

# OPEN SIGCOMP™

Library Characteristics



## About this Document

This document describes measured and projected memory, compression, and speed metrics for the Open SigComp™ stack.

## Contact Information

For further information on the content of this document, please contact:

Adam Roach  
214-329-0491  
adam@estacado.net

## Document History

Version	Date	Author	Comments
0.1	04/21/2006	Adam Roach	First Draft

## Table of Contents

1	Introduction.....	1
2	Characterization .....	2
2.1	Speed.....	2
2.1.1	Typical Server.....	2
2.1.2	Typical Terminal.....	3
2.2	Compression .....	4
2.2.1	Deflate without SIP Dictionary.....	4
2.2.2	Deflate with SIP Dictionary.....	5
2.2.3	LZJH.....	5
2.3	Memory.....	6
2.3.1	Static Memory Usage.....	6
2.3.2	Heap Usage.....	8
2.3.3	Stack Usage.....	11
3	References.....	12

## **1 Introduction**

The Open SigComp™ stack is an extensible library that implements SigComp as described in RFC 3320 [1] and RFC 4077 [2]. Its design goals include support for both high-performance, multithreaded server applications as well as extremely constrained mobile device applications.

This document describes the various characteristics of the 0.9 (beta) version of the Open SigComp™ stack, in terms of execution speed, compression ratios, and memory usage.

## 2 Characterization

Except where noted, all characterization is performed using the GCC 4.0 compiler, targeting the Intel 80386 (i386) architecture. If a specific compression algorithm is not indicated, then the characterization was performed using a modified version of the Deflate algorithm [4], without taking advantage of the SIP SigComp dictionary [3].

### 2.1 Speed

#### 2.1.1 Typical Server

Server speed characterizations were performed under Linux (using a 2.6.11 version of the kernel) with a 2.4 GHz processor. Specific CPU information (in the form presented by the Linux `/proc/cpuinfo` pseudofile) is given in Figure 1. Only i386 opcodes were used in generation of the code. Execution was performed within a single thread. GCC level 2 optimization employed, and none of the space optimizations in the code were activated. The messages used for such characterization consisted of requests of approximately 800 to 900 bytes, and responses approximately 400 to 500 bytes.

Performance numbers on multi-core CPUs and/or multi-CPU systems should be approximately linear, since the bulk of the work performed by the library does not rely on objects shared between threads.

```
processor      : 0
vendor_id     : AuthenticAMD
cpu family    : 15
model         : 12
model name    : AMD Athlon(tm) 64 Processor 3300+
stepping      : 0
cpu MHz       : 2411.355
cache size    : 256 KB
fdiv_bug      : no
hlt_bug       : no
f00f_bug      : no
coma_bug      : no
fpu           : yes
fpu_exception : yes
cpuid level   : 1
wp            : yes
flags         : fpu vme de pse tsc msr pae mce cx8 apic mtrr pge
               mca cmov pat pse36 clflush mmx fxsr sse sse2 syscall
               nx mmxext lm 3dnowext 3dnow
bogomips      : 4784.12
```

**Figure 1: Profile System CPU Information**

Due to SigComp's heavy use of SHA-1 hashing (which is a computationally complex hashing algorithm), a substantial portion of the execution any SigComp implementation is spent computing SHA-1 hashes. The Open SigComp™ stack contains a highly portable SHA-1

---

# Open SigComp™

## Library Characteristics

---

implementation, written in C++, that has been organized to make efficient use of both CISC and RISC processor architectures. Nonetheless, aggressively hand optimized assembly that has been generated for a specific processor can usually outperform compiler-generated code. Such aggressive optimization has been performed by the OpenSSL project. The Open SigComp™ stack can take advantage of the OpenSSL SHA-1 optimizations, if the OpenSSL library (known as “libcrypto”) is available.

With libcrypto, the library is able to compress and decompress 100,000 messages in approximately 16.29 seconds of CPU time. This results in a processing speed 6,138 messages per second, with an introduced latency of about 162 microseconds per message.

Without libcrypto, the library is able to compress and decompress 100,000 messages in approximately 21.18 seconds of CPU time. This results in a processing speed 4,721 messages per second, with an introduced latency of about 211 microseconds per message.

These results are summarized in Table 1.

	<b>Seconds per 100k msgs</b>	<b>Msgs/sec per CPU core</b>	<b>Latency (<math>\mu</math>s)</b>
With libcrypto	16.29	6,138.74	162.90
Without libcrypto	21.18	4,721.44	211.80

**Table 1: Server Speed Summary**

### 2.1.2 Typical Terminal

Terminal speed has not yet been characterized.

