

# SIPp 使用手册中文版

黄龙舟 译

# 目录

一、 简介 .....	4
二、 安装 SIPp.....	4
1. 获取 SIPp .....	4
2. 安装 SIPp .....	5
3. 提高文件标识符限制.....	6
三、 使用 SIPp.....	7
1. 主要特性 .....	7
2. 集成的脚本（场景） .....	7
2.1 示范脚本：主叫（UAC） .....	8
2.2 示范脚本：带媒体的 UAC .....	8
2.3 示范脚本：被叫（UAS.xml） .....	9
2.4 示范脚本：正则表达式，regexp.xml.....	9
2.5 示范脚本：分支，branchc.xml 和 branchs.xml .....	9
2.6 UAC Out-of-call 消息 .....	10
2.7 3PCC（第三方呼叫控制） .....	10
3. 扩展的 3PCC .....	13
4. 控制 SIPp .....	14
5. 后台运行 SIPp.....	16
6. 创建自定义的 XML 脚本.....	16
6.1 创建客户端（类似 UAC）脚本 .....	23
6.2 创建服务端（类似 UAS）脚本.....	27
6.3 动作（Actions） .....	27
6.4 变量.....	34
6.5 条件分支.....	36
6.6 SIP 认证.....	38
6.7 初始场景.....	40
7. 运行界面 .....	40
8. 传输层模式.....	42
9. 媒体处理 .....	44
10. 退出码 .....	44
11. 统计 .....	44
12. 错误处理 .....	45
13. 在线帮助 .....	46
四、 使用 SIPp 做性能测试 .....	50
1. 使用 SIPp 做性能测试的建议 .....	50
2. SIPp 内部调度机制.....	50
五、 有用的工具 .....	51
1. JEdit.....	51
2. Wireshark/tshark .....	51
3. SIP callflow .....	51

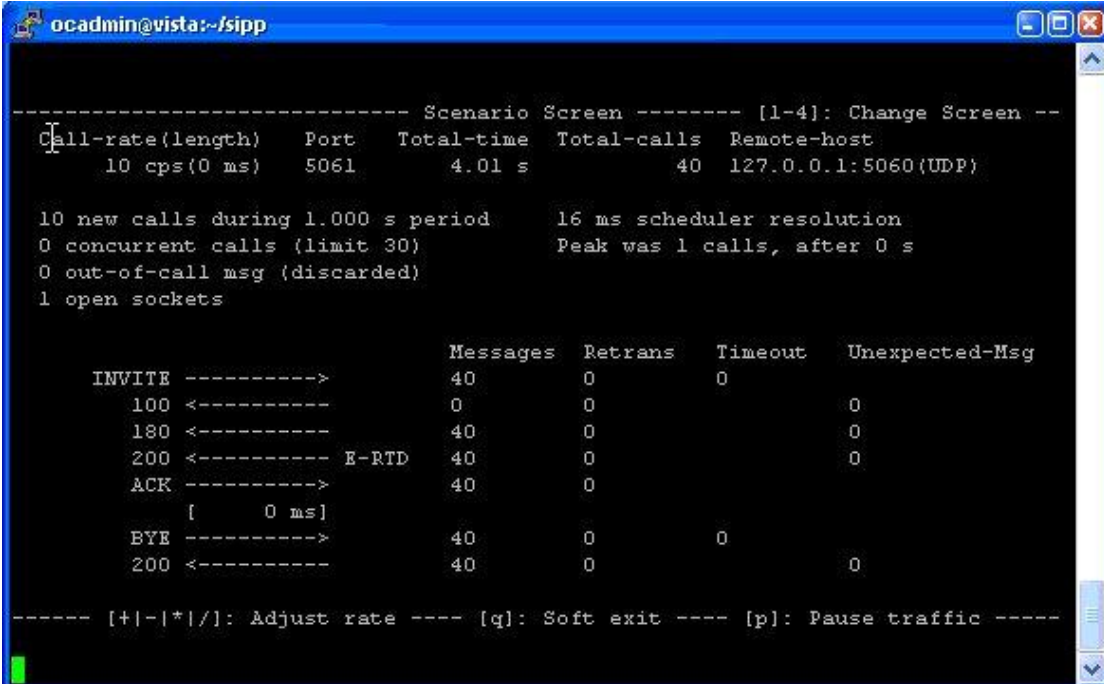
六、	获得支持 .....	51
七、	捐助 SIPP.....	52

## 一、简介

Sipp 是一个测试 SIP 协议性能的工具软件，它包含了一些基本的 SipStone 用户代理工作流程（UAC 和 UAS），并可使用 INVITE 和 BYE 建立和释放多个呼叫。它也可以读 XML 的场景文件，即描述任何性能测试的配置文件（可以用他来模拟现场的 SIP 信令，以重现出现的故障；或者可以自定义 SIP 协议以测试终端对某些方面的容错或错误处理能力）。它能动态显示测试运行的统计数据（呼叫速率、信号来回的延迟，以及消息统计）。周期性地把 CSV 统计数据转储，在多个套接字上的 TCP 和 UDP，利用重新传输管理的多路复用。在场景定义文件中可以使用正则表达式，动态调整呼叫速率。Sipp 可以用来测试许多真实的 SIP 设备，如 SIP 代理，B2BUAs, SIP 媒体服务器，SIP/x 网关，SIP PBX，等等，它也可以模仿上千个 SIP 代理呼叫你的 SIP 系统。另外，Sipp 还可以简单用来测试 SIP 协议。

本文档是根据 sipp v3.4 版本来编写的，早期的 sipp 版本可能不含有本文档提到的一些特性。

下面是 Sipp 的运行界面截图：



```
ocadmin@vista:~/sipp
----- Scenario Screen ----- [1-4]: Change Screen --
Call-rate(length)  Port  Total-time  Total-calls  Remote-host
      10 cps(0 ms)  5061      4.01 s      40  127.0.0.1:5060(UDP)

10 new calls during 1.000 s period      16 ms scheduler resolution
0 concurrent calls (limit 30)           Peak was 1 calls, after 0 s
0 out-of-call msg (discarded)
1 open sockets

          Messages  Retrans  Timeout  Unexpected-Msg
INVITE  ----->      40      0      0
100 <-----      0      0      0
180 <-----      40      0      0
200 <----- E-RTD  40      0      0
ACK  ----->      40      0
[      0 ms]
BYE  ----->      40      0      0
200 <-----      40      0      0

----- [+-|*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----
```

## 二、安装 Sipp

### 1. 获取 Sipp

Sipp 在 GNU GPU 许可下发行，满足所有发行许可。Sipp 项目本来是由 Hewlett-Packard 的工程师创建并提供给 SIP 社区的，旨在为 SIP 社区提供一些有用的东西，但现在 HP 不再提供任何帮助和授权，sipp 下载地址为 <http://sourceforge.net/projects/sipp/files/>

## 2. 安装 SIPp

就像其他许多开源项目一样，SIPp 也同样提供两个版本：稳定版本和非稳定版本。稳定版本会被完全地测试再发布，本文档描述的所有功能都能正常使用。非稳定版本含有一些新功能并修正了一些新发现的 bug，可以从 SIPp SVN 下载非稳定版本。

SIPp 可安装在 Linux 和 Cygwin 上面，其他 Unix 版本可能也可以安装使用 SIPp，但在每一个发布周期中并未被测试。不过 SIPp 运行于 Cygwin 上面的性能没有 Linux 上面好，且 SIPp 只能在运行于 windows xp 或更高 windows 版本的 Cygwin 上面运行。在 Linux 环境中，SIPp 以源代码的形式提供，在使用 SIPp 之前需要编译 SIPp，在编译之前需要如下依赖组件：

- C++ 编译器
- curses 或者 ncurses 库
- 对于需要支持 TLS 功能：需要 OpenSSL 0.9.8 以上版本
- 对于需要支持播放 pcap：需要 libpcap 和 libnet
- 对于需要支持 SCTP：需要 lksctp-tools
- 对于需要支持统计分布的暂停：需要开源科学计算库（Gnu Scientific Libraries）

有如下四个选项来编译 SIPp：

- 不包含对 TLS, SCTP 或 PCAP 的支持

```
# tar -xvzf sipp-xxx.tar
# cd sipp
# ./configure
# make
```
- 包含 TLS 支持

```
# tar -xvzf sipp-xxx.tar
# cd sipp
# ./configure --with-openssl
# make
```
- 包含 PCAP play 支持

```
# tar -xvzf sipp-xxx.tar
# cd sipp
# ./configure --with-pcap
# make
```
- 包含 SCTP 支持

```
# tar -xvzf sipp-xxx.tar
# cd sipp
# ./configure --with-sctp
# make
```
- 或者组合支持以上功能

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

下面以 CentOS 6.4 为例，可以按如下步骤编译安装

```
#yum install gcc-c++ gcc automake autoconf libtool make
#yum install libpcap libpcap-devel
#yum install ncurses ncurses-devel
#tar -xvzf sipp-xxx.tar.gz
#cd sipp
#./configure --with-sctp --with-pcap --with-openssl
#make && make install
```

安装后使用 `sipp -v` 命令可查看当前 sipp 版本，如下图所示。

```
SIPp v3.4.0-TLS-SCTP-PCAP-RTPSTREAM built Feb 19 2014, 22:19:04.

This program is free software; you can redistribute it and/or
modify it under the terms of the GNU General Public License as
published by the Free Software Foundation; either version 2 of
the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public
License along with this program; if not, write to the
Free Software Foundation, Inc.,
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Author: see source files.
```

注：在 windows 平台的 CYGWIN 编译 SIPp 时，需要预先安装用于 CYGWIN 的 IPv6 扩展（IPv6 extension for CYGWIN），以及 libncurses，OpenSSL 和 WinPcap，SCTP 目前不支持。如果要支持播放媒体（pcap play），需要做如下工作：

- 复制 [WinPcap developer package](#) 到"C:\cygwin\lib\WpdPack"目录。
- 移除或重命名"C:\cygwin\lib\WpdPack\Include"目录下的"pthread.h"文件
- 根据以上步骤编译 SIPp

### 3. 提高文件标识符限制

如果你的系统不支持足够的文件标识符，当使用 TCP/TLS 模式（译者注：测试中发现使用 UDP 模式也一样）进行大并发量呼叫测试可能会遇到问题，可以使用两种方法来突破这种限制：使用命令行参数 `-max socket` 或者改变系统的设置来突破限制。至于如何提高文件描述符的最大值，不同的系统修改方法或步骤不同。

- 在内核为 2.4 版本的 Linux 系统上可以通过修改 `/etc/security/limits.conf` 文件和 `/etc/pam.d/login` 文件来更改，首先打开 `/etc/security/limits.conf` 文件并添加如下内容：

```
* soft    nofile    32768
* hard    nofile    65535
```

- 然后打开/etc/pam.d/login 文件并添加如下内容：  
session required /lib/security/pam\_limits.so
- 然后使用如下命令来永久更改文件描述符所能支持的最大值：  
ulimit -s unlimited（注，原文档命令为 ulimit -n unlimited，在 CentOS 6.4 上测试提示没有权限）
- 注销系统再登录，然后使用 ulimit -a 命令验证是否更改成功，成功后如下图所示（open files 为设置的值 32768 了）。

```
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority     (-e) 0
file size               (blocks, -f) unlimited
pending signals         (-i) 5892
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files              (-n) 32768
pipe size               (512 bytes, -p) 8
POSIX message queues    (bytes, -q) 819200
real-time priority      (-r) 0
stack size              (kbytes, -s) 10240
cpu time                (seconds, -t) unlimited
max user processes      (-u) 5892
virtual memory          (kbytes, -v) unlimited
file locks              (-x) unlimited
```

## 三、使用 SIPp

### 1. 主要特性

sipp 允许向远端系统发送一个或多个呼叫，sipp 目前只支持从命令行运行。可以在同一台设备上同时运行 UAS 和 UAC 脚本以示范 SIPp 的能力，举例如下。

首先运行 SIPp 内嵌的服务器脚本，命令如下：

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

然后在同一台主机上运行 SIPp 内嵌的客户端脚本，命令如下：

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

### 2. 集成的脚本（场景）

集成的脚本？是的，SIPP 内置了若干个可执行的脚本，如服务器（UAS）脚本和客户端（UAC）脚本，你也可以创建自己自定义的脚本（可参考第 6 节：创建自定义的 XML 脚本），SIPP 内置的只是一些基本的脚本。以下介绍的均为集成（内置）的脚本

## 2.1 示范脚本：主叫（UAC）

脚本文件：uac.xml，流程如下。[请点击我下载](#)。

```
SIPp UAC                               Remote
| (1) INVITE                             |
|----->|
| (2) 100 (optional)                       |
|<-----|
| (3) 180 (optional)                       |
|<-----|
| (4) 200                                   |
|<-----|
| (5) ACK                                   |
|----->|
| (6) PAUSE                                |
| (7) BYE                                   |
|----->|
| (8) 200                                   |
|<-----|
```

