

sonata

Generated by Doxygen 1.5.7

Fri Nov 28 10:52:02 2008

Contents

1	Main Page	1
2	Module Index	3
2.1	Modules	3
3	Data Structure Index	5
3.1	Data Structures	5
4	File Index	7
4.1	File List	7
5	Module Documentation	9
5.1	External code changes and modifications	9
5.1.1	Detailed Description	9
5.1.2	Function Documentation	9
5.1.2.1	FindSctpRedirectAddress	9
5.1.2.2	SctpShowAliasStats	10
5.2	SCTP Packet Parsing	11
5.2.1	Detailed Description	12
5.2.2	Define Documentation	12
5.2.2.1	SN_ASCONFACK_PARAM_SIZE	12
5.2.2.2	SN_MIN_CHUNK_SIZE	12
5.2.2.3	SN_MIN_PARAM_SIZE	12
5.2.2.4	SN_PARSE_ERROR_AS_MALLOC	12
5.2.2.5	SN_PARSE_ERROR_CHHL	13
5.2.2.6	SN_PARSE_ERROR_CHUNK	13
5.2.2.7	SN_PARSE_ERROR_DIR	13
5.2.2.8	SN_PARSE_ERROR_IPSHL	13
5.2.2.9	SN_PARSE_ERROR_LOOKUP	13
5.2.2.10	SN_PARSE_ERROR_LOOKUP_ABORT	13

5.2.2.11	SN_PARSE_ERROR_PARTIALLOOKUP	13
5.2.2.12	SN_PARSE_ERROR_PORT	13
5.2.2.13	SN_PARSE_ERROR_VTAG	14
5.2.2.14	SN_PARSE_OK	14
5.2.2.15	SN_SCTP_ABORT	14
5.2.2.16	SN_SCTP_ASCONF	14
5.2.2.17	SN_SCTP_ASCONFACK	14
5.2.2.18	SN_SCTP_FIRSTCHUNK	14
5.2.2.19	SN_SCTP_INIT	14
5.2.2.20	SN_SCTP_INITACK	15
5.2.2.21	SN_SCTP_NEXTCHUNK	15
5.2.2.22	SN_SCTP_NEXTPARAM	15
5.2.2.23	SN_SCTP_OTHER	15
5.2.2.24	SN_SCTP_SHUTACK	15
5.2.2.25	SN_SCTP_SHUTCOMP	15
5.2.2.26	SN_VTAG_PARAM_SIZE	15
5.2.3	Function Documentation	16
5.2.3.1	AddGlobalIPAddresses	16
5.2.3.2	GetAsconfVtags	18
5.2.3.3	IsADDorDEL	19
5.2.3.4	IsASCONFack	20
5.2.3.5	RmGlobalIPAddresses	21
5.2.3.6	sctp_PktParser	22
5.3	SCTP NAT State Machine	27
5.3.1	Detailed Description	27
5.3.2	Define Documentation	28
5.3.2.1	SN_CL	28
5.3.2.2	SN_ID	28
5.3.2.3	SN_INa	28
5.3.2.4	SN_INi	28
5.3.2.5	SN_RM	28
5.3.2.6	SN_UP	28
5.3.3	Function Documentation	28
5.3.3.1	CL_process	28
5.3.3.2	ID_process	29
5.3.3.3	INa_process	30

5.3.3.4	INi_process	31
5.3.3.5	ProcessSctpMsg	32
5.3.3.6	UP_process	33
5.4	Logging Functionality	35
5.4.1	Detailed Description	35
5.4.2	Define Documentation	35
5.4.2.1	SN_LOG	35
5.4.2.2	SN_LOG_DEBUG	36
5.4.2.3	SN_LOG_DEBUG_MAX	36
5.4.2.4	SN_LOG_DETAIL	36
5.4.2.5	SN_LOG_EVENT	36
5.4.2.6	SN_LOG_INFO	36
5.4.2.7	SN_LOG_LOW	36
5.4.3	Function Documentation	36
5.4.3.1	logsctpassoc	36
5.4.3.2	logsctperror	37
5.4.3.3	logSctpGlobal	38
5.4.3.4	logSctpLocal	38
5.4.3.5	logsctpparse	38
5.4.3.6	logTimerQ	39
5.4.3.7	SctpAliasLog	39
5.5	Hash Table Macros and Functions	41
5.5.1	Detailed Description	42
5.5.2	Define Documentation	42
5.5.2.1	SN_ADD_CLASH	42
5.5.2.2	SN_ADD_OK	42
5.5.2.3	SN_BOTH_TBL	42
5.5.2.4	SN_DEFAULT_HASH_SIZE	42
5.5.2.5	SN_GLOBAL_TBL	42
5.5.2.6	SN_LOCAL_TBL	42
5.5.2.7	SN_MAX_GLOBAL_ADDRESSES	43
5.5.2.8	SN_MAX_HASH_SIZE	43
5.5.2.9	SN_MIN_HASH_SIZE	43
5.5.2.10	SN_NULL_TBL	43
5.5.2.11	SN_TABLE_HASH	43
5.5.2.12	SN_WAIT_TOGLOBAL	43

5.5.2.13	SN_WAIT_TOLOCAL	43
5.5.3	Function Documentation	43
5.5.3.1	AddSctpAssocGlobal	43
5.5.3.2	AddSctpAssocLocal	44
5.5.3.3	FindSctpGlobal	45
5.5.3.4	FindSctpGlobalClash	46
5.5.3.5	FindSctpGlobalT	47
5.5.3.6	FindSctpLocal	48
5.5.3.7	FindSctpLocalT	49
5.5.3.8	freeGlobalAddressList	49
5.5.3.9	RmSctpAssoc	50
5.6	Timer Queue Macros and Functions	52
5.6.1	Detailed Description	52
5.6.2	Define Documentation	52
5.6.2.1	SN_C_T	52
5.6.2.2	SN_I_T	53
5.6.2.3	SN_MAX_TIMER	53
5.6.2.4	SN_MIN_TIMER	53
5.6.2.5	SN_TIMER_QUEUE_SIZE	53
5.6.2.6	SN_U_T	53
5.6.2.7	SN_X_T	53
5.6.3	Function Documentation	53
5.6.3.1	sctp_AddTimeOut	53
5.6.3.2	sctp_CheckTimers	54
5.6.3.3	sctp_ResetTimeOut	54
5.6.3.4	sctp_RmTimeOut	55
5.7	SysCtl Variable and callback function declarations	56
5.7.1	Detailed Description	57
5.7.2	Define Documentation	57
5.7.2.1	SN_ERROR_ON_OOTB	57
5.7.2.2	SN_LOCAL_ERROR_ON_OOTB	57
5.7.2.3	SN_LOCALandPARTIAL_ERROR_ON_OOTB	58
5.7.2.4	SN_NO_ERROR_ON_OOTB	58
5.7.3	Function Documentation	58
5.7.3.1	sysctl_chg_accept_global_ootb_addip	58
5.7.3.2	sysctl_chg_chunk_proc_limit	58

5.7.3.3	sysctl_chg_error_on_ootb	58
5.7.3.4	sysctl_chg_hashtable_size	59
5.7.3.5	sysctl_chg_initialising_chunk_proc_limit	59
5.7.3.6	sysctl_chg_loglevel	59
5.7.3.7	sysctl_chg_param_proc_limit	60
5.7.3.8	sysctl_chg_timer	60
5.7.3.9	sysctl_chg_track_global_addresses	60
5.7.4	Variable Documentation	61
5.7.4.1	sysctl_accept_global_ootb_addip	61
5.7.4.2	sysctl_chunk_proc_limit	61
5.7.4.3	sysctl_error_on_ootb	61
5.7.4.4	sysctl_hashtable_size	61
5.7.4.5	sysctl_holddown_timer	61
5.7.4.6	sysctl_init_timer	61
5.7.4.7	sysctl_initialising_chunk_proc_limit	62
5.7.4.8	sysctl_log_level	62
5.7.4.9	sysctl_param_proc_limit	62
5.7.4.10	sysctl_shutdown_timer	62
5.7.4.11	sysctl_track_global_addresses	62
5.7.4.12	sysctl_up_timer	62
6	Data Structure Documentation	63
6.1	sctp_GlobalAddress Struct Reference	63
6.1.1	Detailed Description	63
6.1.2	Member Function Documentation	63
6.1.2.1	LIST_ENTRY	63
6.1.3	Field Documentation	63
6.1.3.1	g_addr	63
6.2	sctp_nat_assoc Struct Reference	64
6.2.1	Detailed Description	64
6.2.2	Member Function Documentation	64
6.2.2.1	LIST_ENTRY	64
6.2.2.2	LIST_ENTRY	64
6.2.2.3	LIST_ENTRY	64
6.2.2.4	LIST_HEAD	65
6.2.3	Field Documentation	65
6.2.3.1	a_addr	65

