



SigComp Stack

Technical Reference Manual

© Copyright 2005 Bitlynx Technologies Inc.

Introduction

SigComp is a solution for compressing messages generated by application protocols such as the Session Initiation Protocol (SIP). The SIP protocol is text-based, not optimized in terms of size, and engineered for bandwidth-rich links. With the planned usage of these protocols in wireless handsets, as part of 2.5G and 3G cellular networks, large message size becomes a problem. The IETF solution is SigComp (RFC3320), which offers robust, loss-less compression of application messages. Due to the heavy signaling involved and SIP, being a text based protocol, 3GPP IMS Release 5 standards mandates SigComp.

The foundation for SigComp is the Universal Decompressor Virtual Machine (UDVM), which enables the compressor to dynamically select the compression algorithm of its choice without requiring the decompressor to support the algorithm a priori.

SigComp is offered to applications as a layer between the application and an underlying transport. The service provided is that of the underlying transport plus compression. SigComp supports a wide range of transports including TCP and UDP.

In its simplest form, a SigComp message consists of a short identification header, followed by UDVM bytecode, and the message payload. The bytecode is algorithm-specific instructions used by the UDVM to decompress the message payload. Even with the size of the UDVM bytecode added to the compressed payload, the size of the SigComp message is generally much smaller than the original data packet.

Improved compression is attainable by providing side-channel feedback information from the remote decompressor to the compressor. This feedback information can be used by the compressor to build a stateful protocol and synchronize state information at the compressor and decompressor. An immediate advantage of a stateful protocol is that the UDMV bytecode need only be transferred once between the compressor and decompressor, resulting in improved compression performance.

This document outlines the architecture and API of the SigComp stack, state handler, compression engine, decompression engine and UDVM.

Features

The Bitlynx Technologies Inc. SigComp stack is an implementation of the SigComp header compression framework. The implementation is fully compliant with RFC3320 and conforms to the requirements of draft-ietf-rohc-sigcomp-impl-guide-05 and draft-ietf-rohc-sigcomp-torture-test-01.

The SigComp stack supports the following features:

- Modular design for UDVM, compressors, decompressor and state handler for easy integration
- Automatic detection of compressed packets
- Supports state access and compartments
- Supports datagram and streaming transports
- Includes SIP and SDP static dictionaries defined in RFC3485
- Supports Deflate, LZSS and LZJH (V.44) compressors with dynamic compression according to RFC3321
- simple SIP stack integration
- C, C++ and Java bindings

The SigComp stack is available for Microsoft Windows, Mac OS X, NetBSD, Linux and Symbian. Embedded platforms are supported on request.

Architecture

The SigComp API closely matches the architecture of SigComp as specified in RFC3320. The SigComp stack consists of the following interfaces:

- UDVM interface
- decompressor interface
- state handler interface
- sigcomp interface
- compressor interfaces

The UDVM interfaces provides access to the virtual machine. It is the lowest level of processing. The UDVM is controlled using callbacks through the UDVM interface.

The decompressor interface performs the decompressor dispatching role and provides a higher-level interface to the UDVM. It's responsible for dispatching SigComp messages, handling the callbacks from the UDVM, and interfacing to the state handler.

The compressor interface is a uniform interface used by all compressors. The interface is suitable for high-level application integration and is easily extensible in many different languages.

The sigcomp interface is the high-level interface and provides simple-to-use access to the decompressor and state-handler interfaces. The sigcomp interface manages the state handler and the compressors. The sigcomp interface is the interface which is generally used by most users.

From an application perspective the SigComp API appears as a stateful transformation filter, which converts the application data packet into a SigComp message.

Sigcomp Stack File Index

Sigcomp Stack File List

Here is a list of all files with brief descriptions:

compressor.h	13
decompressor.h	14
deflate.h	16
lzjh.h	18
lzss.h	20
sigcomp.h	22
statehandler.h	23
udvm.h	28

Sigcomp Stack Module Documentation

Sigcomp

Detailed Description

The high-level interface is based on a SigComp session. A session is a full-duplex communication channel between two endpoints. A session is initially linked with a Compressor. SigComp messages are decompressed and assigned to a particular session depending on the application-specific contents of the decompressed packet. Invalid packets are not assigned to any session.

Functions

- `int sigcomp_init (void)`
 - `int sigcomp_new_session (compressor_handle_t chandle, unsigned int memory_size, sigcomp_session_handle_t *handle)`
 - `int sigcomp_end_session (sigcomp_session_handle_t handle)`
 - `int sigcomp_is_message (unsigned char *msg, unsigned int msglen)`
 - `int sigcomp_compress (sigcomp_session_handle_t handle, unsigned char *pkt, unsigned int pktlen, unsigned char *msg, unsigned int msglen)`
 - `int sigcomp_decompress (unsigned char *msg, unsigned int msglen, unsigned char *pkt, unsigned int pktlen, udvm_profile_t *profile, decompressor_handle_t *dhandle)`
 - `int sigcomp_accept (decompressor_handle_t dhandle, sigcomp_session_handle_t handle)`
 - `int sigcomp_reject (decompressor_handle_t dhandle)`
-

Function Documentation

`int sigcomp_init (void)`

Initialize the SigComp stack. **sigcomp_init()** must be called once before any other functions in the SigComp stack are used. It initializes the stack and loads the SIP+SDP dictionary (RFC3485) into the state handler.

Returns:

Returns 0 on success, or a non-zero value on error.

Examples:

`basic.c`.

`int sigcomp_new_session (compressor_handle_t chandle, unsigned int memory_size, sigcomp_session_handle_t * handle)`

Create a new SigComp session. A new session must be associated with a Compressor.

Parameters:

chandle Opque Compressor handle.

memory_size StateHandler memory size (in bytes)

handle Pointer to opaque Session handle.

Returns:

Returns 0 on success, or a non-zero value on error.

Examples:

basic.c.

int sigcomp_end_session (sigcomp_session_handle_t handle)

Release the context and resources associated with a session.

Parameters:

handle Opaque Session handle.

Returns:

Returns 0 on success, or a non-zero value on error.

int sigcomp_is_message (unsigned char * msg, unsigned int msglen)

Verify that the message is a valid SigComp message. This function is useful in applications that support both compressed and uncompressed packets.