## 2.2 示范脚本：带媒体的 UAC

脚本文件：uac\_pcap.xml，流程如下。[请点击我下载](#)。

```
SIPp UAC                               Remote
| (1) INVITE                             |
|----->|
| (2) 100 (optional)                       |
|<-----|
| (3) 180 (optional)                       |
|<-----|
| (4) 200                                   |
|<-----|
| (5) ACK                                   |
|----->|
| (6) RTP send (8s)                       |
|=====>|
| (7) RFC2833 DIGIT 1                     |
|=====>|
| (8) BYE                                   |
|----->|
| (9) 200                                   |
|<-----|
```



### 2.3 示范脚本：被叫（UAS.xml）

流程如下。[请点我下载](#)。

```
Remote                SIPp UAS
| (1) INVITE          |
|----->|
| (2) 180             |
|<-----|
| (3) 200             |
|<-----|
| (4) ACK             |
|----->|
| (5) PAUSE           |
| (6) BYE             |
|----->|
| (7) 200             |
|<-----|
```

### 2.4 示范脚本：正则表达式，regexp.xml

流程如下。[请点我下载](#)。该脚本是一个 UAC 行为的脚本，关于正则表达式的详细解释请见“6.3 动作”。

```
SIPp regexp          Remote
| (1) INVITE          |
|----->|
| (2) 100 (optional) |
|<-----|
| (3) 180 (optional) |
|<-----|
| (4) 200             |
|<-----|
| (5) ACK             |
|----->|
| (6) PAUSE           |
| (7) BYE             |
|----->|
| (8) 200             |
|<-----|
```

### 2.5 示范脚本：分支，branchc.xml 和 branchs.xml

请点击下载 [branchc.xml](#) 和 [branchs.xml](#)。这两个脚本是相互配合使用的，一个作主叫，一个作被叫，并且用到了条件分支的功能，关于条件分支的详细介绍请参考“6.5 条件分支”。

```

REGISTER ----->
200 <-----
200 <-----
INVITE ----->
100 <-----
180 <-----
403 <-----
200 <-----
ACK ----->
[ 5000 ms]
BYE ----->
200 <-----

```

## 2.6 UAC Out-of-call 消息

脚本文件名称为 `ooc_default.xml`，[请点我下载](#)。

当 SIPp 作为 UAC 使用时收到一个 out-of-call 请求（注：out-of-call 请求是指所有没有被 uac 脚本所定义请求），则可能会用到 out-of-call 脚本，默认情况下脚本只是简单地回 200OK。这个脚本的功能可以被命令行参数 `-oocsf` 或者 `-oocsn` 所覆盖。

```

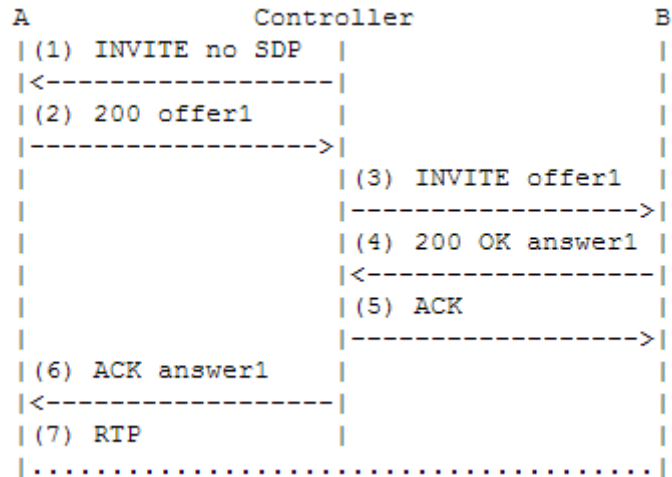
SIPp UAC          Remote
| (1) .*          |
|<-----|
| (2) 200         |
|----->|

```

## 2.7 3PCC（第三方呼叫控制）

3PCC 表示 3rd Party Call Control，即第三方呼叫控制，由 rfc3725 定义。虽然 SIPp 的 3PCC 功能本来是用来实现类似 3PCC 的情况，但是这个功能也可以用于其它场景，比如当你希望使用 sipp 来呼叫多方时。

为了保持 SIPp 的简单（记住，SIPp 是一个测试工具），通常情况下，一个 SIPp 实例（注：实例的概念是，对于一个可执行程序，打开一次就是一个实例，再开另一次又是另一个实例，每个实例会分配到一个 `HINSTANCE` 变量）只能同时与一个远端系统交互，这个是 3PCC 呼叫流程中的一个问题，以下面的流程 1 为例说明：其中，要求 SIPp 作为控制器（Controller）实现 3PCC 功能。

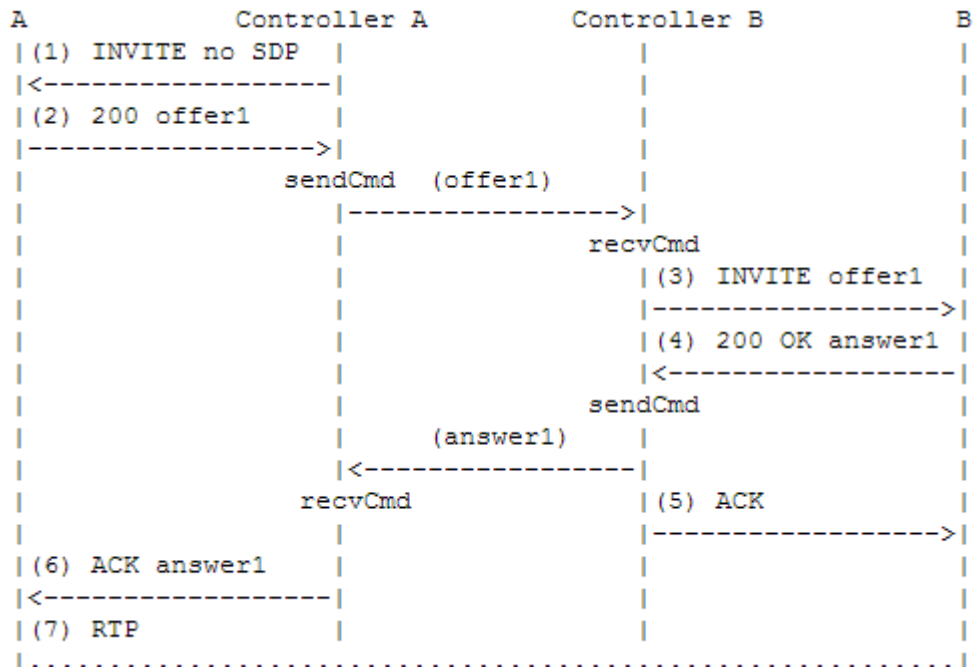


流程 1

用到的脚本文件如下：

- 脚本文件 A: [3pcc-A.xml \(original XML file\)](#)
- 脚本文件 B: [3pcc-B.xml \(original XML file\)](#)
- 脚本文件 C-A: [3pcc-C-A.xml \(original XML file\)](#)
- 脚本文件 C-B: [3pcc-C-B.xml \(original XML file\)](#)

不过，SIPp 提供了 3PCC 特性，SIPp 的 3PCC 功能允许同时运行的两个 SIPp 实例同步交互运行，我们还是以流程 1 为例说明：流程 1 中的一个 sipp 实例与远端 A 交互（这个实例我们称之为 3PCC 控制器 A），另一个 sipp 实例与远端 B 交互（我们称之为 3PCC 控制器 B），在 SIPp 内部看来，实际上的整个流程如下：



如上所示，大家可能注意到了，在控制器 A 和控制器 B 之间需要传递信息，比如由 A 提供的 SDP offer1 需要发送给 B，反过来也是一样，这个机制在 SIPp 中由命令 <sendCmd>和<recvCmd>来提供，如：

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

  ]]>
</sendCmd>
```

如上，我们发送了一个命令给孪生实例，这个消息中传递了呼叫的 Call-ID，用同样的方式，我们可以使用正则表达式再提取我们想要提取的内容，在下例中，我们把内容存储在变量 2 中：

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

接下来我们可以使用上面存储的变量 2，如下：

```
<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>
```

总之，sendCmd 和 recvCmd 可以看作是连接两个 sipp 实例的纽带，通过这个纽带，两个 sipp 实例之间可以互相传递参数。另一个可以使用 3PCC 特性实现的脚本流程如下：

1. A 呼 B，B 接听，B 和 A 通话
2. B 呼 C，C 接听，B 和 C 通话
3. B 将呼叫由 A 转到 C（REFER A 到 C）

4. A 接受转接，A 和 C 通话，B 退出通话

### 3. 扩展的 3PCC

SIPp 对 3PCC 功能进行了扩展，扩展的 3PCC 功能可以让 SIPp 的任意多个 SIPp 实例之间相互通信，并且每个实例连接一个远端主机或设备。

具体是这样实现的。在扩展的 3PCC 模式中，发起呼叫的 SIPp 实例运行于主（master）模式，其他实例运行于从（slave）模式，运行于从模式的 SIPp 实例含有实例名，可以使用命令行来对 SIPp 实例命名（如 s1,s2,...,sN 为从模式的实例名，m 为主模式的实例名），实例名和地址的对应关系必须存储在一个配置文件中，配置文件由命令行参数-slave\_cfg 指定，配置文件的格式如下：

```
s1;127.0.0.1:8080
s2;127.0.0.1:7080
m;127.0.0.1:6080
```

另外，每个 SIPp 实例必须使用这个文件的另一个拷贝，sendCmd 和 recvCmd 使用另外的属性：

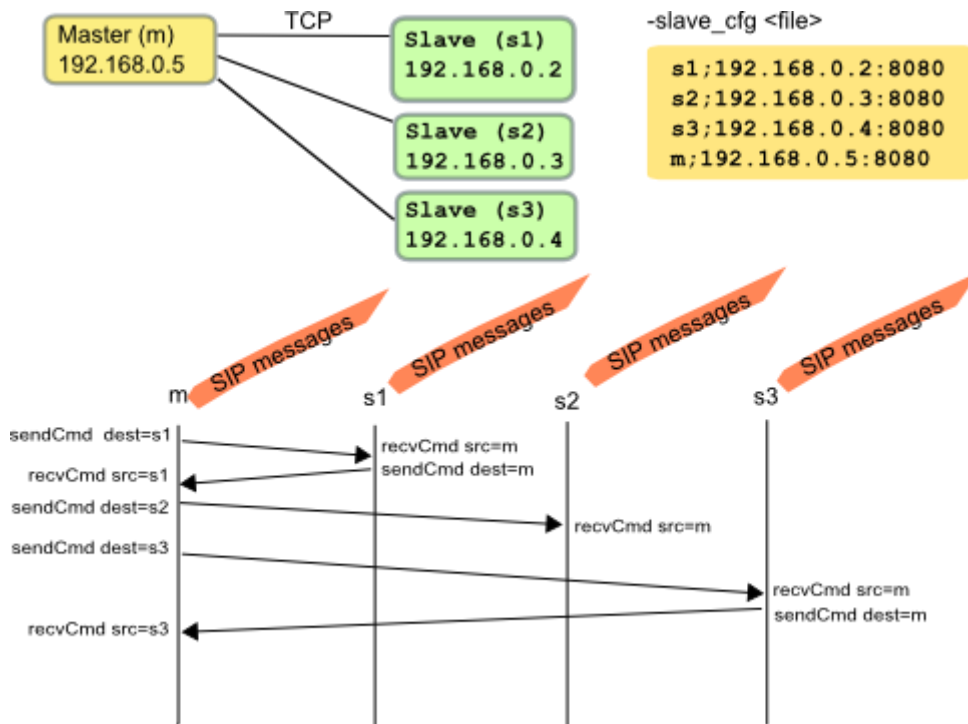
```
<sendCmd dest="s1">
  <![CDATA[
    Call-ID: [call_id]
    From: m
    [$1]
  ]]>
</sendCmd>
```

上面的脚本会发送一个命令到对点实例“s1”，“s1”既可以是“主”也可以是“从”，这取决于命令行参数，且必须与脚本定义的行为保持一致，“从”实例在使用 recvCmd 前必须要有一个 sendCmd。**注意：脚本必须包含一个“From”域，“From”的值为发送方的名称。**

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

上面这一段表明该实例正期望从“主”实例接收消息，**注意：主实例脚本必须在最后一个运行。**SIPp 中并没有集成（内置）扩展模式的 3PCC 脚本，不过可以很容易地通过对现有的 3PCC 脚本的调整来实现扩展的 3PCC

例：下面这个示意图表明了整个过程。在 SIPp 的“主”和“从”之间的箭头只是用来表示两个不同 SIPp 实例之间的同步命令的交换，SIP 消息的交互跟正常的是一样的。



## 4. 控制 SIPp

可以使用按键或者 UDP socket 交互地控制 SIPp。在运行过程中，SIPp 支持使用热键或者命令控制台的一些简单命令来控制，使用到的热键如下：

按键	作用
+	以 1 倍于 rate_scale (rate_scale 表示基准速率) 的速率增加呼叫
-	以 1 倍于 rate_scale 的速率减少呼叫
*	以 10 倍于 rate_scale 的速率增加呼叫
/	以 10 倍于 rate_scale 的速率减少呼叫
c	进入命令控制台模式
q	待所有呼叫完成后退出 sipp
Q	立即退出 sipp
s	转储当前屏幕状态到日志文件 (仅当使用了 -trace_screen 参数时)
p	暂停当前工作
1	显示脚本页面
2	显示统计页面
3	显示响应时间分布页面
4	显示变量页面
5	显示 TDM 页面
6-9	显示第 2 到第 5 个的响应时间分布页面

在命令控制台模式，可以使用单个命令控制 SIPp 的动作。命令控制台模式比热键更灵活，只不过需要手动输入命令，SIPp 提供了如下命令 (在运行过程按热键 c 进入命令控制台)：

命令	描述	举例
dump tasks	把错误信息转储到日志文件	dump tasks

set rate x	设置呼叫率	set rate 10
set rate-scale x	设置呼叫速率步长 (rate-scale)	set rate-scale 10
set users x	设置同时在线的用户数 (仅在指定了参数-users 时生效)	set users 10
set limit x	设置呼叫速率上限 (等同于参数-l)	set limit 100
set hide <true false>	是否显示被隐藏的 XML 属性	set hide false
set display <main ooc>	改变视图, 在主脚本页面和 ooc 脚本页面之间切换	set index true
set index <true false>	是否显示脚本视图中的索引信息	set display main set display ooc
trace <log><on off>	开启或关闭错误显示, 仅对"error", "logs", "messages", and "shortmessages"有效。	trace error on

### 1) 流量控制

SIPp 根据脚本指定的内容产生 sip 流量, 你可以控制每秒产生的 sip 呼叫 (calls) 个数。如果使用了参数-users, 则是指定每秒并发的用户数 (users), 可以使用下面的方法来调节呼叫速率:

- 热键 (上文已描述)
- 交互式命令
- 开数呼叫时指定的参数

有两个命令可用于控制速率, set rate X 用于设置当前的呼叫速率为 X。另外, set rate-scale X 用于设置呼叫基准速率为 X, 设置步长后, 可以使用+, -, \*, / 快速地控制呼叫速率。另外, 在开始运行 SIPp 时, 可以通过如下参数来指定呼叫速率:

- -r: 指定每秒呼叫的个数
- -rp: 指定呼叫速率的时间单位 (毫秒), 默认是 1000 毫秒 (1 秒)。通过这个参数可以指定每 m 毫秒里有 n 个呼叫 (使用命令-r n -rp m)

在运行过程中, 可以通过按“p”键暂停脚本的执行, SIPp 会不再发起新的呼叫并直到当前已经进行的呼叫正常结束, 你可以再按一下按键“p”恢复脚本的执行。在运行过程中可以按“q”键退出脚本执行, SIPp 会不再发起新的呼叫并直到当前已经进行的呼叫正常结束后退出。你也可以通过按“Q”键强制立即退出脚本的执行, 当按下 Q 键后, SIPp 会立即终止当前的呼叫并发送 BYE 或 CANCEL 消息, 你也可以按两次“q”键实现这一功能。

### 2) 远程控制

可以通过一个 UDP socket 来远程控制 SIPp, 例如:

- 可以使用脚本来自动地控制一些动作, 比如平滑地增加呼叫速率
- 被测试的程序通过脚本可以根据负载向 SIPp 发送命令自动地控制呼叫速率

每一个 SIPp 实例监听一个 UDP socket, 监听的端口从 8888 开始, 一直递增到 60 个, 可以使用如下命令控制 SIPp:

```
echo p >/dev/udp/x.y.z.t/8888 -> 暂停 SIPp, 相当于按 p 键
echo q >/dev/udp/x.y.z.t/8888 -> 退出 SIPp, 相当于按 q 键
echo "cset rate 100" >/dev/udp/127.0.0.1/8888 -> 发送 set 命令
```

如下是一个比较有用的实例, 动态地控制 SIPp 的呼叫速率:

```
#!/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

```

## 5. 后台运行 SIPp

使用命令参数 `-bg` 可以将 SIPp 在后台运行，如果不指定 `sipp` 脚本执行的次数，`sipp` 将会一直在后台运行。可以使用 `kill` 命令（`kill SIPp_PID`，`kill SIPp` 进程号）终止 SIPp。

## 6. 创建自定义的 XML 脚本

虽然 SIPp 已内置了脚本，但这远远是不够的，所以现在介绍一下如何创建自己的 SIPp 脚本。SIPp 脚本使用 XML 编写，一个 SIPp 脚本总是以如下开头：

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<scenario name="Basic Sipstone UAC">

```

而且总是以下面语句结束：

```

</scenario>

```

非常容易，对吧？开头和结尾很简单，至于中间的写法规则其实也不难，现在不要急于看完下面的整个表格，你可以先从后面的章节找到一个示例整体了解一下。在 SIPp 的脚本文件中，有很多公共属性用来实现流量控制功能和统计功能，这些公共属性可以应用于所有 sip 消息命令（SIP 消息命令包括 `<send>`，`<recv>`，`<nop>`，`<pause>`，`<sendCmd>` 和 `<recvCmd>`）。

可以应用于所有命令的公共属性的详细列表如下：

属性	描述	实例
<code>start_rtd</code>	开启一个“响应时间值（Response Time Duration）”计时器，便于统计响应时间	<code>&lt;send start_rtd="invite"&gt;</code> : 当这个消息被发送的时候会开启一个名为“invite”的计时器。可以在统计视图按 3, 6, 7, 8 察看所定义的计算器
<code>rtd</code>	停止五个中的一个“响应时间值”计时器，便于统计响应时间。计时器的个序号可在运行界面看到，如 RTD1, RTD2	<code>&lt;send rtd="2"&gt;</code> : 当这个消息被发送的时候会停止 2 号计时器，可以在统计视图按 3, 6, 7, 8 察看所定义的计算器。例如： <code>&lt;send rtd="invite"&gt;</code> : 当这个消息被发送的时候会停止 invite 计时器
<code>repeat_rtd</code>	重复使用“响应时间值”计时器，便于统计响应时间	<code>&lt;send rtd="invite" repeat_rtd="true"&gt;</code> : 名为 invite 的计时器值会被统计出来，但计时器不会停止
<code>crlf</code>	在脚本视图中对应的	<code>&lt;send crlf="true"&gt;</code>



	位置处输出一个空行	
next	在任意命令中使用 next 属性可以使当前的执行跳到脚本的另一部分，前提是收到了当段脚本定义的消息。	<p>例 1: 发送 ACK 后跳到标号为 12 的脚本</p> <pre> &lt;send next="12"&gt;   &lt;![CDATA[      ACK sip:[service]@[remote_ip]:[remote_port]SIP/2.0     Via: ...     From: ...     To: ...     Call-ID: ...     Cseq: ...     Contact: ...     Max-Forwards: ...     Subject: ...     Content-Length: 0    ]]&gt; &lt;/send&gt; </pre> <p>例 2: 当收到 403 消息后跳到标号为 5 的脚本</p> <pre> &lt;recv response="100" optional="true"&gt; &lt;/recv&gt; &lt;recv response="180" optional="true"&gt; &lt;/recv&gt; &lt;recv response="403" optional="true" next="5"&gt; &lt;/recv&gt; &lt;recv response="200"&gt; &lt;/recv&gt; </pre>
test	与 next 一起使用，除了满足 next 属性的要求外，同时仅当设置了与 test 相关的变量才能跳到指定脚本段。	<p>例: 当设置了变量 4 时，在发送了 ACK 消息后跳到标号为 6 的脚本</p> <pre> &lt;send next="6" test="4"&gt;   &lt;![CDATA[      ACK sip:[service]@[remote_ip]:[remote_port] SIP/2.0     Via: ...     From: ...     To: ...     Call-ID: ...     Cseq: ...     Contact: ...     Max-Forwards: ... </pre>

		<pre>Subject: ... Content-Length: 0  ]]&gt; &lt;/send&gt;</pre>
chance	与 test 一起使用，设置后按设定的机率跳到指定脚本。	例：设置了变量 3 后，以 90%的机率跳到标号 5 的脚本 <pre>&lt;recv response="403" optional="true" next="5" test="3" chance="0.90"&gt; &lt;/recv&gt;</pre>
condexec	当 condexec 中的属性值被设置时执行这个元素，具体例子可以参考“6.7 初始场景”部分。	<pre>&lt;nop condexec="executethis"&gt;</pre>
condexec_inverse	与 condexec 相反	<pre>&lt;nop condexec="skiphis" condexec_inverse="true"&gt;</pre>
counter	累加计数器，计算对应参数的执行次数，结果在统计视图中体现出来。	<pre>&lt;send counter="MsgA"&gt;</pre> :当该消息发送后，“MsgA”递增 1。

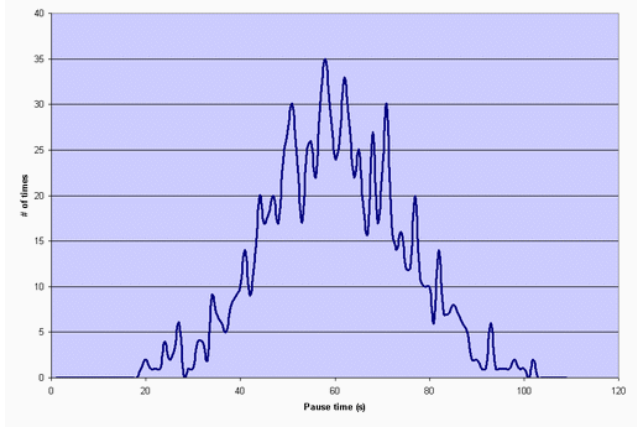
上表列出了公共属性，下表介绍所有 SIP 消息命令，每一个 SIP 消息命令(<send>, <recv>, <nop>, <pause>, <sendCmd>和<recvCmd>)含有自己独特的属性，注意不要与公共属性混淆。

SIP 消息命令及它们具有的属性详细列表如下：

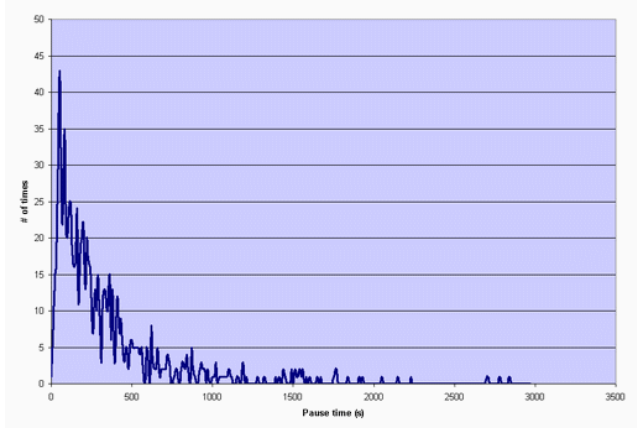
命令	属性	描述	用例
<send>	retrans	仅用于使用 udp 协议时，定义 udp 报文重发送时间，遵循 rfc3261 标准 section 17.1.1.2	<pre>&lt;send retrans="500"&gt;</pre> :表示每 500ms 重传输该消息。
	lost	模拟丢包率	<pre>&lt;send lost="10"&gt;</pre> :发送的丢包率为 10%
	start_txn	本属性用来记录响应时间，该属性记录本次发送的消息的 branch id，以在收到回应时能够精确匹配，若没有此属性则根据 Cseq 匹配。	<pre>&lt;send start_txn="invite"&gt;</pre> : 存储本次会话（由 branch id 唯一指定）消息，名称记为“invite”
	ack_txn	本属性用来记录响应时间，该属性与 start_txn 指定的消息对应，这两个消息必须成对出现，否则报错	<pre>&lt;send ack_txn="invite"&gt;</pre> :对应上面发送的名为“invite”的会话消息
<recv>	response	指定 SIPp 期望收到的 SIP 消息代码，如 1xx, 2xx, 3xx 等	<pre>&lt;recv response="200"&gt;</pre> :表示 SIPp 期望收到代码为 200 的消息
	request	指定 SIPp 期望到收的 SIP 请求消息，如： INVITE，	<pre>&lt;recv request="ACK"&gt;</pre> :表示 SIPp 期望收到 ACK 请求

	ACK, BYE, REGISTER, CANCEL, OPTIONS 等。	
optional	<p>指定 SIPp 期望收到的消息是可选的, 对端可以回这个期望的消息, 也可以没有回这个期望的消息。如果 SIPp 收到一个消息, SIPp 先检查这个消息是否匹配一个标记为 optional 的消息, 如果不匹配则标记为 unexpected。</p> <p>当 optional 设置为 “global” 时, SIPp 在收到一个消息时会检查脚本的全部步骤。</p>	<recv response="100" optional="true">:表示 SIPp 期望收到代码为 100 的消息, 但如果没有收到也没有关系。
rrs	<p><b>Record Route Set</b>, 即路由记录设置, 如果该属性被设置为 “true” 则收到的消息中的消息头 “Record-Route” 会被存储, 并可以通过 [route] 关键词调用。</p>	<recv response="100" rrs="true">.
auth	<p>鉴权认证。如果该属性被设置为 “true” 则消息中的消息头 "Proxy-Authenticate:" 会被存储并可以用来生成认证用的关键词 [authentication]。</p>	<recv response="407" auth="true">.
lost	模拟丢包率	
timeout	<p>设置超时, 如果期望的消息在指定的时间里面没有收到, 除非指定了 ontimeout 标否, 否则呼叫将被终止。</p>	<recv timeout="100000">
ontimeout	在超时之后跳到指定标号	<p>例: 在 100s 内没有收到消息 100 则跳到标号 5.</p> <pre>&lt;recv response="100" timeout="100000" ontimeout="5"&gt; &lt;/recv&gt;</pre>
action	指定当收到指定的消息时 SIPp 需要采取的动作	<p>action 的正则表达式实例:</p> <pre>&lt;recv response="200"&gt; &lt;action&gt;</pre>

