Automated Load Testing for SIP Applications

Serge Kruppa
Director of Engineering

Astricon 2008, Glendale AZ
About LiveVox

- Leading provider of hosted VoIP dialing solutions, with integrated ACD and IVR, to the credit and collections industry

- Combines patented technology with industry expertise to address clients operational challenges and compliance requirements

- Replaces premise-based dialers and doubles agent productivity at the lowest cost of ownership in the market

- Use of Hosted VoIP Dialer increased 400% in the past year (May 2008)

- Based in San Francisco with 80 FTEs around the world
Why load testing?

- To meet business objectives
  - Customer satisfaction
  - Revenue growth
- How to achieve them?
  - Minimize downtime
  - Maximize scalability
  - Offer consistent quality
- Proactive approach to capacity management
  - Find possible bottlenecks
  - Forecast scalability limits
  - Address issues early and effectively
Different types of load

• Load triggered by internal or external events
  – SIP messages
  – RTP packets
  – HTTP requests

• Load created on platform components
  – Internal H/W: CPU, memory, NICs, hard disk
  – External H/W: LAN/WAN, dependent systems
  – Applications
What will break?

- Excessive load will create
  - Real-time issues (deadlocks, race conditions)
  - Network overload (packet loss, jitter, etc.)
  - Hard disk capacity issues (running out of space)
  - Memory failures (application crash due to memory leaks, malloc() failures, etc.)
  - CPU overload
- Affecting call quality
  - Choppy audio, call delays or failures, loss of data
- Load and quality are correlated
  - Maintain sufficient headroom, timely upgrades
What to measure?

- Monitor
  - CPUs
  - Memory
  - SIP messages timestamps
  - RTP jitter and packet loss
  - Application logs
- Place live calls (MOS human validation)

Often symptoms of network issues
Real world scenarios

• **Contact center call recording capacity**
  - Simulate several SIP UAS (agents and debtors) receiving calls made by the Media Server in response to HTTP requests. Agent UAS instances must play an audio file that will be recorded by the Media Server.

• **SIP proxy performance**
  - Simulate multiple SIP UAC (Media Servers) making calls to a high capacity UAS (carrier) through a SIP proxy with load balancing functionality.

• **Contact center outbound call pacing**
  - Simulate several agents that will log in to a predictive dialer via a Web interface and then receive a call (SIP UAS) while debtor (other UAS) calls are bridged to them.
Open source tools used

• **Load generators**
  – SIP : SIPp
  – HTTP : JMeter

• **SIP packet capture and analysis**
  – Tcpdump (Linux) or WinPCAP (Windows)
  – Wireshark

• **Load monitoring and reporting**
  – Cacti
  – Top, vmstat, sar, etc.
Downloads

► SIPp
http://sipp.sourceforge.net/

► Wireshark
http://www.wireshark.org/

► JMeter
http://jakarta.apache.org/jmeter/
Test architecture

- ACD100 ACD Server
- SIPp
- TM0122 Media Server 2
- TM0120 Media Server 1
- Gigabit Ethernet Switch
- Topdump
- SIP Proxy Application
- SIP100 SIP Proxy Server
- REC100 Call Recording Server
- Java Call Recording Application
- JMeter
- Wireshark
Call recording scenario

- Concepts covered
  - SIPp and JMeter introduction
  - UAS scenario customization

- Network diagram
Introduction to SIPp

• Requires detailed knowledge of SIP
• Scenarios are expressed in XML
• "SIPp is a free Open Source test tool and traffic generator for the SIP protocol"
• Has several user agent scenarios
  – uac
  – uas
  – uac_pcap, etc.
• Can use custom scenario files
SIPp features

- Support for both IPv4 and IPv6
- RTP media support (PCAP)
- Call rate distributions
  - Fixed, uniform, exponential
- High performance and reliable
- Complex scenarios are possible
- Statistics
UAC and UAS default scenarios

► UAC: sipp –sn uac 127.0.0.1
► UAS: sipp –sn uas
UAC and UAS custom scenarios

- `sipp -sd uas > uas.xml`
  - UAS: `sipp -sf uas.xml`
  - `uas.xml`

- `sipp -sd uac > uac.xml`
  - UAC: `sipp -sf uac.xml 127.0.0.1`
  - `uac.xml`
SIPp command line options

Usage

- sipp remote_host[:remote_port][options]

Regular options

- -sn name : use a default scenario
- -sd filename : Dumps a default scenario
- -sf filename : load an alternate XML scenario file
SIPp advanced options