Parameters:

msg Pointer to buffer containing packet data.

msglen Length of packet data (in bytes).

Returns:

Returns 0 on success, or a non-zero value on error.

int sigcomp_compress (sigcomp_session_handle_t handle, unsigned char * pkt, unsigned int pktlen, unsigned char * msg, unsigned int msglen)

Compress application packet into SigComp message.

Parameters:

handle Opaque Session handle.

pkt Pointer to uncompressed packet data.

pktlen Length of uncompressed packet data (in bytes).

msg Pointer to buffer to store compressed SigComp message.

msglen Length of buffer to store compressed SigComp message.

Returns:

If successful, returns the number of bytes in the compressed SigComp message, otherwise returns a negative error value.

Examples:

basic.c.

int sigcomp_decompress (unsigned char * msg, unsigned int msglen, unsigned char * pkt, unsigned int pktlen, udvm_profile_t * profile, decompressor_handle_t * dhandle)

Decompress SigComp message into application data packet. If the decompression application packet is valid, the application should call **sigcomp_accept()** on the returned Decompressor handle.

Parameters:

msg Pointer to SigComp message.

msglen Length of SigComp message (in bytes).

pkt Pointer to uncompressed packet data.

pktlen Length of uncompressed packet data (in bytes).
profile Pointer to UDVM profile. If NULL, the default profile is used.
dhandle Pointer to Opaque Decompressor handle.

Returns:

If successful, returns the number of bytes in the decompressed application packet, otherwise returns a negative error value.

Examples:

basic.c.

int sigcomp_accept (decompressor_handle_t *dhandle*, sigcomp_session_handle_t *handle*)

Accept the decompressed packet and associate with the specified Session handle.

Parameters:

dhandle Opaque Decompressor handle.
handle Opaque Session handle.

Returns:

Returns 0 on success, or a non-zero value on error.

Examples:

basic.c.

int sigcomp_reject (decompressor_handle_t *dhandle*)

Reject the decompressed packet.

Parameters:

dhandle Opaque Decompressor handle.

Sigcomp Stack Data Structure Documentation

compressor_handle_data Struct Reference

Detailed Description

A `compressor_handle_t` type is a handle to a compressor. The type should be considered opaque for users of sigcomp stack.

A compressor implementation should create a `compressor_handle_t` type during initialization and populate the fields with appropriate callbacks to its internal implementation. The callbacks are called by the sigcomp module during compression.

Examples:

`basic.c`.

Data Fields

- `int(* process)(struct compressor_handle_data *handle, unsigned char *pkt, unsigned int pktlen, unsigned char *msg, unsigned int msglen, unsigned char *feedback, unsigned int feedback_length, unsigned int *partial_state_len)`

Field Documentation

`int(* process)(struct compressor_handle_data *handle, unsigned char *pkt, unsigned int pktlen, unsigned char *msg, unsigned int msglen, unsigned char *feedback, unsigned int feedback_length, unsigned int *partial_state_len)`

Callback to process a data packet to produce a SigComp message.

Parameters:

handle The compressor handle.

pkt Pointer to the input data packet.

pktlen The length the input data packet (in bytes).

msg Pointer to the buffer to put the output message.

msglen The length of the buffer to put the output message.

feedback Pointer to feedback data from decompressor.

feedback_length The length of feedback data (in bytes).

partial_state_len The length of the partial state identifier inserted into the message.

Returns:

Returns 0 on success, or a non-zero value on error.

udvm_decompressor_callback_data Struct Reference

Detailed Description

The `udvm_decompressor_callback_t` type is a structure with data-processing callbacks which is passed to the UDVM. These callbacks are responsible for reading the SigComp message payload and writing decompressed UDVM output.

Data Fields

- `int(* input_byte)(void *cookie, uint8_t *octet)`
 - `int(* input_seek)(void *cookie, int offset, unsigned int whence)`
 - `int(* output_byte)(void *cookie, uint8_t *octet)`
-

Field Documentation

`int(* input_byte)(void *cookie, uint8_t *octet)`

Callback to input one byte of the SigComp message payload.

Parameters:

cookie Opaque handle passed to the UDVM.
octet Pointer to byte to load the input byte.

Returns:

Returns 0 on success, or a non-zero value on error.

`int(* input_seek)(void *cookie, int offset, unsigned int whence)`

Callback to move the read pointer in the SigComp message payload. The new position, measured in bytes, is obtained by adding offset bytes to the position specified by whence. If whence is set to `SEEK_SET`, `SEEK_CUR`, or `SEEK_END`, the offset is relative to the start of the file, the current position indicator, or end-of-file, respectively.

Parameters:

cookie Opaque handle passed to the UDVM.
offset Offset to move read pointer.
whence Reference for Offset.

Returns:

Returns 0 on success, or a non-zero value on error.

`int(* output_byte)(void *cookie, uint8_t *octet)`

Output one byte of decompressed output.

Parameters:

cookie Opaque handle passed to the UDVM.
octet Pointer to the byte to output.

Returns:

Returns 0 on success, or a non-zero value on error.

udvm_profile_data Struct Reference

Detailed Description

The operation of the UDVM can be controlled with a number of parameters. These parameters are passed to the UDVM as a profile in a `udvm_profile_t` type. The following parameters can be controlled:

- the version of the UDVM specification to support. Currently on version 1 has been specified and is supported.
- the size of UDVM memory
- the number of cycles available per message payload bit used in limiting the computation load (deadlocks) in UDVM execution
- the message header size, also used to limit the computation load in UDVM execution

Public Types

- typedef unsigned int **version**
 - typedef unsigned int **memory_size**
 - typedef unsigned int **cycles_per_bit**
 - typedef unsigned int **header_size**
-

Member Typedef Documentation

typedef unsigned int version

Version of UDVM specification to support.

typedef unsigned int memory_size

Size of UDVM memory.

typedef unsigned int cycles_per_bit

Cycles available per message payload bit.

typedef unsigned int header_size

Size of sigcomp header.

udvm_statehandler_callback_data Struct Reference

Detailed Description

The `udvm_statehandler_t` type is a structure with state-handling callbacks which is passed to the UDVM. These callbacks are responsible for performing all state handling and feedback operations associated with stateful SigComp signalling.