## 2.2 Compression

The nature of the compression ratios that can be achieved by any given compression scheme vary widely depending on the commonality between messages and the amount of memory available for both decompression and state storage. All metrics in the following sections use an 8 kb Decompression Memory Size and an 8 kb State Memory Size. (These terms are formally defined in RFC 3320 [1]).

Because SigComp requires an exchange of bytecodes in the first message sent in each direction, the initial messages tend to have very poor – and sometimes even negative – compression ratios. After such bytecodes have been exchanged, compression ratios tend to improve over several messages, and converge to a steady state within 8 to 10 messages.

### 2.2.1 Deflate without SIP Dictionary

The following characteristics are based on the use of the Open SigComp™ implementation of Deflate [1]. Characterization was performed using a typical set of terminal start-up messages; specifically, a registration exchange followed by a single phone call.

The message types, sizes, and compression ratios are presented in Table 2. Ignoring the overhead represented by the bytecode exchange, this represents an overall compression ratio of 77%. The steady state compression ends up near 80% to 90%. These results are presented graphically in Figure 2.

	Message	Original	Compressed	Compression
1	REGISTER	850	1017	-20%
2	200	468	710	-52%
3	INVITE	1168	586	50%
4	407	483	208	57%
5	ACK	418	42	90%
6	INVITE	1360	105	92%
7	100	393	59	85%
8	200	632	189	70%
9	ACK	405	55	86%
10	BYE	405	39	90%
11	200	384	38	90%

Table 2: Message Compression (startup)

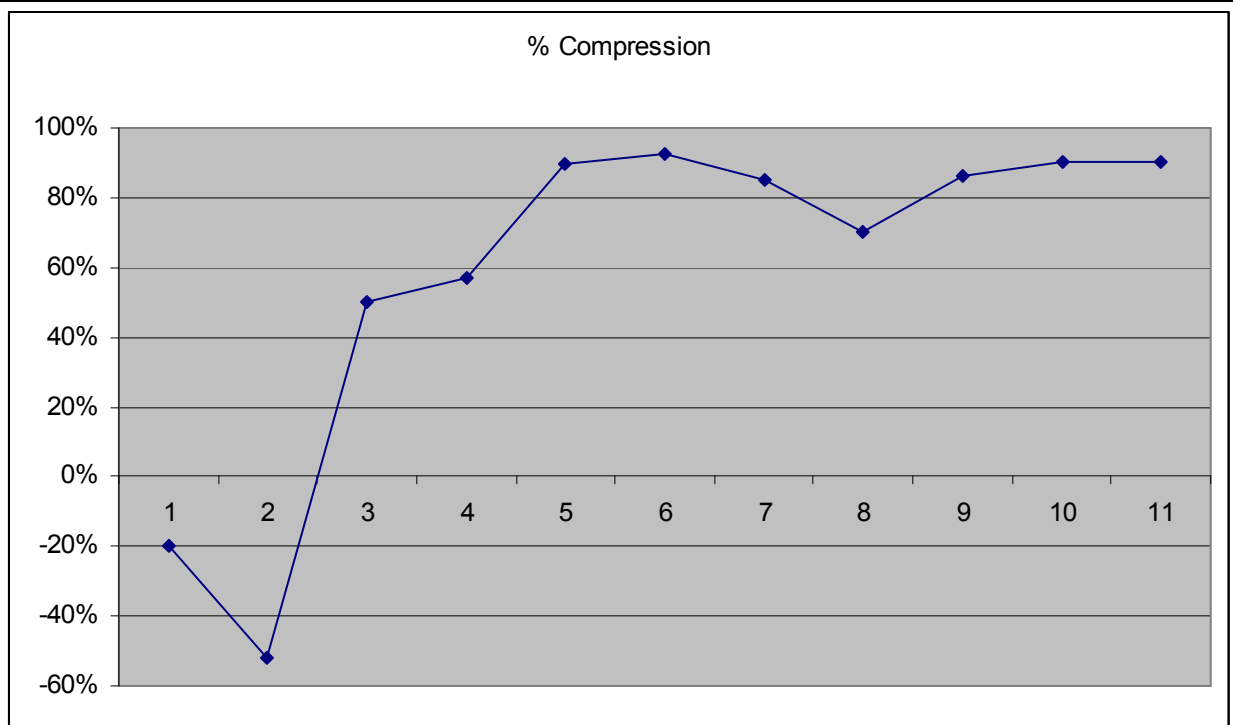


Figure 2: Message Compression (startup)

### 2.2.2 Deflate with SIP Dictionary

Message compression ratios using the Deflate algorithm with the use of the SIP Dictionary [3] have not yet been characterized.