6.2.3.2	exp	65
6.2.3.3	exp_loc	65
6.2.3.4	g_port	65
6.2.3.5	g_vtag	65
6.2.3.6	l_addr	65
6.2.3.7	l_port	65
6.2.3.8	l_vtag	66
6.2.3.9	num_Gaddr	66
6.2.3.10	state	66
6.2.3.11	TableRegister	66
6.3	sctp_nat_msg Struct Reference	67
6.3.1	Detailed Description	67
6.3.2	Field Documentation	67
6.3.2.1	chunk_length	67
6.3.2.2	ip_hdr	67
6.3.2.3	msg	67
6.3.2.4	sctp_hdr	68
6.3.2.5	sctpchnk	68
6.4	sctp_nat_timer Struct Reference	69
6.4.1	Detailed Description	69
6.4.2	Member Function Documentation	69
6.4.2.1	LIST_HEAD	69
6.4.3	Field Documentation	69
6.4.3.1	cur_loc	69
6.4.3.2	loc_time	69
6.5	sctpChunkOfInt Union Reference	70
6.5.1	Detailed Description	70
6.5.2	Field Documentation	70
6.5.2.1	Asconf	70
6.5.2.2	Init	70
6.5.2.3	InitAck	70
7	File Documentation	71
7.1	alias_sctp.c File Reference	71
7.1.1	Detailed Description	77
7.1.2	Define Documentation	78
7.1.2.1	SCTP_MIDDLEBOX_FLAG	78

7.1.2.2	SCTP_MISSING_NAT	78
7.1.2.3	SCTP_NAT_TABLE_COLLISION	78
7.1.2.4	SCTP_VTAG_PARAM	78
7.1.2.5	sn_calloc	78
7.1.2.6	sn_free	78
7.1.2.7	sn_malloc	78
7.1.3	Function Documentation	78
7.1.3.1	Add_Global_Address_to_List	78
7.1.3.2	AliasSctpInit	79
7.1.3.3	AliasSctpTerm	79
7.1.3.4	SctpAlias	80
7.1.3.5	TxAbortErrorM	82
7.2	alias_sctp.h File Reference	85
7.2.1	Detailed Description	86
7.2.2	Define Documentation	87
7.2.2.1	LINK_SCTP	87
7.2.2.2	PKT_ALIAS_RESPOND	87
7.2.2.3	SCTP_PACKED	87
7.2.2.4	SCTP_UNUSED	87
7.2.2.5	SN_DROP_PKT	87
7.2.2.6	SN_NAT_PKT	87
7.2.2.7	SN_PROCESSING_ERROR	87
7.2.2.8	SN_REFLECT_ERROR	88
7.2.2.9	SN_REPLY_ABORT	88
7.2.2.10	SN_REPLY_ERROR	88
7.2.2.11	SN_SEND_ABORT	88
7.2.2.12	SN_TO_GLOBAL	88
7.2.2.13	SN_TO_LOCAL	88
7.2.2.14	SN_TO_NODIR	88
7.2.2.15	SN_TX_ABORT	88
7.2.2.16	SN_TX_ERROR	89

Chapter 1

Main Page

Alias_sctp is part of the SONATA (<http://caia.swin.edu.au/urp/sonata>) project to develop and release a BSD licensed implementation of a Network Address Translation (NAT) module that supports the Stream Control Transmission Protocol (SCTP).

Traditional address and port number look ups are inadequate for SCTP's operation due to both processing requirements and issues with multi-homing. Alias_sctp integrates with FreeBSD's ipfw/libalias NAT system.

Version 0.2 features include:

- Support for global multi-homing
- Support for ASCONF modification from Internet Draft (draft-stewart-behave-sctpnat-04, R. Stewart and M. Tuexen, "Stream control transmission protocol (SCTP) network address translation," Jul. 2008) to provide support for multi-homed privately addressed hosts
- Support for forwarding of T-flagged packets
- Generation and delivery of AbortM/ErrorM packets upon detection of NAT collisions
- Per-port forwarding rules
- Dynamically controllable logging and statistics
- Dynamic management of timers
- Dynamic control of hash-table size

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

External code changes and modifications	9
SCTP Packet Parsing	11
SCTP NAT State Machine	27
Logging Functionality	35
Hash Table Macros and Functions	41
Timer Queue Macros and Functions	52
SysCtl Variable and callback function declarations	56

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

sctp_GlobalAddress	63
sctp_nat_assoc (Sctp association information)	64
sctp_nat_msg (SCTP message)	67
sctp_nat_timer (Sctp nat timer queue structure)	69
sctpChunkOfInt (SCTP chunk of interest)	70

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

alias_sctp.c	71
alias_sctp.h	85

Chapter 5

Module Documentation

5.1 External code changes and modifications

Functions

- void [SctpShowAliasStats](#) (struct libalias *la)
Log current statistics for the libalias instance.
- struct in_addr [FindSctpRedirectAddress](#) (struct libalias *la, struct sctp_nat_msg *sm)
Find the address to redirect incoming packets.

5.1.1 Detailed Description

Some changes have been made to files external to alias_sctp.(c|h). These changes are primarily due to code needing to call static functions within those files or to perform extra functionality that can only be performed within these files.

5.1.2 Function Documentation

5.1.2.1 struct in_addr FindSctpRedirectAddress (struct libalias * la, struct sctp_nat_msg * sm) [read]

Find the address to redirect incoming packets.

This function is defined in alias_db.c, since it calls static functions in this file

Given a destination port for incoming packets to the NAT, discover what (if any) internal IP address this packet should be re-directed to

Parameters:

- la* Pointer to the libalias instance
- sm* Pointer to the incoming message

Returns:

- Address to redirect an incoming INIT to

5.1.2.2 void SctpShowAliasStats (struct libalias * *la*)

Log current statistics for the libalias instance.

This function is defined in alias_db.c, since it calls static functions in this file

Calls the higher level ShowAliasStats() in alias_db.c which logs all current statistics about the libalias instance - including SCTP statistics

Parameters:

la Pointer to the libalias instance

5.2 SCTP Packet Parsing

Defines

- #define `SN_SCTP_FIRSTCHUNK`(sctphead) (struct sctp_chunkhdr *)(((char *)sctphead) + sizeof(struct sctphdr))
- #define `SN_SCTP_NEXTCHUNK`(chunkhead) (struct sctp_chunkhdr *)(((char *)chunkhead) + SCTP_SIZE32(ntohs(chunkhead → chunk_length)))
- #define `SN_SCTP_NEXTPARAM`(param) (struct sctp_paramhdr *)(((char *)param) + SCTP_SIZE32(ntohs(param → param_length)))
- #define `SN_MIN_CHUNK_SIZE` 4
- #define `SN_MIN_PARAM_SIZE` 4
- #define `SN_VTAG_PARAM_SIZE` 12
- #define `SN_ASCONFACK_PARAM_SIZE` 8
- #define `SN_PARSE_OK` 0
- #define `SN_PARSE_ERROR_IPSHL` 1
- #define `SN_PARSE_ERROR_AS_MALLOC` 2
- #define `SN_PARSE_ERROR_CHHL` 3
- #define `SN_PARSE_ERROR_DIR` 4
- #define `SN_PARSE_ERROR_VTAG` 5
- #define `SN_PARSE_ERROR_CHUNK` 6
- #define `SN_PARSE_ERROR_PORT` 7
- #define `SN_PARSE_ERROR_LOOKUP` 8
- #define `SN_PARSE_ERROR_PARTIALLOOKUP` 9
- #define `SN_PARSE_ERROR_LOOKUP_ABORT` 10
- #define `SN_SCTP_ABORT` 0x0000
- #define `SN_SCTP_INIT` 0x0001
- #define `SN_SCTP_INITACK` 0x0002
- #define `SN_SCTP_SHUTCOMP` 0x0010
- #define `SN_SCTP_SHUTACK` 0x0020
- #define `SN_SCTP_ASCONF` 0x0100
- #define `SN_SCTP_ASCONFACK` 0x0200
- #define `SN_SCTP_OTHER` 0xFFFF

Functions

- static int `sctp_PktParser` (struct libalias *la, int direction, struct ip *pip, struct `sctp_nat_msg` *sm, struct `sctp_nat_assoc` **passoc)
Parses SCTP packets for the key SCTP chunk that will be processed.
- static int `GetAsconfVtags` (struct libalias *la, struct `sctp_nat_msg` *sm, uint32_t *l_vtag, uint32_t *g_vtag, int direction)
Extract Vtags from Asconf Chunk.
- static void `AddGlobalIPAddresses` (struct `sctp_nat_msg` *sm, struct `sctp_nat_assoc` *assoc, int direction)
AddGlobalIPAddresses from Init, InitAck, or AddIP packets.
- static void `RmGlobalIPAddresses` (struct `sctp_nat_msg` *sm, struct `sctp_nat_assoc` *assoc, int direction)

RmGlobalIPAddresses from DelIP packets.

- static int [IsASCONFack](#) (struct libalias *la, struct [sctp_nat_msg](#) *sm, int direction)
Check that ASCONF was successful.
- static int [IsADDorDEL](#) (struct libalias *la, struct [sctp_nat_msg](#) *sm, int direction)
Check to see if ASCONF contains an Add IP or Del IP parameter.

