Evaluating SIP Server Performance

[Extended Abstract]

Erich M. Nahum, John Tracey, and Charles P. Wright
IBM T.J. Watson Research Center
Hawthorne, NY, 10532
{nahum,traceyj,cpwright}@us.ibm.com

ABSTRACT

SIP is a protocol of growing importance, with uses for VoIP, instant messaging, presence, and more. However, its performance is not well-studied or understood. In this extended abstract we overview our experimental evaluation of common SIP server scenarios using open-source SIP software such as OpenSER and SIPp running on Linux.

We show performance varies greatly depending on the server scenario and how the protocol is used. Depending on the configuration, throughput can vary from hundreds to thousands of operations per second. For example, we observe that the choice of stateless vs. stateful proxying, using TCP rather than UDP, or including MD5-based authentication can each can affect performance by a factor of 2-4. We also provide kernel and application profiles using Oprofile that help explain and illustrate processing costs. Finally, we provide a simple fix for transaction-stateful proxying that improves performance by a factor of 10. Full details can be found in our accompanying technical report [1].

Categories and Subject Descriptors

C.2.2 [**Network Protocols**]: Applications; C.5.5 [**Computer System Implementation**]: Servers; D.4.8 [**Performance**]: Measurements

General Terms

Measurment, Performance, Experimentation

Keywords

SIP, Servers, Performance, Experimental Evaluation

1. EXPERIMENTAL SETUP

Scenarios and Configurations. We evaluate server performance for 3 core SIP server scenarios: proxying, registration, and redirection. We also examine the impact of authentication and transport protocol on performance, as well as statefulness vs. statelessness for the proxying scenario.

SIP Server Software. We evaluate OpenSER 1.1.0, an open source SIP server and employ MySQL 4.1.12-3.RHEL4.1 as a back-end user database. We identified and fixed a flaw in the OpenSER transaction-stateful processing code, which resulted in a linear search on timer lists. We use this corrected version for all experiments, which resulted in a performance improvement over the original code of 3X and 6X with and without authentication, respectively.

Workload Generation. We use SIPp, another open-source tool, for load generation. We improved SIPp's performance so that we could generate high loads on our limited number of clients. Since we are not only interested in maximum throughput, but also in behavior under overload, we also modified SIPp to support open-loop workload generation to overload the server. These enhancements have been integrated into the mainline SIPp source tree.

Hardware. Our server is a 3.06 GHz Xeon running RedHat Enterprise Linux with a 2.6.17.8 kernel, and has a Gigabit Ethernet connected to a private network. We use 10 clients to generate load on this network; each with a 1.7 GHz Pentium 4 processor, a Gigabit Ethernet adapter, and SuSE SLES 9 with a 2.6.5-9 kernel.

Metrics. We measure throughput, success rate, CPU profiles, and latency (both averages and distributions). Each experiment lasts for 120 seconds after a 5 second warm-up time. Values reported are the average of five experiments and include 95% confidence intervals.

2. SAMPLE RESULTS

Due to space limitations, we only present a small sample of our complete results. Interested readers are referred to our technical report [1] for full details and analysis. In this Section, we present some of our results for proxying.

Throughputs. Figure 1 shows throughput versus offered load for 8 separate proxying configurations: stateful and stateless proxying, with and without authentication, and using UDP and TCP. We report peak throughputs (the maximum throughput with at least a 99% success rate) in Figure 2. The achieved throughputs vary considerably, depending on the configuration. Starting with the results for stateless proxying with UDP and no authentication as a "best case," we can see how the various features impact performance.

The most significant feature that affects performance is authentication, which can reduce performance anywhere from 60 percent (in the stateful TCP case) to 90 percent (in the stateless UDP case). The CPU profile (Figure 3) shows that when authentication is enabled, almost half the cycles are spent in the MySQL database and C library functions. Neither of these components are significant when authentication is not used. The actual MD5 hash calculation typically uses less than 1 percent of cycles.