Advanced options

- `-r rate` : set the call rate (default = 10)
- `-rp period` : rate period in milliseconds (default = 1000ms)
  - Example: `--r 10 --rp 1000`, 10 calls every second
- `-m calls` : stop and exit when set number of calls are processed
- `-l` : set the maximum number of simultaneous calls
- `-rate_increase` : specify the rate increase every --fd seconds
- `-nr` : disable retransmission in UDP mode
- `-rsa host[:port]` : set the proxy server IP address
- `-i` : set local IP address
- `-p` : set local UDP port
Actual call recording scenario

- UAS 1: `sipp -sf debtor.xml -i 10.10.100.122 -p 5061 -rsa 10.10.100.120:5060`
- UAS 2: `sipp -sf agent.xml -i 10.10.100.122 -p 5062 -rsa 10.10.100.120:5060`
SIPp scenario screen

- Default screen (press 1)

```
Scenario Screen --- 11:Y1: Change Screen ---

<table>
<thead>
<tr>
<th>Port</th>
<th>Total-time</th>
<th>Total-calls</th>
<th>Transport</th>
</tr>
</thead>
<tbody>
<tr>
<td>5061</td>
<td>151.01 s</td>
<td>0</td>
<td>UDP</td>
</tr>
</tbody>
</table>

0 new calls during 1.000 s period
0 calls
0 Running, 0 Paused, 0 Woken up
0 dead call msg (discarded)
3 open sockets
0 total RTP pkts sent
0.000 last period RTP rate (kB/s)

<-------- INVITE Messages Retrans Timeout Unexpected-Msg
<-------- 180   0   0   0   0
<-------- 200   0   0   0   0
--------> ACK   E-RTP1 0   0   0   0
[     3:15] Pause        0   0   0
<-------- BYE        0   0   0   0
--------> 200        0   0   0   0

---------------------- Sipp Server Mode ----------------------
```

- Exit codes:
  - 0 (zero) indicate that all calls were successful
  - 1 indicates that at least one call failed
  - 97 indicates an abnormal exit on internal command
  - 99 indicating a normal exit without processing calls
  - -1 used to indicate a fatal error
Scenario keywords mapping

- UAS 2: sipp -sf agent.xml -i 10.10.100.120 -p 5062 -rsa
  10.10.100.50:5060
- Agent.xml

```xml
<scenario name="Agent UAS responder">
  <recv request="INVITE" crlf="true">
  </recv>
  <send>
    <![CDATA[
      SIP/2.0 180 Ringing
      [last_Via:]
      [last_From:]
      [last_To:];tag=[pid]SIPpTag01[call_number]
      [last_Call-ID:]
      [last_CSeq:]
      Contact: <sip: [local_ip]: [local_port]:transport=[transport]>
      Content-Length: 0
    ]]>
  </send>
</scenario>
```

The 'last_*' keyword is replaced automatically by the specified header if it was present in the last message received.
Main SIPp elements

- `<send>`, send a SIP request or response
- `<recv>`, receive a SIP request or response
- `<pause>`, pause the scenario
- `<nop>`, no operation
- `<label>`, branch within scenario (“goto”)

Playing out audio

<!-- Play a pre-recorded PCAP file (RTP stream) -->
<nop>
  <action>
    <exec play_pcap_audio="pcap/g711u.pcap"/>
  </action>
</nop>

<!-- Pause 195 seconds, which is approximately -->
<!-- the duration of the PCAP file -->
<pause milliseconds="195000"/>

Can be any RTP packet capture made by Wireshark or tcpdump (including DTMF)
Note: the action is non-blocking

Blocking pause while the audio is played out
**Regular expressions**

**Problem:** the Media Server expects the BYE message To: and From: headers to contain correct URIs, as set when the session was created (INVITE)

**Solution:** retrieve the To: and From: header URIs using a regular expression and re-use these URIs in the BYE message

```xml
<recv request="ACK" rtd="true" crlf="true">  
  <action>  
    <ereg regexp=".*" search_in="hdr" header="From:" check_it="true" assign_to="1"/>  
    <ereg regexp=".*" search_in="hdr" header="To:" check_it="true" assign_to="2"/>  
  </action>  
</recv>  
<send retrans="500">  
  <![CDATA[  
    BYE sip:[service]@[remote_ip][:remote_port] SIP/2.0  
    Via: SIP/2.0/[transport] [local_ip][:local_port];branch=[branch]  
    From: [$2]  
    To: [$1]  
    [last_Call-ID:]  
  ]]>
```
Introduction to JMeter

- Apache project
- JMeter is a 100% pure Java desktop application designed to load test functional behavior and measure performance of Web as well as other types of applications
- A test plan describes a series of steps JMeter will execute when run.
Call recording JMeter test plan
Test sequence and results

► Test execution
  ► **Start UAS 1**: sipp –sf debtor.xml –i 10.10.100.122 –p 5061 –rsa 10.10.100.120:5060
  ► **Start UAS 2**: sipp –sf agent.xml –i 10.10.100.122 –p 5062 –rsa 10.10.100.120:5060
  ► Run Jmeter CallRecording.jmx test plan