Data Fields

- `int(* state_create)(void *cookie, void *compartment_cookie, uint8_t *id, unsigned int idlen, uint8_t *sbuf, unsigned int len, unsigned int statepri)`
 - `int(* state_free)(void *cookie, void *compartment_cookie, uint8_t *id, unsigned int idlen)`
 - `int(* state_access)(void *cookie, uint8_t *id, unsigned int idlen, uint8_t **sbuf, unsigned int *len)`
 - `int(* requested_feedback)(void *cookie, void *compartment_cookie, uint8_t *feedback, unsigned int length)`
 - `int(* sbit)(void *cookie, void *compartment_cookie, unsigned int sbit)`
 - `int(* ibit)(void *cookie, void *compartment_cookie, unsigned int ibit)`
-

Field Documentation

`int(* state_create)(void *cookie, void *compartment_cookie, uint8_t *id, unsigned int idlen, uint8_t *sbuf, unsigned int len, unsigned int statepri)`

Callback to create a new state in the state handler.

Parameters:

cookie Opaque handle passed to the UDVM.
compartment_cookie Opaque handle passed to the UDVM.
id Pointer to the partial state identifier for the new state.
idlen Length of the partial state identifier (in bytes).
sbuf Pointer to a buffer containing the state to add to the state handler.
len Length of the state (in bytes).
statepri Priority to add new state to the state handler.

Returns:

Returns 0 on success, or a non-zero value on error.

`int(* state_free)(void *cookie, void *compartment_cookie, uint8_t *id, unsigned int idlen)`

Callback to free the resources in the state handler associated with the partial state identifier.

Parameters:

cookie Opaque handle passed to the UDVM.
compartment_cookie Opaque handle passed to the UDVM.
id Pointer to the partial state identifier to free.
idlen Length of the partial state identifier (in bytes)

Returns:

Returns 0 on success, or a non-zero value on error.

int(* state_access)(void *cookie, uint8_t *id, unsigned int idlen, uint8_t **sbuf, unsigned int *len)

Callback to access, or retrieve, the state associated with the partial state identifier.

Parameters:

cookie Opaque handle passed to the UDVM.
id Pointer to the partial state identifier to access.
idlen Length of the partial state identifier (in bytes).
sbuf Pointer to load the pointer to the state.
len Length of the state (in bytes).

Returns:

Returns 0 on success, or a non-zero value on error.

int(* requested_feedback)(void *cookie, void *compartment_cookie, uint8_t *feedback, unsigned int length)

Callback to return feedback data requested by the compressor to the state handler.

Parameters:

cookie Opaque handle passed to the UDVM.
compartment_cookie Opaque handle passed to the UDVM.
feedback Pointer to buffer with the feedback data.
length Length of the feedback data (in bytes).

Returns:

Returns 0 on success, or a non-zero value on error.

int(* sbit)(void *cookie, void *compartment_cookie, unsigned int sbit)

Callback to handle setting of the UDVM S-bit. The compressor sets the S-bit if it does not wish (or no longer wishes) to save state information at the receiving endpoint and also does not wish to access state information that it has previously saved.

Parameters:

cookie Opaque handle passed to the UDVM.
compartment_cookie Opaque handle passed to the UDVM.
sbit Value of the UDVM S-bit.

Returns:

Returns 0 on success, or a non-zero value on error.

int(* ibit)(void *cookie, void *compartment_cookie, unsigned int ibit)

Callback to handle setting of the UDVM I-bit. The compressor sets the I-bit if it does not wish (or no longer wishes) to access any of the locally available state items offered by the receiving endpoint.

Parameters:

cookie Opaque handle passed to the UDVM.
compartment_cookie Opaque handle passed to the UDVM.
ibit Value of the UDVM I-bit.

Returns:

Returns 0 on success, or a non-zero value on error.

Sigcomp Stack File Documentation

compressor.h File Reference

Detailed Description

compressor.h specifies the basic compressor interface to the sigcomp stack.

Generally, most users will not need to use the compressor interface, and can work directly with the stack through the **sigcomp.h** interface.

compressor.h should be included into all custom compressors.

Data Structures

- struct **compressor_handle_data**
Opaque compressor handle.

Typedefs

- typedef **compressor_handle_data** * **compressor_handle_t**
-

Typedef Documentation

typedef struct compressor_handle_data* compressor_handle_t

decompressor.h File Reference

Detailed Description

decompressor.h specifies the decompressor and dispatcher interface of the sigcomp stack.

Generally, most users will not need to use the decompressor interface, and can work directly with the stack through the **sigcomp.h** interface.

The decompressor dispatcher receives messages from the transport layer and passes the decompressed version of each message to the application. To ensure that SigComp can run over an unsecured transport layer, the decompressor dispatcher invokes a new instance of the UDVM for each new SigComp message.

Resources for the UDVM must be released as soon as the message has been decompressed.

Typedefs

- typedef decompressor_handle_data * **decompressor_handle_t**

Functions

- int **decompressor_init** (**udvm_profile_t** *profile, **statehandler_callback_t** *statehandler_callback, **statehandler_handle_t** statehandler_handle, **decompressor_handle_t** *handle)
- int **decompressor_destroy** (**decompressor_handle_t** handle)
- int **decompressor_process** (**decompressor_handle_t** handle, **uint8_t** *msg, unsigned int msglen, **uint8_t** *obuf, unsigned int *obuflen)
- int **decompressor_state_update** (**decompressor_handle_t** handle, **compartment_handle_t** chandle)

Typedef Documentation

typedef struct decompressor_handle_data* **decompressor_handle_t**

Examples:

basic.c.

Function Documentation**int decompressor_init** (**udvm_profile_t** * *profile*, **statehandler_callback_t** * *statehandler_callback*, **statehandler_handle_t** *statehandler_handle*, **decompressor_handle_t** * *handle*)

Initialize the Decompressor.

Parameters:

profile Pointer to UDVM profile. If NULL, the default profile is used.

statehandler_callback Pointer to callback to state handler.

statehandler_handle Opaque handle to the statehandler which is passed as first argument in state handler callbacks.

handle Pointer to opaque decompressor handle

Returns:

Returns 0 on success, or a non-zero value on error.

int decompressor_destroy (decompressor_handle_t handle)

Destroy the Decompressor context and resources. `decompressor_destory()` must be called on the Decompressor handle to free resources.

