-h stat'选项生成统计文件内容的详细信息。
-stat_delimiter	设置统计文件定界符
-stf	生成转储统计的文件名
-fd	设置统计转储日志报告的频率。默认是60，默认单位是秒。
-periodic_rtd	重置每个日志记录间隔响应时间分区
-trace_msg	在<scenario file name>_<pid>_messages.log中显示发送和接收的SIP消息。
-message_file	设置消息日志文件名称
-message_overwrite	覆盖消息日志文件（默认为true）
-trace_shortmsg	以CSV格式在<scenario file name>_<pid>_shortmessages.log中显示发送和接收的SIP消息
-shortmessage_file:	设置短消息日志文件的名称
-shortmessage_overwrite	覆盖短消息日志文件（默认为true）
-trace_counts	把个别的消息转储到一个CSV文件中

-trace_err	把所有意外的消息保存至<scenario file name>_<pid>_errors.log
-error_file	设置错误日志文件的名称
-error_overwrite	覆盖错误日志文件（默认为true）
-trace_error_codes	转储意外消息的SIP响应代码至<scenario filename>_<pid>_error_codes.log
-trace_calldebug	转储关于异常呼叫的debugging信息至<scenario_name>_<pid>_calldebug.log文件中
-calldebug_file	设置call debug文件的名称
-calldebug_overwrite	覆盖call debug文件（默认为true）
-trace_screen	当SIPp退出时转储统计的屏幕至<scenario_name>_<pid>_screens.log文件中。用于在背景模式下得到一个最终的状态报告（当用-bg 选项）
-trace_rtt	保存允许响应时间的跟踪至<scenario file name>_<pid>_rtt.csv中
-rtt_freq	Freq是强制使用的。通过-trace_rtt定义的log文件转储每个freq响应时间，默认值是200。
-trace_logs	允许追踪的<log>保存至<scenario file name>_<pid>_logs.log中
-log_file	设置log actions log文件的名称
-log_overwrite	覆盖log actions log文件（默认为true）

-ringbuffer_files	循环后保存了多少 error、message, shortmessage 和 calldebug 文件（How many error, message, shortmessage and calldebug files should be kept after rotation?）
-ringbuffer_size	在循环前保存 error, message, shortmessage 和 calldebug 文件的大小（How large should error, message, shortmessage and calldebug files be before they get rotated?）
-max_log_size	限制error, message, shortmessage and calldebug file保存的最大值
信号处理:	
可以用POSIX信号控制SIPp。以下信号处理：	
USR1	与按‘q’键类似，它会触发SIPp软退出。不再产生新的呼叫，且在退出SIPp时会完成所有正在进行的呼叫。
USR2	触发一个转储所有统计的屏幕信息至<scenario_name>_<pid>_screens.log文件中。当以背景模式下时特别有用，可以知道当前的状态。 例如：kill -SIGUSR2 732
退出代码:	出现严重错误或到达请求呼叫的数量时，SIPp退出时会生成以下其中的一个代码： 0：所有的呼叫全部成功 1：至少一个呼叫失败； 97：内部命令退出。呼叫可能已经被处理了。 99：没有呼叫处理的一般退出； -1：严重错误； -2：绑定一个socket的严重错误

4 Performance testing with SIPp（用 SIPp 进行性能测试）

4.1 Advice to run performance tests with SIPp（建议用 SIPp 运行性能测试）

SIPp本意就是为了SIP性能测试。研究高呼叫率和/或高数量下的SIP并发呼叫。SIPp提供了下面的一些参考：

- 用一个Linux系统达到一个高性能。SIPp在Windows中(通过用CYGWIN)不能处理高性能.

- 限定追踪到最小范围（使用-trace_msg -trace_logs仅限于场景调试）。

- 理解SIPp内部调度机制和用-timer_resol, -max_recv_loops和-max_sched_loops命令行参数去协调SIPp运行的系统。

一般来讲，运行性能测试也意味着测量响应时间。你能用SIPp计时器（在场景中的start_rtd, rtd和命令行选项-trace_rtt）去测试这些响应时间。测量的精确度完全依赖timer_resol参数（如在"SIPp's internal scheduling"描述的）。你也许想用另一个"objective"方式来测量一个高精度（像Wireshark工具（http://www.wireshark.org/））的响应时间测量，