5.2.1 Detailed Description

Macros to:

- Return pointers to the first and next SCTP chunks within an SCTP Packet
- Define possible return values of the packet parsing process
- SCTP message types for storing in the [sctp_nat_msg](#) structure

These functions parse the SCTP packet and fill a [sctp_nat_msg](#) structure with the parsed contents.

5.2.2 Define Documentation

5.2.2.1 #define SN_ASCONFACK_PARAM_SIZE 8

Size of SCTP ASCONF ACK param in bytes

Definition at line 245 of file alias_sctp.c.

5.2.2.2 #define SN_MIN_CHUNK_SIZE 4

Smallest possible SCTP chunk size in bytes

Definition at line 242 of file alias_sctp.c.

Referenced by [sctp_PktParser\(\)](#).

5.2.2.3 #define SN_MIN_PARAM_SIZE 4

Smallest possible SCTP param size in bytes

Definition at line 243 of file alias_sctp.c.

5.2.2.4 #define SN_PARSE_ERROR_AS_MALLOC 2

Packet parsing error - assoc malloc

Definition at line 250 of file alias_sctp.c.

Referenced by [sctp_PktParser\(\)](#).

5.2.2.5 #define SN_PARSE_ERROR_CHHL 3

Packet parsing error - Chunk header len

Definition at line 251 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.6 #define SN_PARSE_ERROR_CHUNK 6

Packet parsing error - Chunk

Definition at line 254 of file alias_sctp.c.

5.2.2.7 #define SN_PARSE_ERROR_DIR 4

Packet parsing error - Direction

Definition at line 252 of file alias_sctp.c.

5.2.2.8 #define SN_PARSE_ERROR_IPSHL 1

Packet parsing error - IP and SCTP common header len

Definition at line 249 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.9 #define SN_PARSE_ERROR_LOOKUP 8

Packet parsing error - Lookup

Definition at line 256 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.10 #define SN_PARSE_ERROR_LOOKUP_ABORT 10

Packet parsing error - Lookup - but abort packet

Definition at line 258 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.11 #define SN_PARSE_ERROR_PARTIALLOOKUP 9

Packet parsing error - partial lookup only found

Definition at line 257 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.12 #define SN_PARSE_ERROR_PORT 7

Packet parsing error - Port=0

Definition at line 255 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.13 #define SN_PARSE_ERROR_VTAG 5

Packet parsing error - Vtag

Definition at line 253 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.14 #define SN_PARSE_OK 0

Packet parsed for SCTP messages

Definition at line 248 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.15 #define SN_SCTP_ABORT 0x0000

a packet containing an ABORT chunk

Definition at line 261 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.16 #define SN_SCTP_ASCONF 0x0100

a packet containing an ASCONF chunk

Definition at line 266 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.17 #define SN_SCTP_ASCONFACK 0x0200

a packet containing an ASCONF-ACK chunk

Definition at line 267 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.18 #define SN_SCTP_FIRSTCHUNK(sctphead) (struct sctp_chunkhdr *)(((char *)sctphead) + sizeof(struct sctphdr))

Returns a pointer to the first chunk in an SCTP packet given a pointer to the SCTP header

Definition at line 230 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.19 #define SN_SCTP_INIT 0x0001

a packet containing an INIT chunk

Definition at line 262 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.20 #define SN_SCTP_INITACK 0x0002

a packet containing an INIT-ACK chunk

Definition at line 263 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.21 #define SN_SCTP_NEXTCHUNK(chunkhead) (struct sctp_chunkhdr *)(((char *)chunkhead) + SCTP_SIZE32(ntohs(chunkhead) → chunk_length))

Returns a pointer to the next chunk in an SCTP packet given a pointer to the current chunk

Definition at line 234 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.22 #define SN_SCTP_NEXTPARAM(param) (struct sctp_paramhdr *)(((char *)param) + SCTP_SIZE32(ntohs(param) → param_length))

Returns a pointer to the next parameter in an SCTP packet given a pointer to the current parameter

Definition at line 238 of file alias_sctp.c.

5.2.2.23 #define SN_SCTP_OTHER 0xFFFF

a packet containing a chunk that is not of interest

Definition at line 268 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.24 #define SN_SCTP_SHUTACK 0x0020

a packet containing a SHUTDOWN-ACK chunk

Definition at line 265 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.25 #define SN_SCTP_SHUTCOMP 0x0010

a packet containing a SHUTDOWN-COMPLETE chunk

Definition at line 264 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.2.2.26 #define SN_VTAG_PARAM_SIZE 12

Size of SCTP ASCONF vtag param in bytes

Definition at line 244 of file alias_sctp.c.

5.2.3 Function Documentation

5.2.3.1 static void AddGlobalIPAddresses (struct sctp_nat_msg * sm, struct sctp_nat_assoc * assoc, int direction) [static]

AddGlobalIPAddresses from Init,InitAck,or AddIP packets.

AddGlobalIPAddresses scans an SCTP chunk (in sm) for Global IP addresses, and adds them.

Parameters:

sm Pointer to sctp message information

assoc Pointer to the association this SCTP Message belongs to

direction SN_TO_LOCAL | SN_TO_GLOBAL

Definition at line 1297 of file alias_sctp.c.

```

1299                                     {0};
1300 int bytes_left = 0;
1301 int param_size;
1302 int param_count, addr_param_count = 0;
1303
1304 switch(direction) {
1305 case SN_TO_GLOBAL: /* does not contain global addresses */
1306     g_addr = sm->ip_hdr->ip_dst;
1307     bytes_left = 0; /* force exit */
1308     break;
1309 case SN_TO_LOCAL:
1310     g_addr = sm->ip_hdr->ip_src;
1311     param_count = 1;
1312     switch(sm->msg) {
1313     case SN_SCTP_INIT:
1314         bytes_left = sm->chunk_length - sizeof(struct sctp_init_chunk);
1315         param = (struct sctp_paramhdr *)((char *)sm->sctpchnk.Init + sizeof(struct sctp_init));
1316         break;
1317     case SN_SCTP_INITACK:
1318         bytes_left = sm->chunk_length - sizeof(struct sctp_init_ack_chunk);
1319         param = (struct sctp_paramhdr *)((char *)sm->sctpchnk.InitAck + sizeof(struct sctp_init_ack));
1320         break;
1321     case SN_SCTP_ASCONF:
1322         bytes_left = sm->chunk_length;
1323         param = sm->sctpchnk.Asconf;
1324         break;
1325     }
1326 }
1327 if (bytes_left >= SN_MIN_PARAM_SIZE)
1328     param_size = SctpSize32(ntohs(param->param_length));
1329 else
1330     param_size = bytes_left+1; /* force skip loop */
1331
1332 if ((assoc->state == SN_ID) && ((sm->msg == SN_SCTP_INIT) || (bytes_left < SN_MIN_PARAM_SIZE))) /*
1333     G_Addr = (struct sctp_GlobalAddress *) sn_malloc(sizeof(struct sctp_GlobalAddress));
1334     if (G_Addr == NULL) /* out of resources */
1335         SN_LOG(SN_LOG_EVENT,
1336             log_sctperror("AddGlobalIPAddress: No resources for adding global address - revert to no
1337                 sm->sctp_hdr->v_tag, 0, direction));
1338     assoc->num_Gaddr = 0; /* don't track any more for this assoc*/
1339     sysctl_track_global_addresses=0;
1340     return;
1341 }