Parameters:

handle Opaque Decompressor handle.

Returns:

Returns 0 on success, or a non-zero value on error.

int decompressor_process (decompressor_handle_t handle, uint8_t * msg, unsigned int msglen, uint8_t * obuf, unsigned int * obuflen)

Process a SigComp message.

Parameters:

handle Opaque Decompressor handle.

msg SigComp message to decompress.

msglen Length of SigComp message.

obuf Pointer to decompression buffer to place decompressed data.

obuflen Pointer to value specifying the length (in bytes) of the decompression buffer. On return, *obuflen* contains the number of bytes in the decompressed message.

Returns:

Returns 0 on success, or a non-zero value on error.

int decompressor_state_update (decompressor_handle_t handle, compartment_handle_t chandle)

Update the state associated with a decompressed message. Upon decompression, the caller should invoke **decompressor_state_update()** to indicate that the SigComp message decompressed correctly and contains valid data, and the state operations may be committed to state handler.

Parameters:

handle Opaque Decompressor handle.

chandle Opaque compartment handle passed to the state handler.

Returns:

Returns 0 on success, or a non-zero value on error.

deflate.h File Reference

Detailed Description

This file specifies the API to the Deflate compressor.

The Deflate algorithm is defined in RFC1951 and defines a lossless compressed data format that compresses data using a combination of the LZ77 algorithm and Huffman coding. The data is produced using only an a priori bounded amount of intermediate storage. The Deflate algorithm is not covered by patents.

The compressed data format consists of a single block. The block size is not fixed except that non-compressible blocks are limited to 65,535 bytes.

The compressed data consists of a series of elements of two types: literal bytes (of strings that have not been detected as duplicated within the previous 32K input bytes), and pointers to duplicated strings, where a pointer is represented as a pair <length, backward distance>. The representation used in the "deflate" format limits distances to 32K bytes and lengths to 258 bytes, but does not limit the size of a block, except for uncompressible blocks, which are limited as noted above.

Each type of value (literals, distances, and lengths) in the compressed data is represented using a Huffman code, using one code tree for literals and lengths and a separate code tree for distances.

The Deflate compressor supports the use of the SIP+SDP dictionary defined in RFC3485. Use of the dictionary is recommended to significantly reduce the size of the compressed message.

The Deflate compressor supports a stateful protocol for efficient compression without the need to transfer bytecode to the decompressor more than once.

Functions

- `int deflate_process (compressor_handle_t handle, unsigned char *pkt, unsigned int pktlen, unsigned char *msg, unsigned int *msglen, unsigned char *feedback, unsigned int feedback_length, unsigned int *partial_state_len)`
- `int deflate_init (compressor_handle_t *handle, int use_dict)`
- `int deflate_free (compressor_handle_t handle)`

Function Documentation

`int deflate_process (compressor_handle_t handle, unsigned char * pkt, unsigned int pktlen, unsigned char * msg, unsigned int * msglen, unsigned char * feedback, unsigned int feedback_length, unsigned int * partial_state_len)`

Process a data packet to produce a SigComp message.

Parameters:

handle Deflate compressor handle

pkt Pointer to the input data packet

pktlen Length of the input data packet (in bytes)

msg Pointer to the buffer to put the compressed message

msglen Pointer to a value specifying the length of the output buffer. On return, msglen contains the number of bytes in the compressed message.

feedback Pointer to feedback data from decompressor.

feedback_length Length of feedback data (in bytes).

partial_state_len The length of the partial state identifier inserted into the message.

Returns:

Returns 0 on success, or a non-zero value on error.

int deflate_init (compressor_handle_t * handle, int use_dict)

Initialize the Deflate Compressor.

Parameters:

handle Pointer to Deflate compressor handle.

use_dict Whether to use the SIP+SDP dictionary (RFC3485).

Returns:

Returns 0 on success, or a non-zero value on error.

int deflate_free (compressor_handle_t handle)

Free the Deflate resources. **deflate_free()** must be called on the compression handle to free Deflate resources.

Parameters:

handle Opaque Deflate compressor handle.

Returns:

Returns 0 on success, or a non-zero value on error.

Izjh.h File Reference

Detailed Description

This file specifies the API to the LZJH compressor.

The LZJH algorithm is a lossless compressed data format that compresses data using the LZ78 algorithm. The data is produced using only an a priori bounded amount of intermediate storage.

The compressed data consists of a series of elements of two primary types: literal bytes and control words. Control words are special codes to control the operating parameters of the decompressor, or to specify codes into a LZ78 codebook. Each codebook entry represents a pair <length, backward distance> of duplicated strings.

The LZJH compressor supports the use of the SIP+SDP dictionary defined in RFC3485. Use of the dictionary is recommended to significantly reduce the size of the compressed message.

The LZJH compressor supports a stateful protocol for efficient compression without the need to transfer bytecode to the decompressor more than once.

Functions

- int **lzjh_process** (**compressor_handle_t** handle, unsigned char *pkt, unsigned int pktlen, unsigned char *msg, unsigned int *msglen, unsigned char *feedback, unsigned int feedback_length, unsigned int *partial_state_len)
 - int **lzjh_init** (**compressor_handle_t** *handle, int use_dict)
 - int **lzjh_free** (**compressor_handle_t** handle)
-

Function Documentation

int lzjh_process (compressor_handle_t handle, unsigned char * pkt, unsigned int pktlen, unsigned char * msg, unsigned int * msglen, unsigned char * feedback, unsigned int feedback_length, unsigned int * partial_state_len)

Process a data packet to produce a SigComp message.

Parameters:

handle LZJH compressor handle

pkt Pointer to the input data packet

pktlen Length of the input data packet (in bytes)

msg Pointer to the buffer to put the compressed message

msglen Pointer to a value specifying the length of the output buffer. On return, msglen contains the number of bytes in the compressed message.

feedback Pointer to feedback data from decompressor.

feedback_length Length of feedback data (in bytes).

partial_state_len The length of the partial state identifier inserted into the message.