4.2 SIPp's internal scheduling（SIPp 的内部调度）

SIPp有一个单线程事件循环体系来处理高SIP流负载。SIPp的事件循环追踪各种任务，其中大多数是场景定义的呼叫。除了代表呼叫任务，还代表：屏幕更新任务、统计更新任务、打开呼叫任务和监视任务，组成SIPp主要执行循环有：

1. 停止的计时器唤醒任务；

2. 运行到max_sched_loop任务的运行状态（执行每个呼叫直到它不再可能运行为止）

3. 依次处理每个sockets,从各种sockets中读取max_recv_loops消息。

SIPp会不断的循环执行，直到直到告诉停止的一些条件。（例如：用户按下‘q’键或全局呼叫限制或到达超时时间）。

在命令行中有指定几个参数来调整这个调度。

- timer_resol:在主要循环中，管理所有的呼叫（管理等待、重发…）,每"timer_resol"ms(毫秒)是最好的。重发的延时必须要比"timer_resol"高。默认计时器解析度是1毫秒，这是SIPp当前支持最精确的解析度。如果你增加了这个参数，

SIPp会垮掉，在高负载下有可能遇上重发。如果有太多的呼叫或每个呼叫携带的消息太长，计时器解析度就不准了。

- max_recv_loops and max_sched_loops：读取和接收的处理消息。"max_recv_loops"是一次能被读取的最大消息量。这些限制预防SIPp从读取和处理新消息sockets时拒绝其它呼叫，为了达到高呼叫率，可以调大这两个值。小心，这两个参数会使SIPp增加CPU使用率。

- watchdog_interval,watchdog_minor_threshold, watchdog_major_threshold, watchdog_minor_maxtriggers, and watchdog_major_maxtriggers，如果你的呼叫负载导致SIPp不知所措时，设计的watchdog计时器提供反馈。开启Watchdog任务时每watchdog_interval milliseconds (默认400ms). 设置一个计时器，如果超过watchdog_minor_threshold milliseconds (默认500s)计时器没有服务，那么记录个"minor"触发器。如果触发器的数量多于watchdog_minor_maxtriggers；watchdog任务终结SIPp。同样地，如果由于没有服务的计时器多于watchdog_major_threshold milliseconds (默认3000ms),那么记录一个主触发器；如果记录的多于watchdog_major_maxtriggers，会终结SIPp。如果仅是偶尔查看消息，您的测试可能是可以接受的，但是如果这些事件很频繁，你需要考虑用一个更加强大的机器或机器设置去运行场景。

5 Useful tools aside SIPp（除 SIPp 外可用的工具）

5.1 JEdit

JEdit (http://www.jedit.org/)是一个GNU GPL用java编写的文本编辑器。几乎支持所有的平台。它非常强大，如果你把DTD (sipp.dtd (http://sipp.sourceforge.net/doc/sipp.dtd))作为XML场景放在相同的目录，它通常用语法检查来编辑SIPP场景。


sipp.dtd

5.2 Wireshark/tshark

Wireshark (http://www.wireshark.org/)是一个GNU GPL协议分析器，它的前身是Ethereal，支持SIP/SDP/RTP。

5.3 SIP callflow（SIP 呼叫流）

当追踪SIP呼叫时，用wireshark能够得到一个呼叫流。"callflow"工具允许你用一

个图形的方式：http://callflow.sourceforge.net/来实现

6 Getting support(获得支持)

你可以从SIPp用户社区获得基于Email的帮助，邮件列表地址是 sipp-users@lists.sourceforge.net,为了防止垃圾邮件,邮件仅限订阅用户使用，另外，你也可以浏览SIPp邮件列表档案：
http://lists.sourceforge.net/lists/listinfo/sipp-users

7 Contributing to SIPp(捐助 SIPp)

SIPp欢迎您的捐助，许多SIPp的功能的开发已经得到了捐助，你可以点击这里了解更多信息。

SIPp参与人员：

Richard GAYRAUD【原始代码】

Olivier JACQUES【代码/文档】

Robert Day【代码/文档】

Charles P. Wright【代码】

众多捐助者【代码】

李明禄【中文翻译】

对于网站的建议或反馈请联系：Rob Day

由于都不是专业翻译人员，翻译中不免不些纰漏与错误，也请大家发邮件到 402007353@qq.com错误之处，请您批评更正，谢谢。