```

```

1342     G_Addr->g_addr = g_addr;
1343     if (!Add_Global_Address_to_List (assoc, G_Addr))
1344         SN_LOG (SN_LOG_EVENT,
1345             logsctperror ("AddGlobalIPAddress: Address already in list",
1346                 sm->sctp_hdr->v_tag,  assoc->num_Gaddr, direction));
1347 }
1348
1349 /* step through parameters */
1350 while ((bytes_left >= param_size) && (bytes_left >= sizeof (struct sctp_ipv4addr_param))) {
1351     if (assoc->num_Gaddr >= sysctl_track_global_addresses) {
1352         SN_LOG (SN_LOG_EVENT,
1353             logsctperror ("AddGlobalIPAddress: Maximum Number of addresses reached",
1354                 sm->sctp_hdr->v_tag,  sysctl_track_global_addresses, direction));
1355         return;
1356     }
1357     switch (ntohs (param->param_type)) {
1358     case SCTP_ADD_IP_ADDRESS:
1359         /* skip to address parameter - leave param_size so bytes left will be calculated properly*/
1360         param = (struct sctp_paramhdr *) &((struct sctp_asconf_addrv4_param *) param)->addrp;
1361     case SCTP_IPV4_ADDRESS:
1362         ipv4_param = (struct sctp_ipv4addr_param *) param;
1363         /* add addresses to association */
1364         G_Addr = (struct sctp_GlobalAddress *) sn_malloc (sizeof (struct sctp_GlobalAddress));
1365         if (G_Addr == NULL) { /* out of resources */
1366             SN_LOG (SN_LOG_EVENT,
1367                 logsctperror ("AddGlobalIPAddress: No resources for adding global address - revert to 0",
1368                     sm->sctp_hdr->v_tag,  0, direction));
1369             assoc->num_Gaddr = 0; /* don't track any more for this assoc*/
1370             sysctl_track_global_addresses=0;
1371             return;
1372         }
1373         /* add address */
1374         addr_param_count++;
1375         if ((sm->msg == SN_SCTP_ASCONF) && (ipv4_param->addr == INADDR_ANY)) { /* use packet address */
1376             G_Addr->g_addr = g_addr;
1377             if (!Add_Global_Address_to_List (assoc, G_Addr))
1378                 SN_LOG (SN_LOG_EVENT,
1379                     logsctperror ("AddGlobalIPAddress: Address already in list",
1380                         sm->sctp_hdr->v_tag,  assoc->num_Gaddr, direction));
1381             return; /* shouldn't be any other addresses if the zero address is given*/
1382         } else {
1383             G_Addr->g_addr.s_addr = ipv4_param->addr;
1384             if (!Add_Global_Address_to_List (assoc, G_Addr))
1385                 SN_LOG (SN_LOG_EVENT,
1386                     logsctperror ("AddGlobalIPAddress: Address already in list",
1387                         sm->sctp_hdr->v_tag,  assoc->num_Gaddr, direction));
1388         }
1389     }
1390
1391     bytes_left -= param_size;
1392     if (bytes_left < SN_MIN_PARAM_SIZE)
1393         break;
1394
1395     param = SN_SCTP_NEXTPARAM (param);
1396     param_size = SCTP_SIZE32 (ntohs (param->param_length));
1397     if (++param_count > sysctl_param_proc_limit) {
1398         SN_LOG (SN_LOG_EVENT,
1399             logsctperror ("Parameter parse limit exceeded (AddGlobalIPAddress)",
1400                 sm->sctp_hdr->v_tag,  sysctl_param_proc_limit, direction));
1401         break; /* limit exceeded*/
1402     }
1403 }
1404 if (addr_param_count == 0) {
1405     SN_LOG (SN_LOG_DETAIL,
1406         logsctperror ("AddGlobalIPAddress: no address parameters to add",
1407             sm->sctp_hdr->v_tag,  assoc->num_Gaddr, direction));
1408 }

```

```
1409 }
1410
```

5.2.3.2 static int GetAsconfVtags (struct libalias * *la*, struct sctp_nat_msg * *sm*, uint32_t * *l_vtag*, uint32_t * *g_vtag*, int *direction*) [static]

Extract Vtags from Asconf Chunk.

GetAsconfVtags scans an Asconf Chunk for the vtags parameter, and then extracts the vtags.

GetAsconfVtags is not called from within sctp_PktParser. It is called only from within ID_process when an AddIP has been received.

Parameters:

la Pointer to the relevant libalias instance

sm Pointer to sctp message information

l_vtag Pointer to the local vtag in the association this SCTP Message belongs to

g_vtag Pointer to the local vtag in the association this SCTP Message belongs to

direction SN_TO_LOCAL | SN_TO_GLOBAL

Returns:

1 - success | 0 - fail

Definition at line 1230 of file alias_sctp.c.

```
1231     {
1232     struct sctp_paramhdr ph; /* type=SCTP_VTAG_PARAM */
1233     uint32_t local_vtag;
1234     uint32_t remote_vtag;
1235     } __attribute__((packed));
1236
1237 struct sctp_vtag_param *vtag_param;
1238 struct sctp_paramhdr *param;
1239 int bytes_left;
1240 int param_size;
1241 int param_count;
1242
1243 param_count = 1;
1244 param = sm->sctpchnk.Asconf;
1245 param_size = SCTP_SIZE32(ntohs(param->param_length));
1246 bytes_left = sm->chunk_length;
1247 /* step through Asconf parameters */
1248 while((bytes_left >= param_size) && (bytes_left >= SN_VTAG_PARAM_SIZE)) {
1249     if (ntohs(param->param_type) == SCTP_VTAG_PARAM) {
1250         vtag_param = (struct sctp_vtag_param *) param;
1251         switch(direction) {
1252             /* The Internet draft is a little ambiguous as to order of these vtags.
1253              * We think it is this way around. If we are wrong, the order will need
1254              * to be changed. */
1255             case SN_TO_GLOBAL:
1256                 *g_vtag = vtag_param->local_vtag;
1257                 *l_vtag = vtag_param->remote_vtag;
1258                 break;
1259             case SN_TO_LOCAL:
1260                 *g_vtag = vtag_param->remote_vtag;
1261                 *l_vtag = vtag_param->local_vtag;
1262                 break;
```

```

1263     }
1264     return(1); /* found */
1265 }
1266
1267 bytes_left -= param_size;
1268 if (bytes_left < SN_MIN_PARAM_SIZE) return(0);
1269
1270 param = SN_SCTP_NEXTPARAM(param);
1271 param_size = SCTP_SIZE32(ntohs(param->param_length));
1272 if (++param_count > sysctl_param_proc_limit) {
1273     SN_LOG(SN_LOG_EVENT,
1274         logsctperror("Parameter parse limit exceeded (GetAsconfVtags)",
1275             sm->sctp_hdr->v_tag, sysctl_param_proc_limit, direction));
1276     return(0); /* not found limit exceeded*/
1277 }
1278 }
1279 return(0); /* not found */
1280 }
1281

```

5.2.3.3 static int IsADDorDEL (struct libalias * *la*, struct sctp_nat_msg * *sm*, int *direction*) [static]

Check to see if ASCONF contains an Add IP or Del IP parameter.

IsADDorDEL scans an ASCONF packet to see if it contains an AddIP or DelIP parameter

Parameters:

- la* Pointer to the relevant libalias instance
- sm* Pointer to sctp message information
- direction* SN_TO_LOCAL | SN_TO_GLOBAL

Returns:

SCTP_ADD_IP_ADDRESS | SCTP_DEL_IP_ADDRESS | 0 - fail

Definition at line 1617 of file alias_sctp.c.

```

1629     {
1630     if (ntohs(param->param_type) == SCTP_ADD_IP_ADDRESS)
1631         return(SCTP_ADD_IP_ADDRESS);
1632     else if (ntohs(param->param_type) == SCTP_DEL_IP_ADDRESS)
1633         return(SCTP_DEL_IP_ADDRESS);
1634     /* check others just in case */
1635     bytes_left -= param_size;
1636     if (bytes_left >= SN_MIN_PARAM_SIZE) {
1637         param = SN_SCTP_NEXTPARAM(param);
1638     } else {
1639         return(0); /*Neither found */
1640     }
1641     param_size = SCTP_SIZE32(ntohs(param->param_length));
1642     if (bytes_left < param_size) return(0);
1643
1644     if (++param_count > sysctl_param_proc_limit) {
1645         SN_LOG(SN_LOG_EVENT,
1646             logsctperror("Parameter parse limit exceeded IsADDorDEL)",
1647                 sm->sctp_hdr->v_tag, sysctl_param_proc_limit, direction));
1648         return(0); /* not found limit exceeded*/
1649     }
1650 }

```

```

1651 return(0); /*Neither found */
1652 }
1653
1654 /* -----
1655 *                               STATE MACHINE CODE

```

5.2.3.4 static int IsASCONFack (struct libalias * *la*, struct sctp_nat_msg * *sm*, int *direction*) [static]

Check that ASCONF was successful.

Each ASCONF configuration parameter carries a correlation ID which should be matched with an ASCONFack. This is difficult for a NAT, since every association could potentially have a number of outstanding ASCONF configuration parameters, which should only be activated on receipt of the ACK.

Currently we only look for an ACK when the NAT is setting up a new association (ie AddIP for a connection that the NAT does not know about because the original Init went through a public interface or another NAT) Since there is currently no connection on this path, there should be no other ASCONF configuration parameters outstanding, so we presume that if there is an ACK that it is responding to the AddIP and activate the new association.

Parameters:

- la* Pointer to the relevant libalias instance
- sm* Pointer to sctp message information
- direction* SN_TO_LOCAL | SN_TO_GLOBAL

Returns:

- 1 - success | 0 - fail

Definition at line 1564 of file alias_sctp.c.

```

1578                                     {
1579     if (ntohs(param->param_type) == SCTP_SUCCESS_REPORT)
1580         return(1); /* success - but can't match correlation IDs - should only be one */
1581     /* check others just in case */
1582     bytes_left -= param_size;
1583     if (bytes_left >= SN_MIN_PARAM_SIZE) {
1584         param = SN_SCTP_NEXTPARAM(param);
1585     } else {
1586         return(0);
1587     }
1588     param_size = SctpSize32(ntohs(param->param_length));
1589     if (bytes_left < param_size) return(0);
1590
1591     if (++param_count > sysctl_param_proc_limit) {
1592         SN_LOG(SN_LOG_EVENT,
1593             logsctperror("Parameter parse limit exceeded (IsASCONFack)",
1594                 sm->sctp_hdr->v_tag, sysctl_param_proc_limit, direction));
1595         return(0); /* not found limit exceeded*/
1596     }
1597 }
1598 return(0); /* not success */
1599 }
1600

```

5.2.3.5 static void RmGlobalIPAddresses (struct sctp_nat_msg * sm, struct sctp_nat_assoc * assoc, int direction) [static]

RmGlobalIPAddresses from DelIP packets.