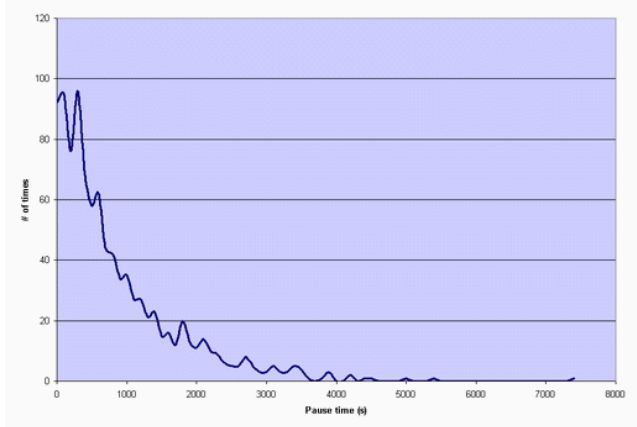
			<pre>&lt;ereg regexp="([0-9]{1,3}\.){3}[0-9]{1,3}:[0-9]*" search_in="msg" check_it="true" assign_to="1,2"/&gt; &lt;/action&gt; &lt;/recv&gt;</pre> <p>表示收到响应消息 200 后,在消息中搜索匹配正则表达式<code>(([0-9]{1,3}\.){3}[0-9]{1,3}:[0-9]*)</code>的字符串,并存储在变量 1 和 2 中,这个表达式的意义实际上是提取 ip 地址和端口号。</p>
	regexp_match	布尔值。检查收到的请求 (Request) 是否匹配指定包含的消息,如果匹配则应用正则表达式,这个方法可以用来一次提取多个请求中的字符串,非常方便	<p>例:检查是否匹配 MESSAGE or PUBLISH or SUBSCRIBE requests。</p> <pre>&lt;recv request="MESSAGE PUBLISH SUBSCRIBE" crlf="true" regexp_match="true"&gt; &lt;/recv&gt;</pre>
	response_txn	表示这个响应消息是与之前开始的 (start_txn) 对应,通过 branch id 来匹配	<pre>&lt;recv response="200" response_txn="invite" /&gt;</pre> <p>表示仅响应以 start_txn="invite" 发送的消息</p>
<pause >	milliseconds	定义暂停的时间,单位为 ms。当没有指定时间时,则使用命令行参数-d 来指定。	<pre>&lt;pause milliseconds="5000"/&gt;</pre> <p>暂停脚本 5 秒钟</p>
	variable	表示使用哪个呼叫变量来决定该呼叫是否需要暂停	<pre>&lt;pause variable="1" /&gt;</pre> <p>当含有呼叫变量 1 时暂停</p>
	distribution	表示使用 GSL(GNU 科学计算库)决定的呼叫长度来对呼叫进行暂停。如果不用 GSL,则可以使用固定值或者指定一个范围。使用高斯分布,则有以下几种统计分布可选: normal, exponential, gamma, lambda, lognormal, negbin, (negative binomial), pareto, 和 weibull,在选择哪种分布时,需要指定对应的参数。	<p>1、不使用 GSL 时,可以使用以下两种方法来对呼叫进行暂停:</p> <ul style="list-style-type: none"> <li>➤ <code>&lt;pause distribution="fixed" value="1000" /&gt;</code>,表示暂停 1 秒钟</li> <li>➤ <code>&lt;pause distribution="uniform" min="2000" max="5000"/&gt;</code>,表示暂停时间在 2 秒到 5 秒钟之间。</li> </ul> <p>2、使用 GSL 时需要指定参数,参数的命名与 Wikipedia 中关于分布的描述页面一致。举几例如下:</p> <ul style="list-style-type: none"> <li>➤ <code>&lt;pause distribution="normal" mean="60000" stdev="15000"/&gt;</code>,提供一个平均偏差为 60 秒和标准差为 15 秒的暂停分布值,平均差与标准差的值为毫秒整型值,分布图形如下:</li> </ul>



➤ `<pause distribution="lognormal" mean="12.28" stdev="1" />`, 创建一个分布, 该分布是以平均差为 12.28 和标准差为 1 为的自然对数, 平均差与标准差为双精度型, 单位为毫秒, 分布图如下:



➤ `<pause distribution="exponential" mean="90000" />`, 创建一个平均值为 15 分钟的指数分布, 分布图如下:



➤ `<pause distribution="weibull" lambda="3" k="4" />`, 创建一个威尔布分布。

➤ `<pause distribution="pareto" k="1" x_m="2" />` 创建一个帕累托分布。

			<ul style="list-style-type: none"> <li>➤ <code>&lt;pause distribution="gamma" k="3" theta="2"/&gt;</code>, 创建一个伽马分布</li> <li>➤ <code>&lt;pause distribution="negbin" p="0.1" n="2"/&gt;</code> 创建一个负二项分布。</li> </ul>
	sanity_check	完整性检查。默认情况下统计分布暂停都会执行完整性检查以保证暂停值不会超过程序设定的最大值, 可以将该值设置为 <code>false</code> 来禁用这个功能, 以提升性能。	<code>&lt;pause distribution="lognormal" mean="10" stdev="10" sanity_check="false"/&gt;</code>
<b>&lt;nop&gt;</b>	action	不对 SIP 协议处理, 仅用于执行命令	<p>例: 执行播放声音或视频的命令</p> <pre>&lt;nop&gt;   &lt;action&gt;     &lt;exec play_pcap_audio="pcap/g711a.pcap"/&gt;   &lt;/action&gt; &lt;/nop&gt;</pre>
<b>&lt;sendCmd&gt;</b>	<code>&lt;![CDATA[]&gt;</code>	用于 3PCC	<pre>&lt;sendCmd&gt;   &lt;![CDATA[     Call-ID: [call_id]     [\$1]   ]]&gt; &lt;/sendCmd&gt;</pre>
	dest	仅用于扩展模式的 3 方通话	<code>&lt;sendCmd dest="s1"&gt;</code>
<b>&lt;recvCmd&gt;</b>	action	当接收到命令时定义一个动作。	<pre>&lt;recvCmd&gt;   &lt;action&gt;     &lt;ereg regexp="Content-Type:.*"       search_in="msg"       assign_to="2"/&gt;   &lt;/action&gt; &lt;/recvCmd&gt;</pre>
	src	仅用于 3PCC 的扩展模式	<code>&lt;sendCmd dest="s1"&gt;</code>
<b>&lt;label&gt;</b>	id	标号, 用于给脚本分支, 属性值 <code>id</code> 是一个整型值, 最大值是 19。	比如标号 13: <code>&lt;label id="13"/&gt;</code>
<b>&lt;ResponseTimeRepartition&gt;</b>	value	定义时间间隔, 用于将响应时间进行分类	<code>&lt;ResponseTimeRepartition value="10, 20, 30"/&gt;</code> : 表示响应时间分类为 0-10, 10-20, 20-30 和大于 30

<b>&lt;Call Length Repartition&gt;</b>	value	定义时间间隔，用于将呼叫时长进行分类	<code>&lt;CallLengthRepartition value="10, 20, 30"/&gt;</code> :
<b>&lt;Globals&gt;</b>	variables	定义全局范围里的变量	<code>&lt;Globals variables="foo,bar" /&gt;</code>
<b>&lt;User&gt;</b>	variables	定义用户范围的变量	<code>&lt;User variables="foo,bar" /&gt;</code>
<b>&lt;Reference&gt;</b>	variables	抑制未用到的变量，当脚本运行提示有存在未用到的变量时，可以使用该命令来禁用它。	<code>&lt;Reference variables="dummy" /&gt;</code>

综上，用到的 SIP 消息命令其实也不多，分别为：send, recv, sendCmd, recvCmd, pause, ResponseTimeRepartition, CallLengthRepartition, Globals, User, 和 Reference。这些命令的使用其实也蛮有规律的。比如有些命令在书写时是成对出现的，如<send> </send>是一对，有些命令在书些时不用成对出现，如<pause milliseconds="5000"/>。为了使大家更容易理解，没有比举例更好的方法了，请看后面章节的构建脚本举例。

## 6.1 创建客户端（类似 UAC）脚本

客户端脚本以“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>

从示例可以看出，在“send”命令内部，必须要把待发送的 sip 消息插入“<![CDATA”和“]]>”中间，在这中间的所有内容将会被发送到远端系统。同时，你可能也注意到了在这个示例中有一些特殊的关键词，比如：**[service]**，**[remote\_ip]**，这些关键词用来指示 SIPp 将要用这些关键词做什么样的事情，具体可见下表“关键词列表”。

关键词	默认值	说明
[service]	service	由参数-s 传递，一般用来指定单个主被叫
[remote_ip]		远端设备地址
[remote_port]	5060	远端设备端口。可以在脚本中使用偏移量，如 [remote_port+3]
[transport]	UDP	指定传输层协议，UDP/TCP，由参数-t 决定
[local_ip]	主机本地地址	可以由参数-i 指定
[local_ip_type]		ip 版本
[local_port]	由系统随机分配	可由-p 指定，可以在脚本中使用偏移量，如 [local_port+3]
[len]		sdp 长度，用于“Content-Length”头域，由 sip 自动生成或者手动指定，可以添加偏移量，如[len+3]
[call_number]		呼叫索引，从 1 开始，每增加一个呼叫递增 1
[cseq]		初始值为 1，可以使用参数 -base_cseq 手动指定初始值。
[media_ip]		本地媒体流 ip，可以由-mi 参数指定
[media_ip_type]		本地媒体流 ip 版本
[media_port]		本地媒体流端口，可由-mp 指定，可以设置偏移量[media_port+3]
[auto_media_port]		自动指定媒体流端口，仅用于 pcap。该参数使多个呼叫的音频和视频端口从-mp 指定的端口开始，并给每个新的呼叫分配端口。SIPp 默认支持 10000 个 rtp 流并发
[last_*]		此关键词用于从接收的上一个 sip 消息中提取指定头域（如果存在）的值。比如[last_to]则表示从接收的上一个 sip 消息中提取 To 域的消息保存到[last_to]中并应用。
[field0-n file=<filename> line=<number>]		从外部文件 csv 加载值，file 表示选择从命令行中指定的 csv 文件的一个文件作为外部文件，line 定义选择的外部文件的起始行，field 选择字段
[file name=<filename>]		把指定的文件中内容全部插入到 sip 消息中。由于空格符，回车符及换行符不被某些关键词识别，所以插入的文件需要被精确地编辑成你想要的格式才行，否则会有乱码。
[timestamp]		当前时间戳，与错误日志的时间戳格式一样
[last_message]		上一个收到的消息



<b>[\$n]</b>		用于插入定义的呼叫变量 n
<b>[authentication]</b>		用于认证头 (authentication head), 这个关键词可以使用参数, 书写格式为: [authentication username=myusername password=mypassword], 如果没有指定用户名 (username), 则需使用命令行参数-au 或者-s 指定; 如果没有指定密码 (password), 则需使用命令行参数-ap 指定
<b>[pid]</b>		指定 sipp 的 pid, 即进程号
<b>[routes]</b>		如果在 recv 命令中已设置“rrs”属性为 true, 则“Record-Route”头被存储在关键词[route]中, 可使用[routes]调用。
<b>[next_url]</b>		如果在 recv 命令中已设置“rrs”属性为 true, 则[next_url]中包含 Contact 头中的内容
<b>[branch]</b>		生成一个由(z9hG4bK) + call number + message 索引组成的 branch id 到脚本中。如果你想要使用与之前的消息一样的 branch id, 你可以使用偏移量指定, 如[branch-N]
<b>[msg_index]</b>		在脚本中提供消息号
<b>[cseq]</b>		提供接收到的上一个请求消息中的 CSeq 值, 可以使用[cseq+1]来递增该值。
<b>[clock_tick]</b>		在消息中包含 sipp 的内部时钟
<b>[sipp_version]</b>		在消息中包含 sipp 的版本
<b>[tdmmap]</b>		在呼叫消息中包含 tdm 映射值
<b>[fill]</b>		使用字符填充消息
<b>[users]</b>		如果在命令行中指定了-users 参数, 则这个关键词表示当前已经运行的用户的数量
<b>[userid]</b>		如果在命令行中指定了-users 参数, 则这个关键词表示当前正在运行的用户的 id (范围为 0 到 users-1)

如上, 在脚本中我们已经发送了 Invite 消息, SIPp 可以使用“recv”命令等待接收消息。如下:

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

<recv response="180"> optional="true"
</recv>

<recv response="200">
</recv>
```

其中, 100 和 180 消息是可选接收的 (optional), 不过 200 是强制接收的, **注意: 在一序列“recv”命令中, 必须至少有一个消息是强制接收的。**接下来使用 SIPp 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>
```

我们也可以在这里插入一个暂停，如下，脚本会暂停 5 秒钟。

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

然后通过发送一个 Bye 消息并期望接收一个 200ok 来结束此次呼叫：

```
<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>
```