Returns:

Returns 0 on success, or a non-zero value on error.

int lzjh_init (compressor_handle_t * handle, int use_dict)

Initialize the LZJH Compressor.

Parameters:

handle Pointer to LZJH compressor handle.

use_dict Whether to use the SIP+SDP dictionary (RFC3485).

Returns:

Returns 0 on success, or a non-zero value on error.

int lzjh_free (compressor_handle_t *handle*)

Free the LZJH resources. **lzjh_free()** must be called on the compression handle to free LZJH resources.

Parameters:

handle Opaque LZJH compressor handle.

Returns:

Returns 0 on success, or a non-zero value on error.

lzss.h File Reference

Detailed Description

This file specifies the API to the LZSS compressor.

The LZSS algorithm is a lossless compressed data format that compresses data using the LZ77 algorithm. The data is produced using only an a priori bounded amount of intermediate storage. The LZSS algorithm is not covered by patents.

The compressed data consists of a series of elements of two types: literal bytes (of strings that have not been detected as duplicated within the previous 4K input bytes), and pointers to duplicated strings, where a pointer is represented as a pair <length, backward distance>.

The LZSS algorithm has the advantage over other compression algorithm in that it is fast the decompress and the bytecode size is small. This characteristic makes it ideal for applications protocols which don't establish a session or dialog.

The LZSS compressor supports the use of the SIP+SDP dictionary defined in RFC3485. Use of the dictionary is recommended to significantly reduce the size of the compressed message.

The LZSS compressor supports a stateful protocol for efficient compression without the need to transfer bytecode to the decompressor more than once.

Functions

- `int lzss_process (compressor_handle_t handle, unsigned char *pkt, unsigned int pktlen, unsigned char *msg, unsigned int *msglen, unsigned char *feedback, unsigned int feedback_length, unsigned int *partial_state_len)`
- `int lzss_init (compressor_handle_t *handle, int use_dict)`
- `int lzss_free (compressor_handle_t handle)`

Function Documentation

`int lzss_process (compressor_handle_t handle, unsigned char * pkt, unsigned int pktlen, unsigned char * msg, unsigned int * msglen, unsigned char * feedback, unsigned int feedback_length, unsigned int * partial_state_len)`

Process a data packet to produce a SigComp message.

Parameters:

- handle* LZSS compressor handle
- pkt* Pointer to the input data packet
- pktlen* Length of the input data packet (in bytes)
- msg* Pointer to the buffer to put the compressed message
- msglen* Pointer to a value specifying the length of the output buffer. On return, msglen contains the number of bytes in the compressed message.
- feedback* Pointer to feedback data from decompressor.
- feedback_length* Length of feedback data (in bytes).
- partial_state_len* The length of the partial state identifier inserted into the message.

Returns:

Returns 0 on success, or a non-zero value on error.

int lzss_init (compressor_handle_t * handle, int use_dict)

Initialize the LZSS Compressor.

Parameters:

handle Pointer to LZSS compressor handle.

use_dict Whether to use the SIP+SDP dictionary (RFC3485).

Returns:

Returns 0 on success, or a non-zero value on error.

Examples:

basic.c.

int lzss_free (compressor_handle_t handle)

Free the LZSS resources. **lzss_free()** must be called on the compression handle to free LZSS resources.

Parameters:

handle Opaque LZSS compressor handle.

Returns:

Returns 0 on success, or a non-zero value on error.

Examples:

basic.c.

sigcomp.h File Reference

Detailed Description

sigcomp.h specifies the primary interface to the SigComp stack. Generally, most users will not be required to use any other interface.

The interface specified in this document is easy to use, yet provides all the capabilities of SigComp, including fully stateful compression. All state handling is handled internally.

Functions

- int **sigcomp_init** (void)
- int **sigcomp_new_session** (**compressor_handle_t** chandle, unsigned int memory_size, **sigcomp_session_handle_t** *handle)
- int **sigcomp_end_session** (**sigcomp_session_handle_t** handle)
- int **sigcomp_is_message** (unsigned char *msg, unsigned int msglen)
- int **sigcomp_compress** (**sigcomp_session_handle_t** handle, unsigned char *pkt, unsigned int pktlen, unsigned char *msg, unsigned int msglen)
- int **sigcomp_decompress** (unsigned char *msg, unsigned int msglen, unsigned char *pkt, unsigned int pktlen, **udvm_profile_t** *profile, **decompressor_handle_t** *dhandle)
- int **sigcomp_accept** (**decompressor_handle_t** dhandle, **sigcomp_session_handle_t** handle)
- int **sigcomp_reject** (**decompressor_handle_t** dhandle)

statehandler.h File Reference

Detailed Description

Generally, most users will not need to use the state handler interface, and can work directly with the stack through the **sigcomp.h** interface.

The function of the state handler is to retain information between received SigComp messages; it is the only SigComp entity that is capable of this function. The overall compression ratio is often significantly higher if messages can be compressed relative to the information contained in previous messages. For this reason, it is possible to create state items for access when a later message is being decompressed.

SigComp protects state access by creating a state identifier that is a hash over the item of state to be retrieved. This `state_identifier` must be supplied to retrieve an item of state from the state handler.

The state handler manages state memory on a per-compartment basis. A compartment is an application-specific grouping of messages that relate to a peer endpoint. Depending on the signaling protocol, this grouping may relate to application concepts such as "session", "dialog", "connection", or "association". Each compartment can store state up to a certain state memory size. Different values for the state memory size may be assigned to different compartments.

As well as storing the state items themselves, the state handler maintains a list of the state items created by a particular compartment and ensures that no compartment exceeds its allocated state memory size.

SigComp performs feedback on a request/response basis, so a compressor makes a feedback request and receives some feedback data in return. The SigComp feedback mechanism allows feedback data to be received by a UDVM and forwarded via the state handler to the correct compressor. Since this feedback data is retained between SigComp messages, it is considered to be part of the overall state and can only be forwarded if accompanied by a valid compartment identifier.