RmGlobalIPAddresses scans an ASCONF chunk for DelIP parameters to remove the given Global IP addresses from the association. It will not delete the the address if it is a list of one address.

Parameters:

sm Pointer to sctp message information

assoc Pointer to the association this SCTP Message belongs to

direction SN_TO_LOCAL | SN_TO_GLOBAL

Definition at line 1458 of file alias_sctp.c.

```

1473                                     {
1474     param_size = SCTP_SIZE32 (ntohs (param->param_length));
1475 } else {
1476     SN_LOG (SN_LOG_EVENT,
1477           logstpperror ("RmGlobalIPAddress: truncated packet - cannot remove IP addresses",
1478                       sm->sctp_hdr->v_tag, sysctl_track_global_addresses, direction));
1479     return;
1480 }
1481
1482 /* step through Asconf parameters */
1483 while ((bytes_left >= param_size) && (bytes_left >= sizeof (struct sctp_ipv4addr_param))) {
1484     if (ntohs (param->param_type) == SCTP_DEL_IP_ADDRESS) {
1485         asconf_ipv4_param = (struct sctp_asconf_addrv4_param *) param;
1486         if (asconf_ipv4_param->addrp.addr == INADDR_ANY) { /* remove all bar pkt address */
1487             LIST_FOREACH_SAFE (G_Addr, &(assoc->Gaddr), list_Gaddr, G_Addr_tmp) {
1488                 if (G_Addr->g_addr.s_addr != sm->ip_hdr->ip_src.s_addr) {
1489                     if (assoc->num_Gaddr > 1) { /* only delete if more than one */
1490                         LIST_REMOVE (G_Addr, list_Gaddr);
1491                         sn_free (G_Addr);
1492                         assoc->num_Gaddr--;
1493                     } else {
1494                         SN_LOG (SN_LOG_EVENT,
1495                               logstpperror ("RmGlobalIPAddress: Request to remove last IP address (didn't)",
1496                                             sm->sctp_hdr->v_tag, assoc->num_Gaddr, direction));
1497                     }
1498                 }
1499             }
1500             return; /* shouldn't be any other addresses if the zero address is given */
1501         } else {
1502             LIST_FOREACH_SAFE (G_Addr, &(assoc->Gaddr), list_Gaddr, G_Addr_tmp) {
1503                 if (G_Addr->g_addr.s_addr == asconf_ipv4_param->addrp.addr) {
1504                     if (assoc->num_Gaddr > 1) { /* only delete if more than one */
1505                         LIST_REMOVE (G_Addr, list_Gaddr);
1506                         sn_free (G_Addr);
1507                         assoc->num_Gaddr--;
1508                         break; /* Since add only adds new addresses, there should be no double entries */
1509                     } else {
1510                         SN_LOG (SN_LOG_EVENT,
1511                               logstpperror ("RmGlobalIPAddress: Request to remove last IP address (didn't)",
1512                                             sm->sctp_hdr->v_tag, assoc->num_Gaddr, direction));
1513                     }
1514                 }
1515             }
1516         }
1517     }
1518     bytes_left -= param_size;
1519     if (bytes_left == 0) return;
1520     else if (bytes_left < SN_MIN_PARAM_SIZE) {

```

```

1521     SN_LOG(SN_LOG_EVENT,
1522           logsctperror("RmGlobalIPAddress: truncated packet - may not have removed all IP addresses
1523                       sm->sctp_hdr->v_tag, sysctl_track_global_addresses, direction));
1524     return;
1525 }
1526
1527 param = SN_SCTP_NEXTPARAM(param);
1528 param_size = SCTP_SIZE32(ntohs(param->param_length));
1529 if (++param_count > sysctl_param_proc_limit) {
1530     SN_LOG(SN_LOG_EVENT,
1531           logsctperror("Parameter parse limit exceeded (RmGlobalIPAddress)",
1532                       sm->sctp_hdr->v_tag, sysctl_param_proc_limit, direction));
1533     return; /* limit exceeded*/
1534 }
1535 }
1536 }
1537

```

5.2.3.6 static int sctp_PktParser (struct libalias * *la*, int *direction*, struct ip * *pip*, struct sctp_nat_msg * *sm*, struct sctp_nat_assoc ** *passoc*) [static]

Parses SCTP packets for the key SCTP chunk that will be processed.

This module parses SCTP packets for the key SCTP chunk that will be processed. The module completes the [sctp_nat_msg](#) structure and either retrieves the relevant (existing) stored association from the Hash Tables or creates a new association entity with state SN_ID.

Parameters:

- la* Pointer to the relevant libalias instance
- direction* SN_TO_LOCAL | SN_TO_GLOBAL
- pip*
- sm* Pointer to sctp message information
- passoc* Pointer to the association this SCTP Message belongs to

Returns:

SN_PARSE_OK | SN_PARSE_ERROR_*

Definition at line 1006 of file alias_sctp.c.

References sctpChunkOfInt::Asconf, sctp_nat_msg::chunk_length, FindSctpGlobal(), FindSctpGlobalT(), FindSctpLocal(), FindSctpLocalT(), sctpChunkOfInt::Init, sctpChunkOfInt::InitAck, sctp_nat_msg::ip_hdr, sctp_nat_msg::msg, sctp_nat_msg::sctp_hdr, sctp_nat_msg::sctpchnk, SN_ID, sn_malloc, SN_MIN_CHUNK_SIZE, SN_NULL_TBL, SN_PARSE_ERROR_AS_MALLOC, SN_PARSE_ERROR_CHHL, SN_PARSE_ERROR_IPSHL, SN_PARSE_ERROR_LOOKUP, SN_PARSE_ERROR_LOOKUP_ABORT, SN_PARSE_ERROR_PARTIALLOOKUP, SN_PARSE_ERROR_PORT, SN_PARSE_ERROR_VTAG, SN_PARSE_OK, SN_SCTP_ABORT, SN_SCTP_ASCONF, SN_SCTP_ASCONFACK, SN_SCTP_FIRSTCHUNK, SN_SCTP_INIT, SN_SCTP_INITACK, SN_SCTP_NEXTCHUNK, SN_SCTP_OTHER, SN_SCTP_SHUTACK, SN_SCTP_SHUTCOMP, SN_TO_LOCAL, sysctl_chunk_proc_limit, and sysctl_initialising_chunk_proc_limit.

```

1006 {
1007     struct sctphdr *sctp_hdr;
1008     struct sctp_chunkhdr *chunk_hdr;
1009     struct sctp_paramhdr *param_hdr;
1010     struct in_addr ipv4addr;

```



```

1011 int bytes_left; /* bytes left in ip packet */
1012 int chunk_length;
1013 int chunk_count;
1014 int partial_match = 0;
1015 // mbuf *mp;
1016 // int mlen;
1017
1018 // mlen = SCTP_HEADER_LEN(i_pak);
1019 // mp = SCTP_HEADER_TO_CHAIN(i_pak); /* does nothing in bsd since header and chain not separate */
1020
1021 /*
1022  * Note, that if the VTag is zero, it must be an INIT
1023  * Also, I am only interested in the content of INIT and ADDIP chunks
1024  */
1025
1026 // no mbuf stuff from Paolo yet so ...
1027 sm->ip_hdr = pip;
1028 /* remove ip header length from the bytes_left */
1029 bytes_left = ntohs(pip->ip_len) - (pip->ip_hl << 2);
1030
1031 /* Check SCTP header length and move to first chunk */
1032 if (bytes_left < sizeof(struct sctphdr)) {
1033     sm->sctp_hdr = NULL;
1034     return(SN_PARSE_ERROR_IPSHL); /* packet not long enough*/
1035 }
1036
1037 sm->sctp_hdr = sctp_hdr = (struct sctphdr *) ip_next(pip);
1038 bytes_left -= sizeof(struct sctphdr);
1039
1040 /* Check for valid ports (zero valued ports would find partially initialised associations */
1041 if (sctp_hdr->src_port == 0 || sctp_hdr->dest_port == 0)
1042     return(SN_PARSE_ERROR_PORT);
1043
1044 /* Check length of first chunk */
1045 if (bytes_left < SN_MIN_CHUNK_SIZE) /* malformed chunk - could cause endless loop*/
1046     return(SN_PARSE_ERROR_CHHL); /* packet not long enough for this chunk */
1047
1048 /* First chunk */
1049 chunk_hdr = SN_SCTP_FIRSTCHUNK(sctp_hdr);
1050
1051 chunk_length = SCTP_SIZE32( ntohs(chunk_hdr->chunk_length));
1052 if ((chunk_length < SN_MIN_CHUNK_SIZE) || (chunk_length > bytes_left)) /* malformed chunk - could c
1053     return(SN_PARSE_ERROR_CHHL);
1054
1055 if ((chunk_hdr->chunk_flags & SCTP_HAD_NO_TCB) &&
1056     ((chunk_hdr->chunk_type == SCTP_ABORT_ASSOCIATION) ||
1057     (chunk_hdr->chunk_type == SCTP_SHUTDOWN_COMPLETE))) {
1058     /* T-Bit set */
1059     if (direction == SN_TO_LOCAL)
1060         *passoc = FindSctpGlobalT(la, pip->ip_src, sctp_hdr->v_tag, sctp_hdr->dest_port, sctp_hdr->src_p
1061     else
1062         *passoc = FindSctpLocalT(la, pip->ip_dst, sctp_hdr->v_tag, sctp_hdr->dest_port, sctp_hdr->src_p
1063 } else {
1064     /* Proper v_tag settings */
1065     if (direction == SN_TO_LOCAL)
1066         *passoc = FindSctpGlobal(la, pip->ip_src, sctp_hdr->v_tag, sctp_hdr->src_port, sctp_hdr->dest_p
1067     else
1068         *passoc = FindSctpLocal(la, pip->ip_src, pip->ip_dst, sctp_hdr->v_tag, sctp_hdr->src_port, sct
1069 }
1070
1071 chunk_count = 1;
1072 /* Real packet parsing occurs below */
1073 sm->msg = SN_SCTP_OTHER; /* Initialise to largest value*/
1074 sm->chunk_length = 0; /* only care about length for key chunks */
1075 while (IS_SCTP_CONTROL(chunk_hdr)) {
1076     switch(chunk_hdr->chunk_type) {
1077     case SCTP_INITIATION:

```

```

1078     if (chunk_length < sizeof(struct sctp_init_chunk)) /* malformed chunk*/
1079         return(SN_PARSE_ERROR_CHHL);
1080     sm->msg = SN_SCTP_INIT;
1081     sm->sctpchnk.Init = (struct sctp_init *) ((char *) chunk_hdr + sizeof(struct sctp_chunkhdr));
1082     sm->chunk_length = chunk_length;
1083     /* if no existing association, create a new one */
1084     if (*passoc == NULL) {
1085         if (sctp_hdr->v_tag == 0){ //Init requires vtag=0
1086             *passoc = (struct sctp_nat_assoc *) sn_malloc(sizeof(struct sctp_nat_assoc));
1087             if (*passoc == NULL) /* out of resources */
1088                 return(SN_PARSE_ERROR_AS_MALLOC);
1089         }
1090         /* Initialise association - malloc initialises memory to zeros */
1091         (*passoc)->state = SN_ID;
1092         LIST_INIT(&((*passoc)->Gaddr)); /* always initialise to avoid memory problems */
1093         (*passoc)->TableRegister = SN_NULL_TBL;
1094         return(SN_PARSE_OK);
1095     }
1096     return(SN_PARSE_ERROR_VTAG);
1097 }
1098 return(SN_PARSE_ERROR_LOOKUP);
1099 case SCTP_INITIATION_ACK:
1100     if (chunk_length < sizeof(struct sctp_init_ack_chunk)) /* malformed chunk*/
1101         return(SN_PARSE_ERROR_CHHL);
1102     sm->msg = SN_SCTP_INITACK;
1103     sm->sctpchnk.InitAck = (struct sctp_init_ack *) ((char *) chunk_hdr + sizeof(struct sctp_chunkhdr));
1104     sm->chunk_length = chunk_length;
1105     return ((*passoc == NULL)?(SN_PARSE_ERROR_LOOKUP):(SN_PARSE_OK));
1106 case SCTP_ABORT_ASSOCIATION: /* access only minimum sized chunk */
1107     sm->msg = SN_SCTP_ABORT;
1108     sm->chunk_length = chunk_length;
1109     return ((*passoc == NULL)?(SN_PARSE_ERROR_LOOKUP_ABORT):(SN_PARSE_OK));
1110 case SCTP_SHUTDOWN_ACK:
1111     if (chunk_length < sizeof(struct sctp_shutdown_ack_chunk)) /* malformed chunk*/
1112         return(SN_PARSE_ERROR_CHHL);
1113     if (sm->msg > SN_SCTP_SHUTACK) {
1114         sm->msg = SN_SCTP_SHUTACK;
1115         sm->chunk_length = chunk_length;
1116     }
1117     break;
1118 case SCTP_SHUTDOWN_COMPLETE: /* minimum sized chunk */
1119     if (sm->msg > SN_SCTP_SHUTCOMP) {
1120         sm->msg = SN_SCTP_SHUTCOMP;
1121         sm->chunk_length = chunk_length;
1122     }
1123     return ((*passoc == NULL)?(SN_PARSE_ERROR_LOOKUP):(SN_PARSE_OK));
1124 case SCTP_ASCONF:
1125     if (sm->msg > SN_SCTP_ASCONF) {
1126         if (chunk_length < (sizeof(struct sctp_asconf_chunk) + sizeof(struct sctp_ipv4addr_param)))
1127             return(SN_PARSE_ERROR_CHHL);
1128         //leave parameter searching to later, if required
1129         param_hdr = (struct sctp_paramhdr *) ((char *) chunk_hdr + sizeof(struct sctp_asconf_chunk));
1130         if (ntohs(param_hdr->param_type) == SCTP_IPV4_ADDRESS) {
1131             if ((*passoc == NULL) && (direction == SN_TO_LOCAL)) { /* AddIP with no association */
1132                 /* try look up with the ASCONF packet's alternative address */
1133                 ipv4addr.s_addr = ((struct sctp_ipv4addr_param *) param_hdr)->addr;
1134                 *passoc = FindSctpGlobal(la, ipv4addr, sctp_hdr->v_tag, sctp_hdr->src_port, sctp_hdr->des);
1135             }
1136             param_hdr = (struct sctp_paramhdr *)
1137                 ((char *) param_hdr + sizeof(struct sctp_ipv4addr_param)); /*asconf's compulsory address
1138             sm->chunk_length = chunk_length - sizeof(struct sctp_asconf_chunk) - sizeof(struct sctp_i
1139         } else {
1140             if (chunk_length < (sizeof(struct sctp_asconf_chunk) + sizeof(struct sctp_ipv6addr_param))
1141                 return(SN_PARSE_ERROR_CHHL);
1142             param_hdr = (struct sctp_paramhdr *)
1143                 ((char *) param_hdr + sizeof(struct sctp_ipv6addr_param)); /*asconf's compulsory address
1144             sm->chunk_length = chunk_length - sizeof(struct sctp_asconf_chunk) - sizeof(struct sctp_i

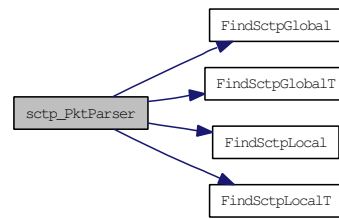
```

```

1145     }
1146     sm->msg = SN_SCTP_ASCONF;
1147     sm->sctpchnk.Asconf = param_hdr;
1148
1149     if (*passoc == NULL) { /* AddIP with no association */
1150         *passoc = (struct sctp_nat_assoc *) sn_malloc(sizeof(struct sctp_nat_assoc));
1151         if (*passoc == NULL) { /* out of resources */
1152             return(SN_PARSE_ERROR_AS_MALLOC);
1153         }
1154         /* Initialise association - malloc initialises memory to zeros */
1155         (*passoc)->state = SN_ID;
1156         LIST_INIT(&(*passoc)->Gaddr); /* always initialise to avoid memory problems */
1157         (*passoc)->TableRegister = SN_NULL_TBL;
1158         return(SN_PARSE_OK);
1159     }
1160 }
1161 break;
1162 case SCTP_ASCONF_ACK:
1163     if (sm->msg > SN_SCTP_ASCONFACK) {
1164         if (chunk_length < sizeof(struct sctp_asconf_ack_chunk)) /* malformed chunk*/
1165             return(SN_PARSE_ERROR_CHHL);
1166         //leave parameter searching to later, if required
1167         param_hdr = (struct sctp_paramhdr *) ((char *) chunk_hdr
1168                                             + sizeof(struct sctp_asconf_ack_chunk));
1169         sm->msg = SN_SCTP_ASCONFACK;
1170         sm->sctpchnk.Asconf = param_hdr;
1171         sm->chunk_length = chunk_length - sizeof(struct sctp_asconf_ack_chunk);
1172     }
1173     break;
1174 default:
1175     break; /* do nothing*/
1176 }
1177
1178 /* if no association is found exit - we need to find an Init or AddIP within sysctl_initialising
1179 if ((*passoc == NULL) && (chunk_count >= sysctl_initialising_chunk_proc_limit))
1180     return(SN_PARSE_ERROR_LOOKUP);
1181
1182 /* finished with this chunk, on to the next chunk*/
1183 bytes_left -= chunk_length;
1184
1185 /* Is this the end of the packet ? */
1186 if (bytes_left == 0)
1187     return(SN_PARSE_OK);
1188
1189 /* Are there enough bytes in packet to at least retrieve length of next chunk ? */
1190 if (bytes_left < SN_MIN_CHUNK_SIZE)
1191     return(SN_PARSE_ERROR_CHHL);
1192
1193 chunk_hdr = SN_SCTP_NEXTCHUNK(chunk_hdr);
1194
1195 /* Is the chunk long enough to not cause endless look and are there enough bytes in packet to read
1196 chunk_length = SCTP_SIZE32(ntohs(chunk_hdr->chunk_length));
1197 if ((chunk_length < SN_MIN_CHUNK_SIZE) || (chunk_length > bytes_left))
1198     return(SN_PARSE_ERROR_CHHL);
1199 if(++chunk_count > sysctl_chunk_proc_limit)
1200     return(SN_PARSE_OK); /* limit for processing chunks, take what we get */
1201 }
1202
1203 if (*passoc == NULL)
1204     return (partial_match) ? (SN_PARSE_ERROR_PARTIALLOOKUP) : (SN_PARSE_ERROR_LOOKUP);
1205 else
1206     return(SN_PARSE_OK);
1207 }
1208