### 2.2.3 LZJH

Message compression ratios using the LZJH algorithm have not yet been characterized.

## 2.3 Memory

Memory usage is based on generation of code for an i386-based system. The actual size for other processors will vary.

For the purposes of memory calculations, a typical server is projected to have 4 CPUs (and consequently 4 threads that could potentially be compressing messages simultaneously) and service 10,000 terminals. Because the heap memory requirements are dominated by state storage, these numbers scale almost linearly with the number of terminals each server is expected to service.

A typical handset is expected to run single-threaded applications, and communicate with up to 2 network servers (P-CSCF or Edge Proxy) at a time.

Memory usage for these typical server and typical handset scenarios is summarized in Table 3. These metrics are described in further detail in the following sections.

Finally, the Open SigComp™ library can be compiled in two different forms; one favors rapid execution, while the other favors low memory utilization. All memory metrics are presented for each of the two configurations.

	Size (in bytes)	
	Typical Handset	Typical Server
Stack	796	131,856
Heap	49,692	198,311,200
Static	44,083	62,194

Table 3: Memory Usage Summary

### 2.3.1 Static Memory Usage

The static memory used by the Open SigComp™ library, which translates to how much space it takes on disk (for servers) or in ROM (for embedded devices) is presented in Table 4 and Table 5.

Table 4 represents data usage, which does will not vary between platforms. It is possible to compile the Open SigComp™ library both with and without support for the SIP Dictionary [3]; this table summarizes the totals for both scenarios.

Table 5 represents the static memory used by executable code. These numbers depend on the processor being targeted. The numbers presented are based on the i386 architecture.

Table	Size (in bytes)	
	Speed Optimized	Size Optimized
osc::CrcComputer::s_table	512	512
osc::DeflateBytecodes::bytecodes	338	338
osc::DeflateCompressor::c_lengthTable	1,036	0
osc::DeflateCompressor::c_literalTable	1,024	0
osc::SipDictionary::s_stateValue	4,837	4,837
<b>Total with SIP Dictionary</b>	<b>7,747</b>	<b>5,687</b>
<b>Total without SIP Dictionary</b>	<b>2,910</b>	<b>850</b>

Table 4: Static Data Usage

Class	Size (in bytes)	
	Speed Optimized	Size Optimized
osc::BitBuffer	2,461	1,815
osc::Buffer	1,832	1,545
osc::Compartment	1,698	1,584
osc::CompartmentMap	2,219	1,723
osc::Compressor	84	84
osc::CompressorData	54	54
osc::CrcComputer	112	110
osc::DeflateBytecodes	44	44
osc::DeflateCompressor	4,486	3,847
osc::DeflateData	730	620
osc::DeflateDictionary	1,028	701
osc::MultiBuffer	13	301
osc::NackMap	1,316	1,164
osc::ReadWriteLockable	0	92
osc::Sha1Hasher	164	146
osc::SigcompMessage	2,664	2,330
osc::SipDictionary	157	140
osc::Stack	1,635	1,372
osc::State	1,084	1,000
osc::StateChanges	1,642	1,379
osc::StateHandler	2,663	2,156
osc::StateList	1,412	1,136
osc::TcpStream	1,214	881
osc::Udvm	24,455	13,018
Vtables and misc. functions	136	239
<b>Total</b>	<b>54,447</b>	<b>38,396</b>

Table 5: Static Text (Executable Code) Usage

## 2.3.2 Heap Usage

The size of objects in the heap may vary depending on whether the stack has been compiled for high performance or low memory usage. Table 6 details the size of various objects allocated from the heap; Table 7 details the size of various arrays allocated from the heap. Table 8 details the projected maximum memory used for the typical handset and typical server as described in section 2.3, with a State Memory Size (SMS) of 8 kb per compartment.

Object	Size (in bytes)		# of Instances
	Speed Optimized	Size Optimized	
osc::Compartment	244	244	1 per remote endpoint
osc::CompartmentMap::CompartmentBucket	12	12	O(# compartments / 4)
osc::DeflateCompressor	8	8	0 or 1 per thread
osc::DeflateData	264	264	1 per compartment
osc::NackMap	32	32	1 per state handler
osc::NackMap::NackNode	32	32	up to 4 x #endpoints
osc::SigcompMessage	104	104	0 or 1 per thread, plus any pending in TCP streams
osc::SipDictionary	64	64	0 or 1 per state handler
osc::Stack	132	132	1 per thread
osc::State	64	64	1 per state (local & remote)
osc::StateChanges	356	356	0 or 1 per thread
osc::StateHandler	160	160	Typically 1
osc::StateList	16	16	2 per compartment
osc::StateList::StateNode	16	16	1 per state in state list
osc::StateMap	48	48	1 per state handler
osc::StateMap::StateNode	8	8	1 per state in state handler
osc::TcpStream	20	20	1 per tcp connection
osc::Udvm	88	88	1 per thread