Consider a stateful compressor at "Endpoint A" sending compressed messages to "Endpoint B" and wishes to monitor the state at the decompressor. This is how it works:

- "Endpoint A" sends the appropriate bytecode instructions to "Endpoint B".
- "Endpoint B" UDVM executes the bytecode to create the feedback data and executes the END-MESSAGE instruction to save the "requested feedback" into its state handler.
- "Endpoint B" compressor retrieves any "requested feedback" in its state handler and bundles the feedback data in a message to "Endpoint A".
- "Endpoint A" decompressor receives the bundled feedback data, now termed "returned feedback", and stores it in its state handler.
- "Endpoint A" compressor retrieves any "returned feedback" which it request from "Endpoint B".

The confusion in this process is that a Compressor must check its state handler for "requested feedback" to bundle feedback data `_to_` the remote endpoint, and check its state handler for "returned feedback" `_from_` the remote endpoint.

Typedefs

- `typedef compartment_handle_data * compartment_handle_t`

- typedef statehandler_handle_data * **statehandler_handle_t**

Functions

- int **statehandler_init** (statehandler_handle_t *handle)
- int **statehandler_destroy** (statehandler_handle_t handle)
- int **statehandler_create_compartment** (statehandler_handle_t handle, unsigned int memory_size, compartment_handle_t *chandle)
- int **statehandler_close_compartment** (statehandler_handle_t handle, compartment_handle_t chandle)
- int **statehandler_create_state** (statehandler_handle_t handle, compartment_handle_t chandle, uint8_t *id, unsigned int idlen, uint8_t *sbuf, unsigned int len, unsigned int statepri)
- int **statehandler_free_state** (statehandler_handle_t handle, compartment_handle_t chandle, uint8_t *id, unsigned int idlen)
- int **statehandler_access_state** (statehandler_handle_t handle, uint8_t *id, unsigned int idlen, uint8_t **sbuf, unsigned int *len)
- int **statehandler_set_requested_feedback** (statehandler_handle_t handle, compartment_handle_t chandle, uint8_t *feedback, unsigned int length)
- int **statehandler_sbit** (statehandler_handle_t handle, compartment_handle_t chandle, unsigned int sbit)
- int **statehandler_ibit** (statehandler_handle_t handle, compartment_handle_t chandle, unsigned int ibit)
- int **statehandler_get_requested_feedback** (statehandler_handle_t handle, compartment_handle_t chandle, uint8_t **feedback, unsigned int *length)
- int **statehandler_set_returned_feedback** (statehandler_handle_t handle, compartment_handle_t chandle, uint8_t *feedback, unsigned int length)
- int **statehandler_get_returned_feedback** (statehandler_handle_t handle, compartment_handle_t chandle, uint8_t **feedback, unsigned int *length)

Typedef Documentation

typedef struct compartment_handle_data* **compartment_handle_t**

typedef struct statehandler_handle_data* **statehandler_handle_t**

Function Documentation

int statehandler_init (statehandler_handle_t * *handle*)

Initialize the StateHandler.

Parameters:

handle Pointer to opaque StateHandler handle.

Returns:

Returns 0 on success, or a non-zero value on error.

int statehandler_destroy (statehandler_handle_t *handle*)

Destroy the StateHandler resources. **statehandler_destroy()** must be called on the StateHandler handle to free StateHandler resources.

Parameters:

handle Opaque StateHandler handle.

Returns:

Returns 0 on success, or a non-zero value on error.

int statehandler_create_compartment (statehandler_handle_t *handle*, unsigned int *memory_size*, compartment_handle_t * *chandle*)

Create a new compartment in the StateHandler.

Parameters:

handle Opaque StateHandler handle.
memory_size Size of memory compartment (in bytes).
chandle Pointer to an opaque Compartment handle.

Returns:

Returns 0 on success, or a non-zero value on error.

int statehandler_close_compartment (statehandler_handle_t *handle*, compartment_handle_t *chandle*)

Close and release resources for a compartment.

Parameters:

handle Opaque StateHandler handle.
chandle Opaque Compartment handle.

Returns:

Returns 0 on success, or a non-zero value on error.

int statehandler_create_state (statehandler_handle_t *handle*, compartment_handle_t *chandle*, uint8_t * *id*, unsigned int *idlen*, uint8_t * *sbuf*, unsigned int *len*, unsigned int *statepri*)

Create a new state in the StateHandler. This function is commonly invoked as a callback from the UDVM.

Parameters:

handle Opaque StateHandler handle.
chandle Opaque Compartment handle.
id Pointer to the partial state identifier for the new state.
idlen Length of the partial state identifier (in bytes).
sbuf Pointer to a buffer containing the state to add to the state handler.
len Length of the state (in bytes).
statepri Priority to add new state to the StateHandler.

Returns:

Returns 0 on success, or a non-zero value on error.

int statehandler_free_state (statehandler_handle_t *handle*, compartment_handle_t *chandle*, uint8_t * *id*, unsigned int *idlen*)

Free the resources in the state handler associated with the partial state identifier. This function is commonly invoked as a callback from the UDVM.

Parameters:

handle Opaque StateHandler handle.
chandle Opaque Compartment handle.
id Pointer to the partial state identifier to free.
idlen Length of the partial state identifier (in bytes)

Returns:

Returns 0 on success, or a non-zero value on error.

int statehandler_access_state (statehandler_handle_t handle, uint8_t * id, unsigned int idlen, uint8_t ** sbuf, unsigned int * len)

Access, or retrieve, the state associated with the partial state identifier. This function is commonly invoked as a callback from the UDVM.

Parameters:

handle Opaque StateHandler handle.
id Pointer to the partial state identifier to access.
idlen Length of the partial state identifier (in bytes).
sbuf Pointer to load the pointer to the state.
len Length of the state (in bytes).

Returns:

Returns 0 on success, or a non-zero value on error.

int statehandler_set_requested_feedback (statehandler_handle_t handle, compartment_handle_t chandle, uint8_t * feedback, unsigned int length)

Save feedback data requested by the compressor into the StateHandler. This function is commonly invoked as a callback from the UDVM.

Parameters:

handle Opaque StateHandler handle.
chandle Opaque Compartment handle.
feedback Pointer to buffer with the feedback data.
length Length of the feedback data (in bytes).