```

Here is the call graph for this function:



5.3 SCTP NAT State Machine

Defines

- #define `SN_ID` 0x0000
- #define `SN_INi` 0x0010
- #define `SN_INa` 0x0020
- #define `SN_UP` 0x0100
- #define `SN_CL` 0x1000
- #define `SN_RM` 0x2000

Functions

- static int `ProcessSctpMsg` (struct libalias *la, int direction, struct `sctp_nat_msg` *sm, struct `sctp_nat_assoc` *assoc)
Process SCTP message.
- static int `ID_process` (struct libalias *la, int direction, struct `sctp_nat_assoc` *assoc, struct `sctp_nat_msg` *sm)
Process SCTP message while in the Idle state.
- static int `INi_process` (struct libalias *la, int direction, struct `sctp_nat_assoc` *assoc, struct `sctp_nat_msg` *sm)
Process SCTP message while waiting for an INIT-ACK message.
- static int `INa_process` (struct libalias *la, int direction, struct `sctp_nat_assoc` *assoc, struct `sctp_nat_msg` *sm)
Process SCTP message while waiting for an AddIp-ACK message.
- static int `UP_process` (struct libalias *la, int direction, struct `sctp_nat_assoc` *assoc, struct `sctp_nat_msg` *sm)
Process SCTP messages while association is UP redirecting packets.
- static int `CL_process` (struct libalias *la, int direction, struct `sctp_nat_assoc` *assoc, struct `sctp_nat_msg` *sm)
Process SCTP message while association is in the process of closing.

5.3.1 Detailed Description

Defines the various states an association can be within the NAT

The SCTP NAT State Machine functions will:

- Process an already parsed packet
- Use the existing NAT Hash Tables
- Determine the next state for the association
- Update the NAT Hash Tables and Timer Queues
- Return the appropriate action to take with the packet

5.3.2 Define Documentation

5.3.2.1 #define SN_CL 0x1000

Closing state

Definition at line 279 of file alias_sctp.c.

Referenced by logsctpassoc(), and ProcessSctpMsg().

5.3.2.2 #define SN_ID 0x0000

Idle state

Definition at line 275 of file alias_sctp.c.

Referenced by logsctpassoc(), ProcessSctpMsg(), and sctp_PktParser().

5.3.2.3 #define SN_INa 0x0020

Initialising, waiting for AddIpAck state

Definition at line 277 of file alias_sctp.c.

Referenced by logsctpassoc(), and ProcessSctpMsg().

5.3.2.4 #define SN_INi 0x0010

Initialising, waiting for InitAck state

Definition at line 276 of file alias_sctp.c.

Referenced by logsctpassoc(), and ProcessSctpMsg().

5.3.2.5 #define SN_RM 0x2000

Removing state

Definition at line 280 of file alias_sctp.c.

Referenced by logsctpassoc(), and ProcessSctpMsg().

5.3.2.6 #define SN_UP 0x0100

Association in UP state

Definition at line 278 of file alias_sctp.c.

Referenced by logsctpassoc(), and ProcessSctpMsg().

5.3.3 Function Documentation

5.3.3.1 static int CL_process (struct libalias * *la*, int *direction*, struct sctp_nat_assoc * *assoc*, struct sctp_nat_msg * *sm*) [static]

Process SCTP message while association is in the process of closing.

This function waits for a SHUT-COMP to close the association. Depending on the the setting of `sysctl_holddown_timer` it may not remove the association immediately, but leave it up until `SN_X_T(la)`. Only SHUT-COMP, SHUT-ACK, and ABORT packets are permitted in this state. All other packets are dropped.

Parameters:

la Pointer to the relevant libalias instance

direction `SN_TO_LOCAL` | `SN_TO_GLOBAL`

sm Pointer to sctp message information

assoc Pointer to the association this Sctp Message belongs to

Returns:

`SN_NAT_PKT` | `SN_DROP_PKT`

Definition at line 1933 of file `alias_sctp.c`.

Referenced by `ProcessSctpMsg()`.

```

1933         : /* a packet containing a SHUTDOWN-COMPLETE chunk */
1934     assoc->state = SN_CL; /* Stay in Close state until timeout */
1935     if (sysctl_holddown_timer > 0)
1936         sctp_ResetTimeout(la, assoc, SN_X_T(la)); /* allow to stay open for Tbit packets*/
1937     else
1938         assoc->state = SN_RM; /* Mark for removal*/
1939     return(SN_NAT_PKT);
1940 case SN_SCTP_SHUTACK: /* a packet containing a SHUTDOWN-ACK chunk */
1941     assoc->state = SN_CL; /* Stay in Close state until timeout */
1942     sctp_ResetTimeout(la, assoc, SN_C_T(la));
1943     return(SN_NAT_PKT);
1944 case SN_SCTP_ABORT: /* a packet containing an ABORT chunk */
1945     assoc->state = SN_RM; /* Mark for removal*/
1946     return(SN_NAT_PKT);
1947 default:
1948     return(SN_DROP_PKT);
1949 }
1950 return(SN_DROP_PKT); /* shouldn't get here very bad: log, drop and hope for the best */
1951 }
1952
1953 /* -----
1954 *

```

5.3.3.2 static int ID_process (struct libalias * la, int direction, struct sctp_nat_assoc * assoc, struct sctp_nat_msg * sm) [static]

Process Sctp message while in the Idle state.

This function looks for an Incoming INIT or AddIP message.

All other Sctp messages are invalid when in `SN_ID`, and are dropped.

Parameters:

la Pointer to the relevant libalias instance

direction `SN_TO_LOCAL` | `SN_TO_GLOBAL`

sm Pointer to sctp message information

assoc Pointer to the association this Sctp Message belongs to

Returns:

SN_NAT_PKT | SN_DROP_PKT | SN_REPLY_ABORT | SN_REPLY_ERROR

Definition at line 1722 of file alias_sctp.c.

Referenced by ProcessSctpMsg().

```

1722             : /* a packet containing an ASCONF chunk with ADDIP */
1723     if (!sysctl_accept_global_ootb_addip && (direction == SN_TO_LOCAL))
1724         return(SN_DROP_PKT);
1725     /* if this Asconf packet does not contain the Vtag parameters it is of no use in Idle state */
1726     if (!GetAsconfVtags(la, sm, &(assoc->l_vtag), &(assoc->g_vtag), direction))
1727         return(SN_DROP_PKT);
1728     case SN_SCTP_INIT: /* a packet containing an INIT chunk or an ASCONF AddIP */
1729     if (sysctl_track_global_addresses)
1730         AddGlobalIPAddresses(sm, assoc, direction);
1731     switch(direction){
1732     case SN_TO_GLOBAL:
1733         assoc->l_addr = sm->ip_hdr->ip_src;
1734         assoc->a_addr = FindAliasAddress(la, assoc->l_addr);
1735         assoc->l_port = sm->sctp_hdr->src_port;
1736         assoc->g_port = sm->sctp_hdr->dest_port;
1737         if(sm->msg == SN_SCTP_INIT)
1738             assoc->g_vtag = sm->sctpchnk.Init->initiate_tag;
1739         if (AddSctpAssocGlobal(la, assoc)) /* DB clash *///**** need to add dst address
1740             return((sm->msg == SN_SCTP_INIT) ? SN_REPLY_ABORT : SN_REPLY_ERROR);
1741         if(sm->msg == SN_SCTP_ASCONF) {
1742             if (AddSctpAssocLocal(la, assoc, sm->ip_hdr->ip_dst)) /* DB clash */
1743                 return(SN_REPLY_ERROR);
1744             assoc->TableRegister |= SN_WAIT_TOLOCAL; /* wait for tolocal ack */
1745         }
1746         break;
1747     case SN_TO_LOCAL:
1748         assoc->l_addr = FindSctpRedirectAddress(la, sm);
1749         assoc->a_addr = sm->ip_hdr->ip_dst;
1750         assoc->l_port = sm->sctp_hdr->dest_port;
1751         assoc->g_port = sm->sctp_hdr->src_port;
1752         if(sm->msg == SN_SCTP_INIT)
1753             assoc->l_vtag = sm->sctpchnk.Init->initiate_tag;
1754         if (AddSctpAssocLocal(la, assoc, sm->ip_hdr->ip_src)) /* DB clash */
1755             return((sm->msg == SN_SCTP_INIT) ? SN_REPLY_ABORT : SN_REPLY_ERROR);
1756         if(sm->msg == SN_SCTP_ASCONF) {
1757             if (AddSctpAssocGlobal(la, assoc)) /* DB clash */ //**** need to add src address
1758                 return(SN_REPLY_ERROR);
1759             assoc->TableRegister |= SN_WAIT_TOGLOBAL; /* wait for toglobal ack */
1760         }
1761         break;
1762     }
1763     assoc->state = (sm->msg == SN_SCTP_INIT) ? SN_INi : SN_INa;
1764     assoc->exp = SN_I_T(la);
1765     sctp_AddTimeOut(la,assoc);
1766     return(SN_NAT_PKT);
1767     default: /* Any other type of SCTP message is not valid in Idle */
1768         return(SN_DROP_PKT);
1769     }
1770     return(SN_DROP_PKT);/* shouldn't get here very bad: log, drop and hope for the best */
1771 }
1772