**Table 6: Object Heap Usage**

Array	Size (in bytes)
osc::Compartment::m_data	O(8x # compartments)
osc::CompartmentMap::m_compartments	O(4x # compartments)
osc::DeflateDictionary::m_indexNext	Message size x threads
osc::MultiBuffer::m_block	16, when compressing
osc::NackMap::m_bucketTops	O(16x # compartments)
osc::NackMap::m_list	O(16x # compartments)
osc::SigcompMessage (various)	Up to 2x the size of the message
osc::Statemap::m_states	O(# states)
osc::Udvm::m_memory	DMS x # threads
osc::MultiBuffer::m_buffer	history size when deflating
osc::Compartment::m_requestedFeedback	~ 32 x # compartments
osc::Compartment::m_returnedFeedback	~ 32 x # compartments
osc::StateChanges::m_requestedFeedback	~ 32 x # threads
osc::StateChanges::m_returnedFeedback	~ 32 x # threads
osc::State::m_buffer	< SMS x compartments x 2
osc::State::m_stateld	<~ 160 x # compartments
osc::TcpStream::partialFrame	up to 64k/stream; ~1k avg

**Table 7: Array Heap Usage**

Object	Typical Handset		Typical Large Server	
	Number	Total Size	Number	Total Size
osc::Compartment	2	488	10,000	2,440,000
osc::CompartmentMap::CompartmentBucket	1	12	2,500	30,000
osc::DeflateCompressor	1	8	4	32
osc::DeflateData	2	528	10,000	2,640,000
osc::NackMap	1	32	1	32
osc::NackMap::NackNode	8	256	40,000	1,280,000
osc::SigcompMessage	1	104	10,004	1,040,416
osc::SipDictionary	1	64	1	64
osc::Stack	1	132	4	528
osc::State	32	2,048	160,000	10,240,000
osc::StateChanges	1	356	4	1,424
osc::StateHandler	1	160	1	160
osc::StateList	4	64	20,000	320,000
osc::StateList::StateNode	32	512	160,000	2,560,000
osc::StateMap	1	48	1	48
osc::StateMap::StateNode	16	128	80,000	640,000
osc::TcpStream	0	0	10,000	200,000
osc::Udvm	1	88	4	352
osc::Compartment::m_data		16		80,000
osc::CompartmentMap::m_compartments		8		40,000
osc::DeflateDictionary::m_indexNext		1,024		4,096
osc::MultiBuffer::m_block		16		64
osc::NackMap::m_bucketTops		32		160,000
osc::NackMap::m_list		32		160,000
osc::SigcompMessage (various)		2,048		8,192
osc::Statemap::m_states		16		80,000
osc::Udvm::m_memory		8,192		32,768
osc::MultiBuffer::m_buffer		0		32,768
osc::Compartment::m_requestedFeedback		64		320,000
osc::Compartment::m_returnedFeedback		64		320,000
osc::StateChanges::m_requestedFeedback		32		128
osc::StateChanges::m_returnedFeedback		32		128
osc::State::m_buffer		32,768		163,840,000
osc::State::m_stateld		320		1,600,000
osc::TcpStream::partialFrame		0		10,240,000
<b>TOTAL</b>		<b>49,692</b>		<b>198,311,200</b>

Table 8: Projected Maximum Heap Usage for Typical Handset and Server Applications

### **2.3.3 Stack Usage**

Stack usage is measured in terms of the amount of *additional* stack memory the Open SigComp™ library requires on top of the stack in use by the application at the time that the Open SigComp™ library is invoked.

For speed optimized use, such as that found in servers, the additional stack usage is 131,856 bytes. This occurs during message compression.

For size optimized use, such as that found in terminals, the additional stack usage is 796 bytes. This occurs during message compression.

### **3 References**

- [1] Price, R., Bormann, C., Christoffersson, J., Hannu, H., Liu, Z., and J. Rosenberg, “Signaling Compression (SigComp)”, RFC 3320, January 2003.
- [2] Roach, A. B., “A Negative Acknowledgement Mechanism for Signaling Compression”, RFC 4077, May 2005.
- [3] Garcia-Martin, M., et al, “The Session Initiation Protocol (SIP) and Session Description Protocol (SDP) Static Dictionary for Signaling Compression (SigComp)”, RFC 3485, February 2003.
- [4] Deutch, P., “DEFLATE Compressed Data Format Specification version 1.3”, RFC 1951, May 1996.