Returns:

Returns 0 on success, or a non-zero value on error.

int statehandler_sbit (statehandler_handle_t handle, compartment_handle_t chandle, unsigned int sbit)

Handle setting of the UDVM S-bit. The compressor sets the S-bit if it does not wish (or no longer wishes) to save state information at the receiving endpoint and also does not wish to access state information that it has previously saved. This function is commonly invoked as a callback from the UDVM.

Parameters:

handle Opaque StateHandler handle.
chandle Opaque Compartment handle.
sbit Value of the UDVM S-bit.

Returns:

Returns 0 on success, or a non-zero value on error.

int statehandler_ibit (statehandler_handle_t handle, compartment_handle_t chandle, unsigned int ibit)

Callback to handle setting of the UDVM I-bit. The compressor sets the I-bit if it does not wish (or no longer wishes) to access any of the locally available state items offered by the receiving endpoint. This function is commonly invoked as a callback from the UDVM.

Parameters:

handle Opaque StateHandler handle.

chandle Opaque Compartment handle.
ibit Value of the UDVM I-bit.

Returns:

Returns 0 on success, or a non-zero value on error.

**int statehandler_get_requested_feedback (statehandler_handle_t *handle*,
compartment_handle_t *chandle*, uint8_t ** *feedback*, unsigned int * *length*)**

Retrieve any feedback data from our StateHandler which the remote compressor asked our UDVM to save. Typically, this function is called by the Compressor and this feedback data is inserted into the header of a SigComp message and returned to the remote endpoint.

Parameters:

handle Opaque StateHandler handle.
chandle Opaque Compartment handle.
feedback Pointer to store the pointer of the feedback data.
length Length of the feedback data.

Returns:

Returns 0 on success, or a non-zero value on error.

**int statehandler_set_returned_feedback (statehandler_handle_t *handle*,
compartment_handle_t *chandle*, uint8_t * *feedback*, unsigned int *length*)**

Save returned feedback data into the StateHandler that our compressor requested. Typically, this function is called by the Decompressor and this feedback data was extracted from the header of a message send to use by the remote endpoint.

Parameters:

handle Opaque StateHandler handle.
chandle Opaque Compartment handle.
feedback Pointer to buffer containing feedback data.
length Length of feedback data (in bytes).

**int statehandler_get_returned_feedback (statehandler_handle_t *handle*,
compartment_handle_t *chandle*, uint8_t ** *feedback*, unsigned int * *length*)**

Retrieve any feedback data from our StateHandler which we requested from the remote endpoint. Typically, this function is called by the Compressor and this feedback data is describes that algorithmic state of the remote decompressor.

Parameters:

handle Opaque StateHandler handle.
chandle Opaque Compartment handle.
feedback Pointer to store the pointer of the feedback data.
length Length of the feedback data.

Returns:

Returns 0 on success, or a non-zero value on error.

udvm.h File Reference

Detailed Description

udvm.h specifies the interface to the UDVM. Interaction with the UDVM is controlled with callbacks.

The low-level interface to the virtual machine is controlled through callbacks to the application for state handling and data processing. Generally, most users should be using the SigComp high-level interface.

The UDVM can only create a state item when a complete message has been successfully decompressed and the application has returned a compartment identifier under which the state can be saved.

Data Structures

- struct **udvm_profile_data**
UDVM profile/configuration.
- struct **udvm_decompressor_callback_data**
Callback/dispatcher table to Decompressor.
- struct **udvm_statehandler_callback_data**
Callback/dispatcher table to State Handler.

Defines

- #define **UDVM_SEEK_CUR** 1

Typedefs

- typedef udvm_handle_data * **udvm_handle_t**
- typedef **udvm_profile_data** **udvm_profile_t**
- typedef **udvm_decompressor_callback_data** **udvm_decompressor_callback_t**
- typedef **udvm_statehandler_callback_data** **udvm_statehandler_callback_t**

Functions

- int **udvm_init** (**udvm_profile_t** *profile, **udvm_decompressor_callback_t** *decompressor_callback, void *decompressor_callback_arg, **udvm_statehandler_callback_t** *statehandler_callback, void *statehandler_callback_arg, **udvm_handle_t** *handle)
 - int **udvm_load** (**udvm_handle_t** handle, uint8_t *bytecode, unsigned int length, unsigned int load_address, unsigned int start_instruction, unsigned int partial_state_len, unsigned int partial_state_id_len)
 - int **udvm_run** (**udvm_handle_t** handle)
 - int **udvm_destroy** (**udvm_handle_t** handle)
 - int **udvm_state_update** (**udvm_handle_t** handle, void *compartment_cookie)
-

Define Documentation

```
#define UDVM_SEEK_CUR 1
```

Typedef Documentation

```
typedef struct udvm_handle_data* udvm_handle_t
```

```
typedef struct udvm_profile_data udvm_profile_t
```

```
typedef struct udvm_decompressor_callback_data udvm_decompressor_callback_t
```

```
typedef struct udvm_statehandler_callback_data udvm_statehandler_callback_t
```

Function Documentation

```
int udvm_init (udvm_profile_t * profile, udvm_decompressor_callback_t *  
decompressor_callback, void * decompressor_callback_arg,  
udvm_statehandler_callback_t * statehandler_callback, void * statehandler_callback_arg,  
udvm_handle_t * handle)
```

Initialize the UDVM.

Parameters:

profile Pointer to UDVM profile.

decompressor_callback Pointer to callbacks to decompressor.

decompressor_callback_arg Opaque handle passed as first argument in decompressor callbacks.

statehandler_callback Pointer to callback to state handler.

statehandler_callback_arg Opaque handle passed as first argument in state handler callbacks.

handle Pointer to opaque UDVM handle.

Returns:

Returns 0 on success, or a non-zero value on error.

```
int udvm_load (udvm_handle_t handle, uint8_t * bytecode, unsigned int length, unsigned  
int load_address, unsigned int start_instruction, unsigned int partial_state_len, unsigned  
int partial_state_id_len)
```

Load state or bytecode into the UDVM

Parameters:

handle UDVM handle returned from `udvm_init()`.

bytecode Pointer to UDVM bytecode.

length Length of bytecode (in bytes).

load_address Load address of bytecode.

start_instruction Address to start VM execution.

partial_state_len Length of the state retrieved from the state handler. Set to 0 if the UDVM state was not retrieved from the state handler.

partial_state_id_len Length of the partial state identifier used to retrieve the the state from the state handler. Set to 0 if the UDVM state was not retrieved from the state handler.