► Test results
  ► Reached 85% CPU usage for 750 simultaneously recorded calls
SIP proxy scenario

- Concepts covered
  - Use of tcpdump and Wireshark
- Network diagram
Test sequence

Test execution

- **Start tcpdump on SIP proxy**: tcpdump udp port 5060 -s 1600 -vv -w test.pcap
- **Start UAC**: sipp -sn uac 10.10.100.50 -r 40 -rp 1000
- **Start UAS**: sipp -sn uas
Test results analysis - packets

TCPdump packet capture analysis

► Launch Wireshark
► Open .pcap file
► Choose VoIP call statistics
► Select one of the calls from the list
► Click on Graph
► Review the SIP message graph
Test results analysis - delays

A Wireshark IO graph can be very useful to see delays at a glance.
Test results

- We believe delays were caused by the NIC or Linux NIC driver of the blade server that was used for the SIP proxy.
- Switching the SIP proxy to a regular (non-blade) server fixed the delay problem.
- This is still being investigated by Livevox.
Call pacing scenario

- Concepts covered
  - Use of JMeter to simulate agent behavior

- Network diagram

New step 1 compared to the original Call Recording test plan
What is call pacing?

- **Call pacing** is the processing logic which determines the rate at which calls are made by a dialer in a call center outbound campaign.
- The timing of dial attempts is dependent on agent availability and other real-time factors.
- Selecting a pacing algorithm implies a tradeoff between agent productivity and the quality of interactions with customers.
- Poor pacing algorithms drastically increase call abandonment and are a nuisance to customers.
What do we want to achieve?

- Simulate the interactions between the agents and the Livevox Voice Portal
  - Campaign login/logout
  - Ready/not ready status change
- Create converged SIPp and JMeter test plans that mimic real-life call pacing challenges
  - Change of shift in the call center
  - Dialing lists of non-homogeneous quality
- Review key performance indicators via the Livevox Voice Portal real-time and historical reports
The Livevox Voice Portal

- Predictive call pacing has been selected for that campaign
Provisioning test agents

Please enter Agent Information

Logon ID: DAGENT
First Name: Demo
Last Name: Agent
Password: 1234
Wrapup Time: -1
Phone Number/Extension: 1166/16009
Home Agent: Yes
Active Agent: Yes

Skill:
- 3-Quick Connect
- 2-Quick Connect
- 2-Agent Registration
- 1-Agent Registration with Team Codes
- 1-Quick Connect
- 1-Agent Registration

Submit | Cancel
The normal agent login process
The agent login JMeter test plan

HTTP Request HTTPClient

Name: LoginAgent
Comments:

Server Name or IP: acd100

Path: /VirtualACD_1.6.0/AgentAgent

Method: GET

Redirect Automatically

Send Parameters With the Request:

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Encode?</th>
<th>Include Eq</th>
</tr>
</thead>
<tbody>
<tr>
<td>skill</td>
<td>9013</td>
<td></td>
<td></td>
</tr>
<tr>
<td>extension</td>
<td>4156716009</td>
<td></td>
<td></td>
</tr>
<tr>
<td>password</td>
<td>1234</td>
<td></td>
<td></td>
</tr>
<tr>
<td>client_id</td>
<td>9012</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Send a File With the Request:

Filename:

Value for "name" attribute:

MIME Type:

Optional Tasks

- Retrieve All Embedded Resources from HTML Files
- Use as Monitor

Embedded URLs must match:
The “virtual” agent has logged in and set itself to the ACD “Ready” state.

At the same time, the SIPp agent.xml script has received a call from the Livevox platform.

The agent is online and ready to receive calls.
Test sequence and results

Test execution

- **Start UAS 1**: sipp –sf debtor.xml –i 10.10.100.122 –p 5061 –rsa 10.10.100.120:5060
- **Start UAS 2**: sipp –sf agent.xml –i 10.10.100.122 –p 5062 –rsa 10.10.100.120:5060
- Run Jmeter AgentLogin.jmx test plans
- Run Jmeter CallRecording.jmx test plans

Test results

- The Livevox call pacing algorithm delivers solid results for all the test scenarios considered, according to several key metrics
  - Avg Ready Time (average amount of time agents spend waiting for a call)
  - % talk, % wrap and % ready times (eg. % talk = total talk / [talk + wrap + ready])
  - Abandon rate (Operator Transfer Failed / Attempted Operator Transfers)
- Please contact skruppa@livevox.com for more information
Questions?

- Please feel free to contact us for more information

Serge Kruppa
skruppa@livevox.com
http://www.livevox.com

THANK YOU!