然后结束这个脚本

```
</scenario>
```

通过以上例子可以看出创建自己的 SIPp 脚本也并不是很难的，如果你想看其它例子，你可以在命令行使用参数 -sd 查看 SIPp 内嵌的脚本，比如 `sipp -sd branches` 可以查看分支脚本 `branches.xml`。

## 6.2 创建服务端（类似 UAS）脚本

脚本端脚本以“recv”命令开始，语法规则和可用命令跟客户端脚本的是一样的，不过在服务端脚本中会用到很多的[*last\_\**]关键词，举个例子如下：

```
<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>
```

在这个脚本中是准备回复 180 Ringing 消息，而且该 180 消息中的一些内容是从上一个接收的 invite 中提取出来的。

## 6.3 动作（Actions）

在一个“recv”或者“recvCmd”命令中，可以执行一些动作，目前支持如下动作：

- 正则表达式（ereg）
- 记录日志（log）
- 执行外部命令（如系统命令），内部命令，以及播放 pcap 文件（exec）
- 执行双精度变量算术运算
- 给变量赋字符串值
- 比较双精度变量
- 跳到脚本的指定索引
- 存储当前时间到变量
- 在注入的可索引文件中查找指定词
- 鉴权
- 改变一个呼叫的网络地址

### 1) 正则表达式

在 SIPp 中使用正则表达式可以实现如下功能：

- 提取 SIP 消息中的内容或者 SIP 头并存储到变量中以便在后续中用到（称之为再注入）
- 检查 SIP 消息中的内容是否满足要求

SIPp 中的正则表达式满足 PSIX 1003.2 规范，如果你想学习如何写表达式，可以参考此正则表达式教程，[请点击我](#)，下面是正则表达式动作的一些语法：

关键词	默认值	描述
regexp	-	用于使用正则表达式匹配接收到的消息头或者消息体。例： <pre>&lt;ereg regexp=".*" search_in="hdr" header="Contact:" check_it="true" assign_to="6" /&gt;</pre> 在消息头的头域 Contact: 中查询所有并分配给变量 6。“.*”用于代表所有字符串。
search_in	msg	有四个值： <ul style="list-style-type: none"> <li>➤ msg: 匹配整个消息（匹配后可以再拆分），比如如果从消息头 From: &lt;sip:hello@192.168.10.25:5060&gt;;tag=39f-29ee138 中做正则表达式为 ereg regexp="&lt;sip:.*@.*&gt;.*"进行匹配，则可以匹配为三部分。</li> <li>➤ hdr: 匹配消息头（消息头匹配后不能再拆分），比如如果消息头为 From: &lt;sip:hello@192.168.10.25:5060&gt;;tag=39f-29ee138, 正则表达式为 ereg regexp="&lt;sip:.*@.*&gt;.*", 此时匹配的结果只有一个，就是整个 From 头。</li> <li>➤ body: 匹配消息体</li> <li>➤ var: 匹配 SIPp 字符串变量。</li> </ul>
header	-	匹配头域，仅在 search_in 被设置为 hdr 时使用
variable	-	匹配变量，仅在 search_in 被设置为 var 时使用
case_indep	false	To look for a header ignoring case . Only used when the search_in tag is set to hdr.
occurrence	1	查找头域第几次出现，默认为 1,, 仅在 search_in 被设置为 hdr 时使用
start_line	false	仅查找起始行，仅在 search_in 被设置为 hdr 时使用
check_it	false	设置为真时，如果不匹配则置此次呼叫为失败，不能同 check_it_inverse 同时使用
check_it_inverse	false	与 check_it 相反，当设置为真时，如果匹配则置呼叫为失败
assign_to	-	将匹配的结果存储到指定单个变量或几个变量，使用\${n} 引用变量，可以将变量\${n}的内容应用于 sip 消息或者用于编写 sipp 条件分支脚本。在分配的几个变量中，第一个变量包含整个正则表达式的内容，接下来的其它变量依次匹配各子正则表达式，例： <pre>&lt;ereg regexp="o=([[:alnum:]]*) ([[:alnum:]]*) [[:alnum:]]*" search_in="msg" check_it="true" assign_to="3,4,5,8"/&gt;</pre> 如果 sip 消息包含有如下行：

		<pre>o=user1 53655765 2353687637 IN IP4 127.0.0.1 那么变量 3, 4, 5, 8 分别为: o=user1 53655765 2353687637 user1 53655765 2353687637</pre>
--	--	--

特别提示一下：在同一个动作（**action**）中可以同时使用多个正则表达式。举例说明如下：

- 第一个动作（**First action**）：
  - 提取接收到的 sip 消息中的第一个 ipv4 地址
  - 检查我们是否能够成功提取该 ip 地址
  - 将该 ip 地址赋给呼叫变量 1 和 2
- 第二个动作（**Second action**）：
  - 提取接收到的消息的 **Contact** 头
  - 将提取的 **contact** 头分配给变量 6

```
<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>
```

## 2) 记录日志

**log** 动作（**action**）可以记录自定义的日志，记录的消息保存在“脚本名\_进程号.log”文件中，所有关键词都可以应用于日志动作中以用来反馈我们想要看到的消息或日志。注意：只有在打开日志跟踪开关-**trace\_logs** 时才能产生日志。

例：

```
<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>
```

还可以使用另一个动作“**warning**”来记录 sipp 的错误日志消息，如下：

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

## 3) 执行命令

动作“**exec**”可以用来执行内部命令，外部命令，或者播放音频或视频（命令“**play\_pcap\_audio**”和“**play\_pcap\_video**”）。

### 内部命令：

内部命令“**int\_cmd**”有三个属性：**stop\_call**，**stop\_gracefully**（类似于按 q 键），**stop\_now**（类似于使用 **ctrl+C** 强制退出 sipp）。举个收到 603 后停止执行脚本的例子：

```

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

```

## 外部命令

使用外部命令“command”的属性可以执行当前主机的 shell 可执行的任意命令。举个例子回显收到了 invite 消息的例子：

```

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

```

## 媒体命令

命令“rtp\_stream”可以将内嵌的编码为 PCMA, PCMU 或者 g729 的音频文件(如 a.wav)转为流媒体向对端发送。动作“rtp\_stream”的用法如下：

- <exec rtp\_stream="file.wav" />, 流传输文件 file.wav, 假设该文件是 pcma 格式的
- <exec rtp\_stream="[filename],[loopcount],[payloadtype]" />, 流传输文件名为 [filename]的文件, 重放[loopcount]次, 默认是 1 次, 如果设置为-1 则一直循环播放, [payloadtype]告诉 SIPp 当前音频的负载类型(PCMA 为 8, PCMU 为 0, G729 为 18)。
- <exec rtp\_stream="pause" />, 暂停当前的流媒体
- <exec rtp\_stream="resume" />, 继续被暂停的流媒体

命令“pcap\_play”允许 sipp 发送预先录制好的 RTP 流媒体, 共有 play\_pcap\_audio / play\_pcap\_video 两个属性。如果选择 play\_pcap\_audio, 则在 SIP/SDP 中使用“m=audio”, play\_pcap\_video 则为“m=video”, 命令用法如下：

```

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

```

## 4) 变量操作

SIPp 的呼叫变量支持使用双精度浮点型, 可以使用 SIPp 的动作来对由参数“assign\_to”指定的值进行修改, 目前支持的动作有: assign, sample 和 todouble。assign 动作能将双精度值存储在“value”参数中; sample 动作基于上文提到的统计分布, 使用到的参数也跟统计分布一样; todouble 命令将 assign\_to 的值翻倍再引用。举例如下：

```

<nop>
  <action>
    <assign assign_to="1" value="1" />
    <sample assign_to="2" distribution="normal" mean="0" stdev="1"/>
    <!-- 将注入文件的第一个域的数据存储到变量 3, 你也可以使用正测表达式来存储-->
    <assignstr assign_to="3" value="[field0]" />
  </action>
</nop>

```

```

<!--将存储在变量 3 中的值转为双精度值并存储到变量 4-->
<todouble assign_to="4" variable="3" />
</action>
</nop>

```

可以使用 SIPP 对呼叫变量执行加减乘除操作，举例如下：

```

<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>

```

做算术操作时还可以使用变量，如下：

```

<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>

```

## 5) 字符串变量

可以使用命令<assignstr>创建字符串变量，该命令有两个参数“assign\_to”和“value”，例：

```

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

```

还可以对字符串进行比较，命令为<strcmp>，其对字符串比较的值是一个小于，等于，或大于 0 的双精度整型值。例：

```

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

```

## 6) 变量测试

变量测试功能可以用来灵活控制脚本的运行。变量测试的动作命令为“test”，含有四个参数：variable, value, assign\_to, compare。variable 是 value 和 assign\_to 比较后的结果，variable

是一个布尔值。test 的比较支持 6 种操作，分别为：

**equal, not\_equal, greater\_than, less\_than, greater\_than\_equal, 或 less\_than\_equal。**

例：如果\$1 比 10 小则设置\$2 为真

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

## 7) 查找

动作查找 **lookup** 主要用来可索引的注入文件，**lookup** 需要一个外部文件作为输入并输出一个行号（可使用命令 `-inindex users.csv 0` 对文件 `user.csv` 输出从 0 开始的行号），如下是一个从认证头（**authorization header**）中提取用户名并根据这个用户名找到 `user.csv` 文件中对应应该用户名的行号的一个例子：

```
<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>
```

## 8) 跳到一个索引

可以使用 **jump** 动作跳到一个指定索引，该功能可以用于编辑一些程序化的脚本。主叫可以使用 `[msg_index]` 来保存他们的索引，被叫可以使用 **jump** 动作跳回一个索引。当脚本中含有一个名为 “`_unexp.main`” 的标号时，**SIPp** 在任意时间如果收到了一个 **unexpected** 消息将会跳到这一个标号，并且存储错误之前的地址到变量 “`_unexp.retaddr`”。

```
<nop>
  <action>
    <jump value="5" />
  </action>
</nop>
```

跳到名为 `_unexp.retaddr` 的变量：

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

## 9) 获得当前时间

动作 **gettimeofday** 允许 **sipp** 获取 “新纪元时间（**unix epoch**）”，可以使用秒与毫秒。

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



## 10) 改变目标 IP 地址

动作 **setdest** 允许改变呼叫的目标 ip 地址。

```
<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>
```

## 11) 鉴权

动作 **verifyauth** 允许根据提供的用户名和密码对接受到的消息中的认证头进行检查，检查的结果存储在一个布尔变量中，该功能可以用来模拟验证鉴权的情况。目前只支持 MD5 摘要加密算法，在使用 **verifyauth** 之前必须发送一个挑战。举例如下：

```
<recv request="REGISTER" />
<send><![CDATA[

SIP/2.0 401 Authorization Required
[last_Via:]
[last_From:]
[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>
```

在接收到第二个请求之后，可以将消息中提取出来的用户名与外部文件的用户名和密码作比较，然后根据结果采取接下来的动作：

```
<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" />
```

注：外部文件 `users.conf` 需使用命令 `-inf users.conf -inindex users.conf 0` 引入

## 6.4 变量

对于复杂的脚本，你会需要存储一些可以跨越消息或者呼叫的信息。就像其他编程语言一样，SIPp 的 XML 脚本允许你使用变量来存储这些信息。在 SIPp 中，变量可以是阿拉伯数字或者字母。在之前的版本仅支持数字类型的变量，此版本可以支持字母类型的变量，所以在使用变量时，可以使用有意义的名词来作为变量名。

除了变量名外，SIPp 的变量类型也是很随意的，SIPp 的变量类型并不会被显示地声明，但是可以从动作看出变量的类型。SIPp 有 4 种变量：字符串，正则表达式匹配，双精度和布尔型。所有算术操作都使用双精度型。动作 `<test>` 和 `<verifyauth>` 创建的是布尔值。

SIPp 的变量也有生效范围，全局变量对所有呼叫生效，默认的变量仅对单个呼叫生效。全局变量一般用于存储脚本参数或者作为一个计数器。与 `-user` 参数结合的用户变量允许在多个呼叫之中保存其状态，全局变量和用户变量可以使用如下方法声明：

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

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

本地变量并不需要被声明。为了防止程序错误，SIPp 使用基本的检查确保在每个脚本中每一个变量至少会被使用一次，不过这个特性会给正则表达式的使用带来一些麻烦。因为正则表达式动作必须给匹配整个正则表达式的字符串分配一个变量，如果你并不想要整个字符串，比如只想要整个字符串中的一小部分，你也必须为整个正则表达式匹配的串分配一个变量。由于变量必须至少要使用一次，所以你必须使用 `Reference` 命令引用变量一次，举例如下：

```
<recv request="INVITE">
  <action>
    <ereg regexp="<:sip:([^\;@]*)" search_in="hdr" header="To:" assign_to="dummy,uri" />
  </action>
</recv>
<Reference variables="dummy" />
```

### 1) 从外部 CSV 文件引入变量

可以在命令行使用 `"-inf 文件名"` 参数来引入变量到脚本中。文件的第一行须申明变量的读取方式是顺序读取 (`SEQUENTIAL`) 还是随机读取 (`RANDOM`) 还是基于用户的方式读取 (`USER`)。每一行对应一个呼叫，使用 `";"` 分隔符分隔每一项数据，分开的项在脚本中作为变量名 `[field0],[field1], ……[fieldn]` 来引用。例如：

```
SEQUENTIAL
#This line will be ignored
Sarah;sipphone32
Bob;sipphone12
#This line too
Fred;sipphone94
```