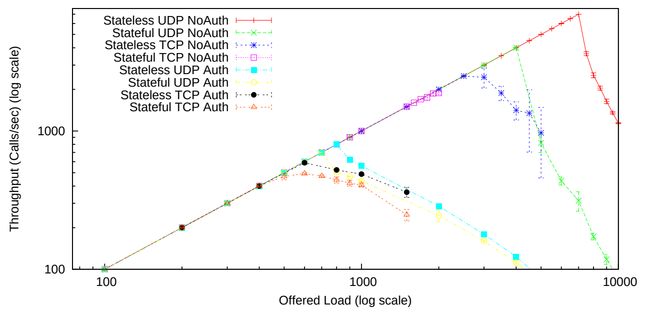

Figure 1: Throughputs vs. Offered Load: Proxying

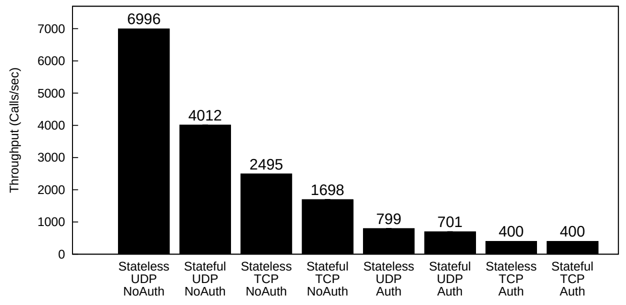

Figure 2: Peak Throughputs: Proxying

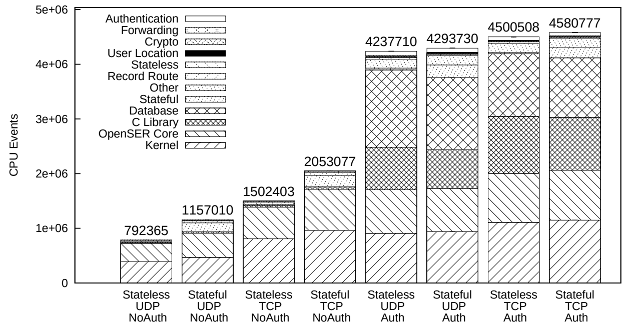

Figure 3: CPU Profile: Proxying 1000 CPS

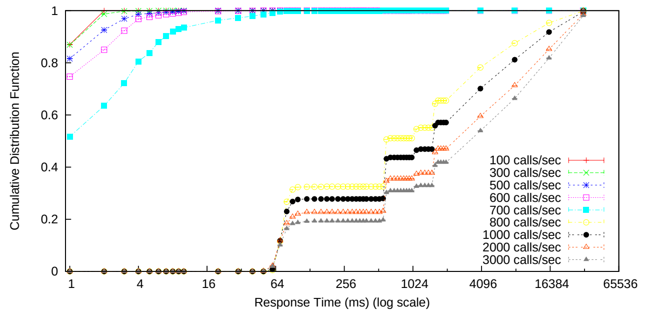

Figure 4: Response Time CDF: Stateful Proxying, UDP, with Authentication

The next most significant performance feature is whether TCP or UDP is used. Using TCP can reduce performance anywhere from 43 percent (stateful proxying with authentication) to 65 percent (stateless proxying without authentication). CPU profiles show that time spent in the OpenSER core goes up significantly, and the time spent in the kernel almost doubles. As TCP is a much more complex protocol that provides more functionality than UDP, it requires significantly larger code paths.

We also see that stateless vs. stateful processing can also have a significant performance impact. Enabling stateful processing can reduce performance by as much as 42 percent (UDP without authentication) to having effectively no impact on performance (TCP with authentication).

Finally, OpenSER does not preserve throughput under overload: throughput falls quickly after load exceeds capacity. Of course, maintaining peak throughput even under overload is difficult, and is the subject of active research.

Latencies. Figure 4 shows the response time CDF measured at several loads for stateful authenticated UDP proxying. As SIPp has a 1 millisecond timer resolution, any responses that occur within less than a millisecond are treated as zero. An obvious and expected result is that, as the loads increase, the response times increase as well (i.e., the curves shift to the right on the graph). There are, however two other interesting features of the graphs.

First, the curves cluster in two clearly different regions of the graph: one, towards the upper left of the graphs, and other, closer to the center and lower right. The characteristic that differentiates these two regions is whether the loads are below or above capacity, i.e., whether the system is overloaded. When overloaded, the response time distributions become significantly worse, and not linearly in proportion to the load. For example, the gap between the 700 curve and the 800 curve is significant. This response time behavior is particularly important for SIP servers, which need to provide service quickly and smoothly, as they are used for real-time media such as voice and video.

Second, observe that several significant jumps occur at certain response times (e.g., 64 ms, 500 ms, 1000 ms, 2000 ms, etc.). These are due to the retransmission timers used by SIP for reliability when using UDP. SIP's primary retransmission timer, called Timer A, uses an exponential backoff starting at 500 milliseconds and doubles each subsequent time that it fires. When the system is overloaded, we see the manifestations of these timers firing by the jumps in response time at those timer values.

3. FUTURE WORK

Based on our results, we believe many potential future research issues exist for SIP-servers, including: OpenSER and Linux optimizations based on profiling; maintaining throughput under overload; and improving database performance. There are also many important aspects of SIP-server performance not yet studied, including user-model based benchmarking; instant messaging and presence; and using SSL. Again, details are available in the tech report.

4. REFERENCES

[1] E. Nahum, J. Tracey, and C. P. Wright. Evaluating SIP Server Performance. Research report RC24183, IBM T. J. Watson Research Center, February 2007.