sipperf Documentation

Release 0.1

Rob Day

May 02, 2016

sipperf is a performance-testing tool for SIP, a VoIP (Voice-over-IP) communication protocol.

It's free software licensed under the 3-clause BSD license.

Quickstart

Create a `users.csv` file in the format SIPURI,USERNAME,PASSWORD:

```
sip:1234@example.com,1234@example.com,secret
sip:1235@example.com,1235@example.com,secret
```

Then run `./sipperf --target sip:localhost --rps 5 --cps 1 --max-calls 1`

You should see output similar to the following:

```
2 successful registers
0 failed registers
0 calls initiated
0 calls successfully set up
0 calls failed
```

1.1 Command-line options

```
sipperf - a command-line SIP performance testing tool

Usage:
  sipperf --target sip:1.2.3.4 [options]
  sipperf --help
  sipperf --version

Options:
  -h --help            Show this screen.

  --version            Show version.

  --target=<sipuri>    Outbound proxy for traffic.

  --users-file=<file>  Path to a CSV file containing URIs, usernames and passwords for
                       each user [default: ./users.csv]

  --rps=<n>            Registrations-per-second rate for initial REGISTERs [default: 10].

  --cps=<n>            Calls-per-second rate after initial registration phase [default: 10].

  --max-calls=<n>      Maximum calls to make before exiting [default: 100].
```

1.2 Performance tips

Here are some tips on getting the best performance out of sipperf.

1.2.1 DNS

sipperf doesn't have any built-in DNS caching, so it does DNS lookups for each call, and may be slowed down by waiting for DNS responses from the network.

Two tips for getting around this problem are:

- install dnsmasq - this adds a local DNS server that caches DNS lookups, giving faster response times
- use SIP URIs that minimise the number of DNS lookups - "sip:example.com" will need three or four DNS lookups (NAPTR, one or two SRV lookups, and an A record lookup), whereas "sip:example.com:5060;transport=tcp" will just need the A record lookup - and "sip:1.2.3.4" won't need any

1.2.2 Scaling to multiple cores

Because sipperf is a single-threaded process, it doesn't make use of multiple CPU cores. If you have multiple CPU cores and want to use them, the best way to do so is to shard your performance testing:

- split your subscriber base into N equal parts, and put each part in its own CSV file
- start N instances of sipperf, each using one CSV file, and each running 1/Nth of your load

1.3 Comparison with similar projects

The main goals of sipperf are to be:

- high-performance - it should be able to handle hundreds of calls per second on a single, reasonably-powered CPU core
- realistic - it should be able to emulate
- SIP-focused - while media performance testing is important

1.3.1 SIPp

SIPp is the main other SIP performance testing tool I'm familiar with

It has a couple of disadvantages:

- it's focused on SIP dialogs rather than users - so it allows you to define a call scenario, for example (SIP INVITE through to BYE) and run that repeatedly, but not to define a mix of dialogs bound together by a user (a user sends a REGISTER, then makes some calls and sends some text messages, and it also able to answer calls)
- it's quite cluttered - it supports a lot of features which are either not directly relevant to SIP performance testing (like the ability to replay media) or quite niche (like SCTP support)

The main advantages of SIPp over sipperf are:

- it's very user-programmable -
- it's much more mature

1.4 Developing sipperf

1.4.1 Getting the source

The sipperf source code is hosted on Github at https://github.com/rkday/sipperf.

You can check the code out by running:

```
git clone --recursive https://github.com/rkday/sipperf
```

1.4.2 Building sipperf

sipperf is built with CMake:

```
cmake .
make
```

This produces a `sipperf` binary in the current directory.

1.4.3 Understanding the code

For a high-level overview, see the architecture documentation (Architecture).

After that, the header files are well-commented and should help you understand what code does what.

If something still isn't clear to you, drop me an email at rkd@rkd.me.uk!

1.4.4 Writing documentation

1.5 Architecture

sipperf uses the libre SIP library to handle all its SIP work (e.g. listening for SIP messages, message parsing, maintaining the various SIP state machines and timers that handle retransmissions).

All its work is done on a single thread - the main thread running libre_main. This thread processes both incoming SIP messages and timers - before calling into libre_main, sipperf sets up various recurring timers to do work (for example, a recurring timer to start calls).

libre is relatively efficient - it uses epoll on Linux for good signaling performance, and the custom version of libre used in sipperf includes a timer heap for good timer performance (which is being contributed back into libre).

1.5.1 Control flow

- main() does various setup, such as parsing arguments and reading the CSV file containing user details
- it then starts the user registration timer
- it then calls libre_main(0, which continues running until program exit
- periodically, the user registration timer will fire and register some users
- when all users are registered, the user registration timer starts the call timer, and stops itself
- libre takes care of re-registering users when their registrations expire - we don't need a separate timer for that

- periodically, the user registration timer will fire and set up some calls
- when the required number of calls have been set up, the call timer will start the cleanup timer and stop itself
- the cleanup timer will attempt a graceful shutdown of the libre_main event loop (e.g. waiting for all outstanding traffic to finish)
- if the cleanup timer fires too many times, it will do a hard shutdown instead
- after the cleanup timer shuts down the event loop, sipperf exits

1.5.2 Representing users

1.5.3 Statistics

Statistics reporting