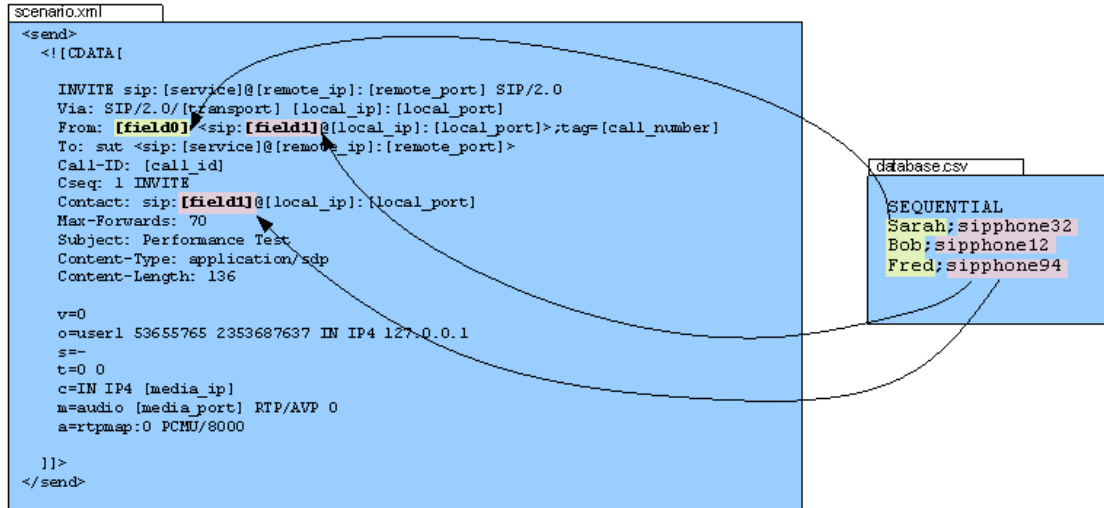
该文件中的行会被按顺序读取，第一个呼叫第一行，第二个呼叫第二行。在脚本中的任何地方只要出现了关键词 `[field0]`，根据第几个呼叫决定，这个关键词就会被替换为 `Sarah` 或

者 Bob 或者 Fred, [field1]也是类似。如果达到了文件末尾则再重新开始, 一直循环, 文件的大小没有限制。

另外可以使用参数不从第一行开始, 例如第从第二行开始:

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

上面的例子用图片说明一下, 以加深理解。



另外, 还可以使用不止一个外部文件来引入变量, 这在测试一些场景是很有用的, 比如你要做一个测试主叫号码是按顺序的但是被叫是随机的时候, 你就可以用一个第一行为顺序的 caller.csv 文件和一个第一行为随机的 callee.csv 文件来实现了。

```
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"]>
```

## 2) 格式化输出的引用文件

一个扩展的标准外部文件是“PRINTF”文件。通常情况下, 一个 SIPp 的外部文件是含有一些重复内容的, 比如:

```
USERS
user000;password000
user001;password001
...
user999;password999
```

如果要做一个这样的文件比较麻烦, SIPp 有一套机制可以自动在内存中生成一套比较有规律格式的文件, 这样既方便又能减少读取大文件的时间, 例如上面的那些内容可以如下简单表示:

```
USERS,PRINTF=999
user%03d;password%03d
```

当模板文前不止含有一行内容时, 如:

```
USERS,PRINTF=4
user%03d;password%03d;Foo
user%03d;password%03d;Bar
```

SIPp 能够将模板文件自动循环, 其相当于:

```

USERS
user000;password000;Foo
user001;password001;Bar
user002;password002;Foo
user003;password003;Bar

```

### 3) PRINTF 引入文件的参数

参数	描述	举例
PRINTF	文件包含多少行	PRINTF=10, 创建 10 行
PRINTFMULTIPLE	设置行增长的步长	PRINTF=10, PRINTFMULTIPLE=2 表示创建 10 行, 号码依次为 0, 2, 4, …… , 18
PRINTFOFFSET	设置生成数的起始值	PRINTF=10, PRINTFOFFSET=100 表示创建 10 行, 号码依次为 100-109. PRINTF=10, PRINTFMULTIPLE=2, PRINTFOFFSET=10 表示创建 10 行, 号码依次为 10, 12, 14, …… 28.

### 4) 索引注入的文件

参数 `-inindex` 允许对引入的文件生成索引, `-inindex` 的参数是将被索引的文件的文件名。假如文件 `user.csv` 的内容如下:

```

USERS
alice,pass_A
bob,pass_B
carol,pass_C

```

如果想提取第一行, 则为 `-inindex users.csv 0`。

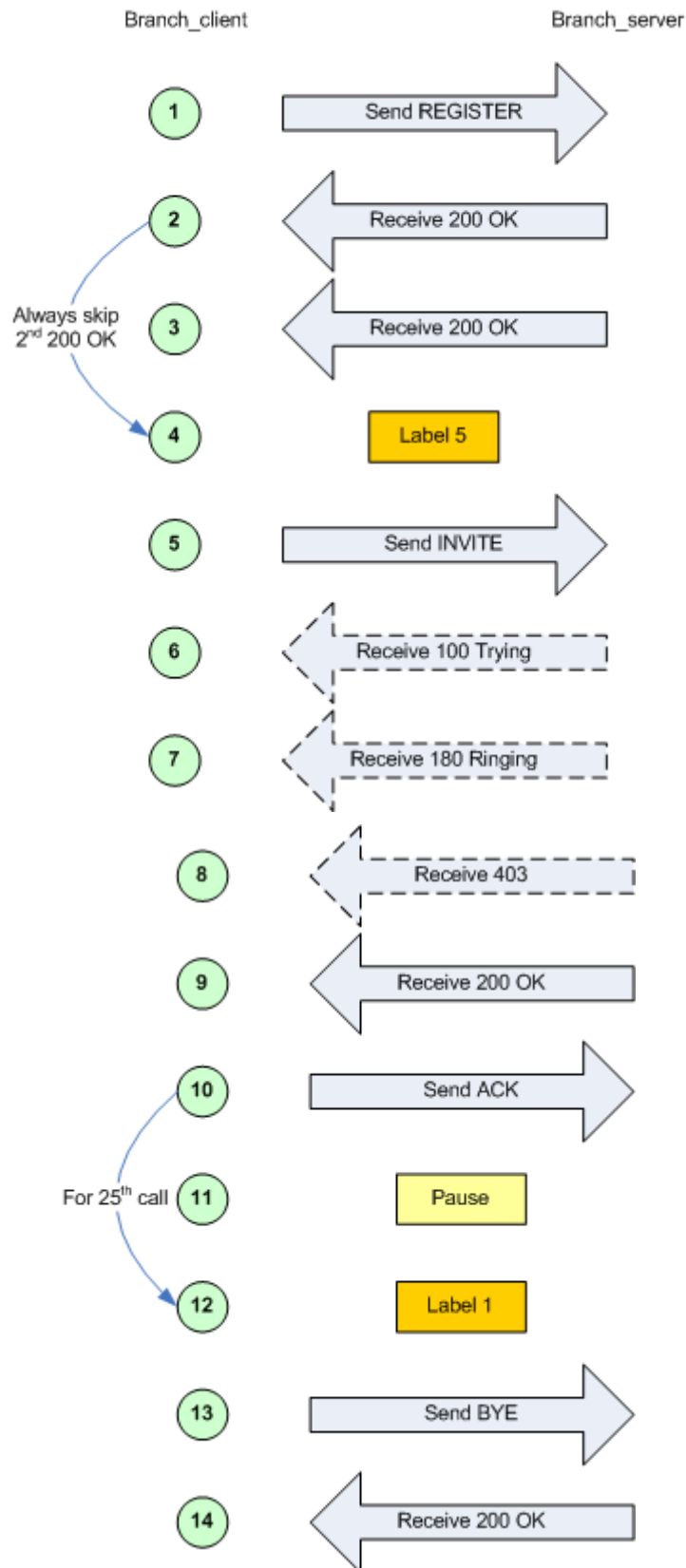
## 6.5 条件分支

### 1) 条件分支脚本

在 SIPp 中可以非线性地执行一个脚本, 你可以在检测到一个既定事件后从脚本的一个部分跳到脚本的另一个部分。这个功能使用标号实现: 在脚本中定义一个标号如 `<label id="n"/>`, 其中 `n` 是 1-19 之间的数, 每个标号则表示一部分。然后在任意一个动作命令之后添加一个 `next="n"` 的参数, 当脚本执行完这个动作后就会跳到指定标号处继续往下执行。

另外, 如果在 `next` 参数后再添加一个 `test="m"` 参数, 脚本仅在变量 `[$m]` 有值时才会跳到指定标号, 这个特性可以用来实现某些特殊功能, 比如允许你检查收到的包中是否存在指定的字符串从而改变脚本执行流程。`test` 动作的判断是针对呼叫变量来进行的, 对于正则表达式, 则必须至少匹配一次; 对于布尔变量则必须为真; 对于其他变量则该变量必须要有值。

如上文介绍的 `branchc` 脚本和 `branches` 脚本, 其流程如下:



## 2) 条件分支脚本中的随机执行

为了使用 SIPp 的行为更类似人工呼叫，可以使用 SIPp 来编写“概率分支 (statistical branching)”脚本，这个功能使用属性“chance”（例如 chance="0.90"）来实现，chance 表示以 chance 所定义的机率来执行脚本。

## 6.6 SIP 认证

SIPp 支持两种 SIP 认证: Digest/MD5 (“algorithm=“MD5””)和 Digest/AKA (“algorithm=“AKAv1-MD5””)。启用认证很简单,当收到 401 或 407 消息时,在<recv>命令中添加 auth=“true”就行了,然后可以使用[authentication]关键词来进行认证了。SIPp 通过使用[authentication]关键词来计算认证消息,根据加密算法的不同, authentication 关键词的参数也不相同:

- Digest/MD5(例: [authentication username=joe password=schmo])
  - **username:** username: 如果没有指定用户名,则使用命令行参数 '-au' (authentication username) or '-s' (service) 指定。
  - **password:** password: 如果没有指定密码,则使用命令行参数 '-ap' (authentication password) 指定
- Digest/AKA: ( 例如 : [authentication username=HappyFeet aka\_OP=0xCDC202D5123E20F62B6D676AC72CB318 aka\_K=0x465B5CE8B199B49FAA5F0A2EE238A6BC aka\_AMF=0xB9B9])
  - **username:** username: 如果没有指定用户名,则使用命令行参数 '-au' (authentication username) or '-s' (service) 指定。
  - **aka\_K:** Permanent secret key. If no aka\_K is provided, the "password" attributed is used as aka\_K.
  - **aka\_OP:** OPerator variant key
  - **aka\_AMF:** Authentication Management Field (indicates the algorithm and key in use)

假如你想针对不同的呼叫使用不同的认证,可以使用 CSV 文件如:

SEQUENTIAL

User0001:[authentication username=joe password=schmo]

User0002:[authentication username=john password=smith]

User0003:[authentication username=betty password=boop]

然后 XML 脚本文件如下 ([field1]会被 auth 替换)

<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>

例如:

## Example:

```
<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=fouser]
    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>
```

## 6.7 初始场景

一些特殊的复杂的脚本可能需要在脚本刚开始运行时设置一些全局变量，SIPp 的初始场景（initialization stanza）功能可以实现这个功能。创建初始场景的方法很简单，只要在命令 `<init>` 和 `</init>` 和之间插入一些 `<nop>` 和 `<label>` 命令就行了。在初始场景中的变量，仅用于全局变量，并不会应用于呼叫。举个例子，下面的初始场景功能为如果变量 `$THINKTIME` 没有设置值，则将该变量的值设置为 1：

```
<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>
```

该功能可以配合命令行参数 `-set` 来使用以使用复杂的功能。

## 7. 运行界面

SIPp 含有多个监控界面，在运行过程中按 1-9 键进行切换。

- 键 1：脚本视图，显示脚本的流程和一些重要的信息



```

ocadmin@vista:~/sipp.2004-07-05
----- Scenario Screen ----- [1-4]: Change Screen --
Call-rate(length)  Port  Total-time  Total-calls  Remote-host
190 cps(0 ms)     5061    50.01 s     8586         127.0.0.1:5060(UDP)

190 new calls during 1.000 s period      3 ms scheduler resolution
205 concurrent calls (limit 570)         Peak was 232 calls, after 6 s
0 out-of-call msg (discarded)
1 open sockets

Messages  Retrans  Timeout  Unexpected-Msg
INVITE ----->      8586     0         0
100 <-----      0         0         0
180 <-----      8586     0         0
200 <----- B-RTD  8586     68        0
ACK ----->      8586     68
[ 1000 ms]
BYE ----->      8381     0         0
200 <----- E-RTD  8381     0         0

----- [ + | - | * | / ]: Adjust rate ---- [ q ]: Soft exit ---- [ p ]: Pause traffic -----