Returns:

Returns 0 on success, or a non-zero value on error.

int udvm_run (udvm_handle_t handle)

Start the UDVM execution.

Parameters:

handle Opaque UDVM handle returned from **udvm_init()**.

Returns:

Returns 0 on success, or a non-zero value on error.

int udvm_destroy (udvm_handle_t handle)

Destroy the UDVM context and resources. **udvm_destory()** must be called on the UDVM handle to free UDVM resources.

Parameters:

handle Opaque UDVM handle.

Returns:

Returns 0 on success, or a non-zero value on error.

int udvm_state_update (udvm_handle_t handle, void * compartment_cookie)

Update the the state associated with the UDVM. Upon completion of UDVM execution, the caller should invoke **udvm_state_update()** to indicate that the SigComp message decompressed correctly and contains valid data, and the state operations may be committed to state handler.

Parameters:

handle Opaque UDVM handle.

compartment_cookie Opaque compartment handle passed to the state handler.

Returns:

Returns 0 on success, or a non-zero value on error.

Sigcomp Stack Example Documentation

basic.c

This is a simple example of using the high-level interface.

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <assert.h>
4 #include <string.h>
5
6 #include <sys/types.h>
7
8 #ifndef WIN32
9 #include <stdint.h>
10 #include <unistd.h> /* close */
11 #include <sys/socket.h>
12 #include <sys/time.h>
13 #include <netinet/in.h>
14 #include <arpa/inet.h>
15 #include <netdb.h>
16 #else
17 #include "winsock.h"
18 #endif
19
20 #include "lzss.h"
21 #include "sigcomp.h"
22
23 char request[] =
24 "INVITE sip:bill@example.com SIP/2.0\r\n"
25 "To: <sip:bill@example.com>\r\n"
26 "From: Bill Smith<sip:bill@example.com>;tag=4b9eac27\r\n"
27 "Via: SIP/2.0/UDP 123.45.67.89:20948;branch=z9hG4bK-d87543-823649442-1--d87543-
;rport\r\n"
28 "Call-ID: 5a03face66f75daf\r\n"
29 "CSeq: 1 INVITE\r\n"
30 "Contact: <sip:bill@123.45.67.89:20948>\r\n"
31 "Max-Forwards: 70\r\n"
32 "Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, REFER, NOTIFY, MESSAGE, SUBSCRIBE,
INFO\r\n"
33 "Content-Type: application/sdp\r\n"
34 "User-Agent: Cisco ATA v.31-525\r\n"
35 "Content-Length: 336\r\n"
36 "\r\n"
37 "v=0\r\n"
38 "o=- 382112951 382113079 IN IP4 123.45.67.89\r\n"
39 "s=cisco\r\n"
40 "c=IN IP4 123.45.67.89\r\n"
41 "t=0 0\r\n"
42 "m=audio 20950 RTP/AVP 100 3 8 97 101\r\n"
43 "a=alt:1 1 : C99F10BA 8B000000 123.45.67.89 20950\r\n"
44 "a=alt:2 3 : C992C917 09000000 192.168.1.2 8990\r\n"
45 "a=fmtp:101 0-15\r\n"
46 "a=rtpmap:100 iLBC/8000\r\n"
47 "a=rtpmap:97 speex/8000\r\n"
48 "a=rtpmap:101 telephone-event/8000\r\n"
49 "a=sendrecv\r\n";
50
51
52 #define BUF_SIZ 2048
53
54 static unsigned char buf1[BUF_SIZ];
55 static unsigned char buf2[BUF_SIZ];
56
57 sigcomp_session_handle_t session_handle;
58 compressor_handle_t compressor_handle;
```

```

59 decompressor_handle_t decompressor_handle;
60
61 int
62 main (int argc, char *argv[])
63 {
64     int failed;
65     int i, n;
66
67     failed = sigcomp_init();
68     assert(!failed);
69
70 #define USE_DICTIONARY 1
71     failed = lzss_init(&compressor_handle, USE_DICTIONARY);
72     assert(!failed);
73
74 #define STATE_MEMORY_SIZE 8192
75     failed = sigcomp_new_session(compressor_handle, STATE_MEMORY_SIZE,
&session_handle);
76     assert(!failed);
77
78     /* compress first */
79     printf("1: uncompressed size = %d\n", (int)sizeof(request));
80     n = sigcomp_compress(session_handle,
81         request, sizeof(request),
82         buf1, BUF_SIZ);
83     printf("1: compressed size = %d (bytecode, no feedback)\n", n);
84     /* now decompress */
85     i = sigcomp_decompress(buf1, n, buf2, BUF_SIZ, NULL, &decompressor_handle);
86     printf("1: decompressed size = %d\n", i);
87     failed = sigcomp_accept(decompressor_handle, session_handle);
88     assert(!failed);
89
90     /* compress first */
91     printf("2: uncompressed size = %d\n", (int)sizeof(request));
92     n = sigcomp_compress(session_handle,
93         request, sizeof(request),
94         buf1, BUF_SIZ);
95     printf("2: compressed size = %d (bytecode, feedback)\n", n);
96     /* now decompress */
97     i = sigcomp_decompress(buf1, n, buf2, BUF_SIZ, NULL, &decompressor_handle);
98     printf("2: decompressed size = %d\n", i);
99     failed = sigcomp_accept(decompressor_handle, session_handle);
100     assert(!failed);
101
102     /* compress first */
103     printf("3: uncompressed size = %d\n", (int)sizeof(request));
104     n = sigcomp_compress(session_handle,
105         request, sizeof(request),
106         buf1, BUF_SIZ);
107     printf("3: compressed size = %d (no bytecode, feedback)\n", n);
108     /* now decompress */
109     i = sigcomp_decompress(buf1, n, buf2, BUF_SIZ, NULL, &decompressor_handle);
110     printf("3: decompressed size = %d\n", i);
111     failed = sigcomp_accept(decompressor_handle, session_handle);
112     assert(!failed);
113
114     lzss_free(compressor_handle);
115
116     return 0;
117
118 }

```