```

- 键 2，统计视图，显示主要的统计信息，其中“Cumulative”栏表示从 SIPP 开始运行以来的所有统计情况，“Periodic”表示每次刷新时间里的统计情况（使用 `-f` 参数可以调整刷新频率）。

```

ocadmin@vista:~/sipp.2004-07-05
----- Statistics Screen ----- [1-4]: Change Screen --
Start Time          | 2004-07-13 17:24:08
Last Reset Time     | 2004-07-13 17:26:05
Current Time        | 2004-07-13 17:26:06
-----+-----+-----
Counter Name        | Periodic value | Cumulative value
-----+-----+-----
Elapsed Time        | 00:00:00:999   | 00:01:58:019
Call Rate           | 26.026 cps     | 24.886 cps
-----+-----+-----
Incoming call created | 0              | 0
OutGoing call created | 26             | 2937
Total Call created   |                | 2937
Current Call         | 0              |
-----+-----+-----
Successful call      | 26             | 2937
Failed call          | 0              | 0
-----+-----+-----
Response Time       | 00:00:00:000   | 00:00:00:000
Call Length         | 00:00:00:000   | 00:00:00:000
-----+-----+-----
----- [ + | - | * | / ]: Adjust rate ---- [ q ]: Soft exit ---- [ p ]: Pause traffic -----

```

- 键 3：区间视图，显示响应时间和呼叫长度的分布区间，呼叫长度和响应时间的值在脚本中可以指定。

```

ocadmin@vista:~/sipp.2004-07-05
----- Repartition Screen ----- [1-4]: Change Screen --
Average Response Time Repartition
  0 ms <= n < 1000 ms : 0
 1000 ms <= n < 1040 ms : 385
 1040 ms <= n < 1080 ms : 388
 1080 ms <= n < 1120 ms : 384
 1120 ms <= n < 1160 ms : 382
 1160 ms <= n < 1200 ms : 382
                   n >= 1200 ms : 190
Average Call Length Repartition
  0 ms <= n < 1000 ms : 0
 1000 ms <= n < 1100 ms : 946
 1100 ms <= n < 1200 ms : 975
 1200 ms <= n < 1300 ms : 190
 1300 ms <= n < 1400 ms : 0
                   n >= 1400 ms : 0
----- [+|-|*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----

```

➤ 键 4，变量视图，显示变量信息

```

ocadmin@vista:~/sipp.2004-07-05
----- Variables Screen ----- [1-4]: Change Screen --
Action defined Per Message :
=> Message[3] (Receive Message) - [3] action(s) defined :
--> action[0] = Type[1] - where[Full Msg] - checkIt[1] - varId[1]
--> action[1] = Type[1] - where[Full Msg] - checkIt[1] - varId[2]
--> action[2] = Type[1] - where[Header-Contact:] - checkIt[1] - varId[6]

Setted Variable Liste :
=> Variable[1] : setted regexp[([0-9]{1,3}\.){3}[0-9]{1,3}: [0-9]*]
=> Variable[2] : setted regexp[([0-9]{1,3}\.){3}[0-9]{1,3}: [0-9]*]
=> Variable[6] : setted regexp[.*]
----- [+|-|*|/]: Adjust rate ---- [q]: Soft exit ---- [p]: Pause traffic -----

```

## 8. 传输层模式

SIPp 含有多个传输层模式，其默认模式为“UDP mono socket”模式

### 1) UDP mono socket

在 UDP mono socket 模式中（命令行参数-t u1），仅打开一个 IP/UDP socket 与远端系统进行通信，所有呼叫均使用这个套接字，这个模式一般用来模拟两个 sip 服务器的场景。

### 2) UDP multi socket

在 UDP multi socket 模式中（命令行参数-t un），SIPp 为每个新的呼叫分配一个 IP/UDP socket。这个模式一般被用来模拟多个 UA 来呼叫一个 sip 服务器。

### 3) UDP 每 socket 每 IP

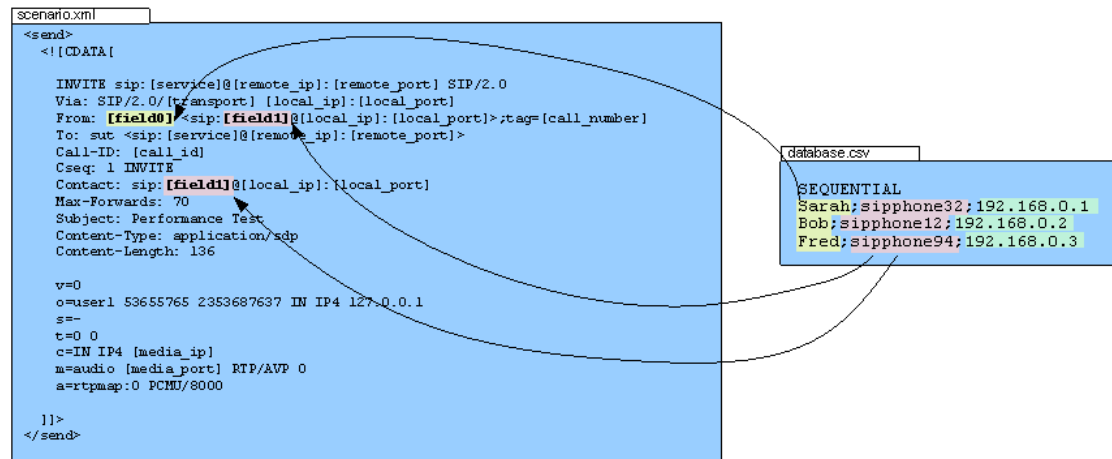
在 UDP 每 socket 每 IP 模式中（命令行参数-t ui），对于 inf 文件中给定的每个 IP 地址均会分别打开一个 UDP/IP socket。在使用参数-t ui 后，还必须指定 inf 文件中用到的用来表示 ip 地址的 field（用命令行参数“ip\_field x”来指定），这种模式需分两种情况讨论：

- 客户端侧：使用-t ui 参数后，SIPp 会对发起的每个呼叫使用不同的 ip 地址，ip 地址为 inf 文件指定，用来替代脚本中的[local\_ip]
- 服务端侧：使用-t ui 参数后，服务端会监听本地所有 ip 地址，所以需将脚本中的 [local\_ip] 替换成 [server\_ip]

该模式一般用来每用户每个 ip 地址的场景。如下图所示，客户端的脚本执行命令应形如：

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

通过使用此命令，会依次从 IP 地址 192.168.0.1, 192.168.0.2, 192.168.0.3, 192.168.0.1, ... 发起新的呼叫。



### 4) TCP mono socket

参数为-t t1，类似 UDP mono

### 5) TCP multi socket

参数为-t un，类似 UDP multi

6) 其余 TLS, SCTP 均支持 mono socket 和 multi socket 模式，用法类似。

### 7) TCP 重连接

SIPp 会处理 TCP 重新连接。为防止 TCP socket 丢失，SIPp 会尝试重新连接，在命令行使用如下参数来使用 tcp 重连接功能：

- -max\_reconnect: 设置尝试重新连接的最大次数
- -reconnect\_close true/false: 设置是否关闭呼叫当重新连接时
- -reconnect\_sleep int: 指定在关闭呼叫并重新连接之前的休眠时间，单位为毫秒

### 8) IPv6 支持

SIPp 支持 IPv6，要使用 IPv6，只需使用-i 参数指定本地的 IPv6 地址即可，命令举例如下：

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

## 9. 媒体处理

SIPp 主要用于产生 SIP 消息，对媒体的支持有一些限制，主要有如下一些处理媒体的方式。

### 1) RTP 回送 (RTP echo)

RTP 回送功能使 SIPp 能够监听一或两个（使用 `-mi` 或 `-mp` 命令指定）本地端口用来处理 RTP 媒体，在这两个端口上接收到的所有数据包都会被回送给发送端。在比这个端口高 2 的端口上收到的 RTP/UDP 包也会被回送给发送端。

### 2) RTP 流传送

SIPp 能够通过 RTP 传送格式为 PCMA, PCMU 或 G729 的音频文件

### 3) PCAP 播放功能

PCAP 播放功能使用 PCAP 库来向目标设备重放预先录制好的 RTP 流文件，RTP 流文件可以使用由 `wireshark` 或者 `tcpdump` 录制好的，使用这种方式你可以实现如下功能：

- 播放任意的 RTP 流（包括语音，视频，语音+视频，DTMF/RFC2833, T38 传真等）
- 支持任意编码格式（SIPp 并不处理解码）
- 根据所录制的 RTP 流，SIPp 能精确地模拟任意 SIP 设备的行为
- 能够产生与所抓取的网络包完全一致的消息

可以参考 SIPp 内置的脚本 `UAC_with_media(uac_pcap)`

## 10. 退出码

为了更易于自动化测试，SIPp 退出时会使用如下退出码：

- 0: 所有呼叫均成功
- 1: 至少有一个呼叫失败
- 97: 由于内部命令退出，呼叫可能已经处理。也有可能是全局超时（参考 `-time_global` 选项）
- 99: 正常退出，没有处理呼叫
- -1: 致命错误

不同的系统查看退出码的方法不一样，你可以使用“`echo ?`”命令查看退出码

## 11. 统计

### 1) 响应时间统计

SIPp 提供了一套机制来收集并统计响应时间。每一个响应时间在两个 SIPp 命令（`send`, `recv` 或者 `nop`）之间进行计算。可以使用 `start_rtd` 开启一个计时器并使用 `rtd` 结束它。在 SIPp 运行界面可以按 3, 6, 7, 8 或 9 来查看计时器的值，另外还可以使用 `-trace_stat` 命令来将统计结果保存到 CSV 文件中。

### 2) 可用的计数器

参数 `-trace_stat` 转储所有统计信息到文件“脚本名\_进程号.csv”中，可用的统计类型如

下:

- **StartTime:** 脚本开始运行的时间日期
- **LastResetTime:** Date and time when periodic counters where last reseted.
- **CurrentTime:** 当前统计的日期和时间
- **ElapsTime:** 已运行的时间
- **CallRate:** 呼叫速率, 每秒
- **IncomingCall:** 呼入的呼叫数量
- **OutcomingCall:** 呼出的呼叫数量
- **TotalCallCreated:** 总共生成的呼叫个数
- **CurrentCall:** 正在进行的呼叫
- **SuccessfulCall:** 成功的呼叫
- **FailedCall:** 失败的呼叫
- **FailedCannotSendMessage:** 不能发送消息的呼叫 (传输层问题)
- **FailedMaxUDPReTRANS:** 重试达到最大次数导致的失败呼叫
- **FailedUnexpectedMessage:** 未如脚本所料的错误呼叫
- **FailedRegexpDoesntMatch:** 由于正则表达式应该匹配而没有匹配而造成失败的呼叫
- **FailedRegexpShouldntMatch:** 由于正则表达式不应该匹配而匹配造成失败的呼叫
- **FailedRegexpHdrNotFound:** 由于正则表达式没有找到匹配项导致的失败的呼叫
- **FailedOutboundCongestion:** 由于 TCP 拥塞而导致失败的呼叫
- **FailedTimeoutOnRecv:** 由于接收超时而导致失败的呼叫
- **FailedTimeoutOnSend:** 由于发送超时而导致失败的呼叫
- **OutOfCallMsgs:** 不能与存在的呼叫匹配的消息个数
- **Retransmissions:** 重传输的个数
- **AutoAnswered:** 自动应答并回复 200ok 的消息的个数

### 3) 详细消息统计

使用命令行参数 `-trace_counts` 可对发送或接收的消息的个数统计进行保存。

## 12. 错误处理

### 1) 未预料的消息

- 当 SIPp 收到一个 sip 消息能与现存的呼叫进行匹配 (通过 call-id) 但是不是脚本所预料的 (unexpected) 消息, 如果没有收到 200ok 消息或者发送了 bye 但没有收到 200ok, 那么 sipp 会发送一个 cancel 并记此次呼叫为失败 (failed); 如果收到的是 4XX 或 5XX 消息, SIPp 会发送一个 ACK 消息并记录此次呼叫为失败
- 如果收到的一个 sip 消息不能与现存的呼叫进行匹配, 那么 sipp 会发送 bye 消息, 这个呼叫不会记为失败
- 非 sip 消息 sipp 会直接丢弃

### 2) 重传输机制 (仅 UDP)

SIPp 含用于 UDP 模式的重传输机制。如果需要使用重传输功能, 需要在 “send” 命令中包含 “retrans” 属性。在激活了重传输功能后, 如果发送了一个 SIP 消息出去后没有收到对应的答复或者 ACK 消息, 则 SIPp 会重发这个 SIP 消息。sipp 的重传输机制符合 rfc3261 标准 (标准请参见 rfc3261, 17.1.1.2), 重传输的时间间隔对于 invite 消息和非 invite 是不同

的。<send retrans="500">: 初始一个 T1 计时器，时间为 500ms。

注意：即使在脚本中指定了 retrans 属性，你也可以使用命令行选项-nr 来全局禁用重传输机制。

### 3) 日志功能

在运行过程，SIPp 提供了多种方法记录日志。

- -trace\_msg 命令
- -trace\_shortmsg
- -trace\_err: 追踪错误消息
- -trace\_error\_codes
- -trace\_counts

另外，可以使用-ringbuffer\_size 命令限制日志文件大小。

## 13. 在线帮助

使用 sipp -h 可查看在线帮助，翻译如下：

命令用法：

sipp 远程主机 IP[:远程主机商品] [命令行参数]

用法示例：

运行内置的服务器（uas）脚本：

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

在同一台主机，运行内置的客户端（uac）脚本：

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

可用的参数如下：

参数名称	描述
<b>脚本文件参数</b>	
-sd	导出内嵌的脚本文件
-sf	加载自定义的脚本 XML 格式
-oocsf	加载 out-of-call 脚本
-oocsn	加载 out-of-call 脚本
-sn	使用默认（内置）脚本，如果不指定参数，则默认是 uac，可用的参数如下： uac: 标准 uac uas: 简单 uas regexp: 带正则表达式的 uac branchc: 分支脚本之客户端脚本 brahchs: 分支脚本之服务端脚本 以及一些默认的 3PCC 脚本： 3pcc-C-A: 控制器 A 端 3pcc-C-B: 控制器 B 端 3pcc-B: B 端 3pcc-A: A 端
<b>IP, 端口和协议参数</b>	

-t	设置传输层模式： u1: udp mono, 默认 un: udp multi ui: udp 每个呼叫每个 ip t1: tcp mono tn: tcp multi c1: u1 的压缩模式 cn: un 的压缩模式
-i	设置本地 ip 地址，用于指定'Contact:', 'Via:', and 'From:'的地址
-p	指定本地端口
-bind_local	绑定 socket 到本地地址
-ci	设置本地控制 ip 地址，用于远程控制
-cp	设置本地控制端口，默认是 8888
-max_socket	最大的并发 socket 个数
-max_reconnect	设置最大的重连接次数
-reconnect_close	是否关闭呼叫在重连接时
-reconnect_sleep	在关闭和重连接之间的等待时间 (ms)
-rsa	指定远端主机地址和端口用于发送消息
<b>SIPp 功能参数</b>	
-v	显示版本号
-bg	在后台运行 sipp
-nostdin	禁用标准输入
-plugin	加载一个插件
-sleep	在开始运行时休眠多少秒
-skip_rlimit	跳过对文件标识符限制的设置
-buff_size	设置发送和接收缓冲区大小
-sendbuffer_warn	在发送缓冲区有错时只提示告警而不是报错
-lost	设置默认的丢包率。脚本中定义的值优先级更高
-key	关键词的值，设置通用参数名为“keyword”的值为“value”
-set	设置变量的值，设置变量名为“variable”的值为“value”
-tdmmap	生成并处理 TDM 电路表（貌似是实现路由表功能），格式为 -temmap {0-3} {99} {5-8} {1-31}
-dynamicStart	变量值，设置动态 id 变量 dynamic_id 的起始值
-dynamicMax	dynamic_id 的最大值
-dynamicStep	dynamic_id 的增长步长
<b>SIPp 呼叫行为参数</b>	
-aa	使能 SIPp 对 INFO, UPDATE and NOTIFY 自动回 200ok
-base_cseq	设置起始 cseq 值，默认为 1
-cid_str	设定 call-id 格式，默认为： %u-%p@%s，其中 %u=call_number, %s=ip_address, %p=process_number, %%=%
-d	控制呼叫长度，为脚本中<pause/>指定值
-deadcall_wait	设置一个 call-id 的保持时间

-auth_uri	强制使用 uri 的值来作为鉴权参数，默认情况下，uri 由 remote_ip:remote_port 组成。
-au	设置用于认证的用户名，默认由-s 指定
-ap	设置用于认证的密码，默认是 password
-s	设置请求 uri 中的用户名部分，默认是 service
-default_behaviors	设置 Slpp 的默认行为，可能的值为： all: 使用所有默认行为能力 none: 不使用任何行为 bye: 对中止的呼叫发送 bye abortunexp: 在收到 unexpected 消息时中止呼叫 pingreply: 对 ping 请求回应。 可以叠加使用，如 all,-bye
-nd	no default，禁用默认行为，具体将禁用以下行为： ➢ udp 重传输超时后发送 bye 或者 cancel ➢ 对于 unexpected 的 cancel 发送 200ok ➢ 对 unexpected 的 bye 发送 200ok ➢ 对 unexpected 的 PING 消息发送 200ok ➢ 对于任意的 unexpected 的消息发送 bye 或者 cancel 并终止呼叫。
-pause_msg_ign	忽略在脚本中定义的暂时时间内收到的消息
<b>注入文件（Injection file）选项</b>	
-inf	在呼叫过程中，从一个外部 CSV 文件引入值到脚本中去。文件的第一行表明数据的读取顺序。
-inindex	根据文件中的字段（field）来创建索引。例如：-inf users.csv -inindex users.csv 0 表示根据第一个键值来创建索引。
-ip_field	设置从注入文件中哪个字段读取 ip 地址
<b>RTP 选项</b>	
-mi	设置本地媒体的地址，默认为主 ip 地址
-rtp_echo	启用 rtp 回送功能。将由 -mp 参数指定的端口收到的 rtp 流回送给原发送者。
-mb	设置 rtp 回送 buffer 大小，默认为 2048
-mp	设置本地 rtp 回送地址，默认为 6000
-min_rtp_port	rtp 端口范围最小值
-max_rtp_port	rtp 端口范围最大值
-rtp_threadtasks	每线程的 rtp
-rtp_buffsize	设置 rtp socket 的发送和接收 buffer 大小
<b>呼叫速率选项</b>	
-r	设置呼叫速率（设置多少个呼叫每秒）
-rp	设置呼叫速率的周期，默认是 1000 毫秒。例如-r 7 -rp 2000 表示 2 秒中 7 个呼叫
-rate_scale	控制按键+、*/的变化跨度
-rate_increase	控制每单位时间后呼叫的增长速率。这个功能允许周期地递增呼叫速率，例如-rate_increase 10 -fd 10s 表示呼叫速率每 10



	秒增加 10 个
-rate_max	如果指定了 -rate_increase, 则这个参数表示当速率达到该值时不再增加
-no_rate_quit	如果设置了 -rate_increase, 当达到了最大速率时不退出 sipp
-l	设置最大的并发呼叫量
-m	设置最本最大的呼叫个数, 当 sipp 达到该指定值会自动退出
-users	指定同时进行的呼叫个数
<b>重传输与超时选项</b>	
-recv_timeout	全局参数, 指定消息接收的超时时间, 单位为毫秒
-send_timeout	全局参数, 指定消息发送的超时时间, 单位为毫秒
-timeout	Sipp 在设置的时间后退出
-max_retrans	指定最大的重传次数
-max_invite_retrans	指定 invite 消息最大的重传输次数
-max_non_invite_retrans	指定非 invite 消息最大的重传输次数
-nr	禁用重传输功能
-rtcheck	指定重传输检测模式, 默认为 full, 还有 loose 可供选择
-T2	全局设置 T2 定时器
<b>三方呼叫控制选项</b>	
-3pcc	使用 3pcc 模式
-master	使用主模式
-slave	使用从模式
-slave_cfg	指定 3pcc 的配置文件
<b>性能和软件狗选项</b>	
-timer_resol	设置计时器精确度, 默认为毫秒。这个选项影响统计的精确度。较小的值能够提高精确度但是会消耗更多的 cpu, 默认为 10ms
-max_recv_loops	设置每一个循环可接收的最大消息数。提高这个值可用来测试更大的话务量, 默认是 1000
-max_sched_loops	设置每个循环(loop)最大的呼叫数量, 提高这个值可用来测试更大的话务量, 默认是 1000
-watchdog_interval	设置软件狗检查的间隔, 默认为 400 秒
-watchdog_reset	如果在此参数设定的时间内, sipp 的软件狗还没有被触发过, 则复位最大的触发计数器, 默认是 10 分钟
<b>消息跟踪、日志记录和测试统计选项</b>	
-f	设置在屏幕上的统计报告频率, 默认为 1 秒
-trace_stat	转储所有统计记录到文件名为“脚本名称_进程号.csv”文件中, 可以使用命令行参数“-h stat”查看统计文件中各列内容的详细描述
-stat_delimiter	设置统计文件的分隔符
-stf	设置统计文件的文件名
-fd	设置转储的统计文件的更新(写入)频率, 默认是 60 秒
-peridoic_rtd	每更新一次统计文件复位区间计时器
-trace_msg	将发送和接收的 sip 消息保存在消息日志文件<scenario file

	name>_<pid>_messages.log 中
-message_file	设置消息日志文件的文件名
-message_overwrite	覆盖消息日志文件，默认覆盖
-trace_shortmsg	以短格式记录接收和发送的 sip 消息，并保存在文件中
-shortmessage_file	设置短消息格式文件的文件名
-shortmessage_overwrite	覆盖短消息日志文件
-trace_counts	存储自定义的消息计数到 csv 文件中
-trace_err	跟踪并记录所有意外的消息到错误日志文件<scenario file name>_<pid>_errors.log 中
-error_file	设置错误日志文件的文件名
-trace_error_codes	记录未预料的 sip 消息的响应代码到日志文件中
-trace_calldebug	保存被非法中断的呼叫的调试消息到日志中
-trace_screen	保存退出 sipp 时的最后的统计屏幕到日志中
-trace_rtt	跟踪并保存所有响应时间到日志文件中
-rtt_freq	设置保存响应时间到日志文件的频率，默认是 200 个呼叫保存一次
-trace_logs	跟踪日志动作（log action）到日志文件中
-ringbuffer_files	设置循环保存的日志文件的个数的最大值
-ringbuffer_size	设置保存的日志文件的大小的最大值
-max_log_size	设置所有保存的日志文件的个数的上限

## 四、使用 SIPp 做性能测试

### 1. 使用 SIPp 做性能测试的建议

SIPp 本身就是被设计用来做性能测试，通常情况下能够满足中等并发中等呼叫速率的测试场景。如果测试时满足下列条件，则可以用 SIPp 来测试高并发、高呼叫速率的场景：

- 使用 Linux 系统来达到高并发。Windows 版本(使用 CYGWIN)性能不够高
- 尽量不使用消息跟踪功能，消息跟踪（如-trace\_msg, -trace\_logs 等）仅用于调试而不是性能测试
- 理解 SIPp 的内部调度机制，并使用命令行参数-timer\_resol, -max\_recv\_loops 和 -max\_sched\_loops 来调整 SIPp，使其运行在最佳状态。

### 2. SIPp 内部调度机制

SIPp 使用单线程、event-loop 架构，该架构使它能够处理很高的 SIP 消息流量。Sipp 的 event loop 跟踪各种任务，大部分任务便是脚本中定义的呼叫；另外一些是特殊的任务：屏幕刷新任务，统计更新任务，新呼叫任务和软件狗任务。SIPp 的主执行 loop 由以下构成：

- 唤醒已经过期的任务

- 运行任务直到达到 `max_sched_loop` 最大值
- 轮流处理每一个 `socket`, 从各个 `socket` 中读取不超过 `max_recv_loops` 指定个数的消息

SIPp 持续地执行 `loop` 直到遇到让它停止执行的条件, 如运行过程中按 `q` 键, 或达到最大的呼叫量等等

几个可以对 `sipp` 的性能进行微调的命令行参数如下:

- 定时器精确度 (`timer_resol`): 默认的定时器精确度是 `1ms`, 这也是 `sipp` 所能支持的最高的精确度, 如果调大了精确度会导致 SIPp 处理不过来
- `max_recv_loops` 和 `max_sched_loops`: `max_recv_loops` 是 `sipp` 在单位时间内所能读取的最大消息量, `max_sched_loops` 是单位时间内所能处理的最大呼叫量, 在呼叫量很大的测试时可以调大这两个值, 不过要注意, 调大这两个值后会对 `cpu` 的占用带来较大的影响
- `watchdog_interval`, `watchdog_minor_threshold`, `watchdog_major_threshold`, `watchdog_minor_maxtriggers`, and `watchdog_major_maxtriggers`: `watchdog` 是用来监视 `sipp` 是否已经挂死的, 可以适当调大这些值

## 五、有用的工具

### 1. JEdit

JEdit (<http://www.jedit.org/>)是一个 GNU GPL 的文本编辑器, 使用 Java 编写, 有多个平台的版本。当你把 DTD (`sipp.dtd`) 文件与你编写的 XML 脚本文件放在同一个目录下, JEdit 能够对 SIPp 脚本进行语法检查, 功能非常强大。

### 2. Wireshark/tshark

Wireshark (<http://www.wireshark.org/>)是一个遵循 GNU GPL 的协议分析器, 以前叫做 Ethereal, 它支持 SIP, SDP 和 RTP。

### 3. SIP callflow

当跟踪 SIP 呼叫时, 可以使用 `wireshark` 来查看一个呼叫流程, 不过 `callflow` 有图形界面查看, 非常方便: <http://callflow.sourceforge.net/>。

## 六、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](mailto: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>

## 七、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】

黄龙舟 【中文翻译】

Copyright © 2004-2014 作者保持所有权利

最后一次编辑: 3/1/2014 14:35:21

对于网站的建议或反馈请联系: [Rob Day](#)

对于翻译的任何建议请发给邮箱 [hlz\\_2599@163.com](mailto:hlz_2599@163.com)，错误之处，请您批评更正，谢谢