```

5.3.3.3 static int INa_process (struct libalias * la, int direction, struct sctp_nat_assoc * assoc, struct sctp_nat_msg * sm) [static]

Process SCTP message while waiting for an AddIp-ACK message.

Only an AddIP-ACK, resent AddIP, or an ABORT message are valid, all other SCTP packets are dropped

Parameters:

la Pointer to the relevant libalias instance

direction SN_TO_LOCAL | SN_TO_GLOBAL

sm Pointer to sctp message information

assoc Pointer to the association this SCTP Message belongs to

Returns:

SN_NAT_PKT | SN_DROP_PKT

Definition at line 1842 of file alias_sctp.c.

Referenced by ProcessSctpMsg().

```

1842             :                /* a packet containing an ASCONF chunk*/
1843     sctp_ResetTimeout(la,assoc, SN_I_T(la));
1844     return(SN_NAT_PKT);
1845     case SN_SCTP_ASCONFACK:      /* a packet containing an ASCONF chunk with a ADDIP-ACK */
1846     switch(direction){
1847     case SN_TO_LOCAL:
1848         if (!(assoc->TableRegister & SN_WAIT_TOLOCAL)) /* wrong direction */
1849             return(SN_DROP_PKT);
1850         break;
1851     case SN_TO_GLOBAL:
1852         if (!(assoc->TableRegister & SN_WAIT_TOGLOBAL)) /* wrong direction */
1853             return(SN_DROP_PKT);
1854     }
1855     if (IsASCONFack(la,sm,direction)) {
1856         assoc->TableRegister &= SN_BOTH_TBL; /* remove wait flags */
1857         assoc->state = SN_UP; /* association established for NAT */
1858         sctp_ResetTimeout(la,assoc, SN_U_T(la));
1859         return(SN_NAT_PKT);
1860     } else {
1861         assoc->state = SN_RM; /* Mark for removal*/
1862         return(SN_NAT_PKT);
1863     }
1864     case SN_SCTP_ABORT:         /* a packet containing an ABORT chunk */
1865         assoc->state = SN_RM; /* Mark for removal*/
1866         return(SN_NAT_PKT);
1867     default:
1868         return(SN_DROP_PKT);
1869     }
1870     return(SN_DROP_PKT); /* shouldn't get here very bad: log, drop and hope for the best */
1871 }
1872

```

5.3.3.4 static int INi_process (struct libalias * *la*, int *direction*, struct sctp_nat_assoc * *assoc*, struct sctp_nat_msg * *sm*) [static]

Process SCTP message while waiting for an INIT-ACK message.

Only an INIT-ACK, resent INIT, or an ABORT SCTP packet are valid in this state, all other packets are dropped.

Parameters:

la Pointer to the relevant libalias instance

direction SN_TO_LOCAL | SN_TO_GLOBAL

sm Pointer to sctp message information

assoc Pointer to the association this Sctp Message belongs to

Returns:

SN_NAT_PKT | SN_DROP_PKT | SN_REPLY_ABORT

Definition at line 1790 of file alias_sctp.c.

Referenced by ProcessSctpMsg().

```

1790             : /* a packet containing a retransmitted INIT chunk */
1791     sctp_ResetTimeOut(la, assoc, SN_I_T(la));
1792     return(SN_NAT_PKT);
1793 case SN_SCTP_INITACK: /* a packet containing an INIT-ACK chunk */
1794     switch(direction){
1795     case SN_TO_LOCAL:
1796         if (assoc->num_Gaddr) /*If tracking global addresses for this association */
1797             AddGlobalIPAddresses(sm, assoc, direction);
1798         assoc->l_vtag = sm->sctpchnk.Init->initiate_tag;
1799         if (AddSctpAssocLocal(la, assoc, sm->ip_hdr->ip_src)) { /* DB clash */
1800             assoc->state = SN_RM; /* Mark for removal*/
1801             return(SN_SEND_ABORT);
1802         }
1803         break;
1804     case SN_TO_GLOBAL:
1805         assoc->l_addr = sm->ip_hdr->ip_src; // Only if not set in Init! *
1806         assoc->g_vtag = sm->sctpchnk.Init->initiate_tag;
1807         if (AddSctpAssocGlobal(la, assoc)) { /* DB clash */
1808             assoc->state = SN_RM; /* Mark for removal*/
1809             return(SN_SEND_ABORT);
1810         }
1811         break;
1812     }
1813     assoc->state = SN_UP; /* association established for NAT */
1814     sctp_ResetTimeOut(la, assoc, SN_U_T(la));
1815     return(SN_NAT_PKT);
1816 case SN_SCTP_ABORT: /* a packet containing an ABORT chunk */
1817     assoc->state = SN_RM; /* Mark for removal*/
1818     return(SN_NAT_PKT);
1819 default:
1820     return(SN_DROP_PKT);
1821 }
1822 return(SN_DROP_PKT); /* shouldn't get here very bad: log, drop and hope for the best */
1823 }
1824

```

5.3.3.5 static int ProcessSctpMsg (struct libalias * *la*, int *direction*, struct sctp_nat_msg * *sm*, struct sctp_nat_assoc * *assoc*) [static]

Process Sctp message.

This function is the base state machine. It calls the processing engine for each state.

Parameters:

la Pointer to the relevant libalias instance

direction SN_TO_LOCAL | SN_TO_GLOBAL

sm Pointer to sctp message information

assoc Pointer to the association this SCTP Message belongs to

Returns:

SN_DROP_PKT | SN_NAT_PKT | SN_REPLY_ABORT | SN_REPLY_ERROR | SN_PROCESSING_ERROR

Definition at line 1684 of file alias_sctp.c.

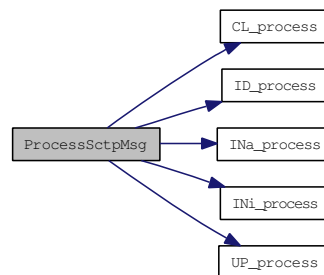
References CL_process(), ID_process(), INa_process(), INi_process(), SN_CL, SN_ID, SN_INa, SN_INi, SN_NAT_PKT, SN_RM, SN_UP, sctp_nat_assoc::state, and UP_process().

```

1685     {
1686     case SN_ID: /* Idle */
1687         rtnval = ID_process(la, direction, assoc, sm);
1688         if (rtnval != SN_NAT_PKT) {
1689             assoc->state = SN_RM; /* Mark for removal*/
1690         }
1691         return(rtnval);
1692     case SN_INi: /* Initialising - Init */
1693         return(INi_process(la, direction, assoc, sm));
1694     case SN_INa: /* Initialising - AddIP */
1695         return(INa_process(la, direction, assoc, sm));
1696     case SN_UP: /* Association UP */
1697         return(UP_process(la, direction, assoc, sm));
1698     case SN_CL: /* Association Closing */
1699         return(CL_process(la, direction, assoc, sm));
1700     }
1701     return(SN_PROCESSING_ERROR);
1702 }
1703

```

Here is the call graph for this function:



5.3.3.6 static int UP_process (struct libalias * la, int direction, struct sctp_nat_assoc * assoc, struct sctp_nat_msg * sm) [static]

Process SCTP messages while association is UP redirecting packets.

While in the SN_UP state, all packets for the particular association are passed. Only a SHUT-ACK or an ABORT will cause a change of state.

Parameters:

la Pointer to the relevant libalias instance

direction SN_TO_LOCAL | SN_TO_GLOBAL

sm Pointer to sctp message information

assoc Pointer to the association this Sctp Message belongs to

Returns:

SN_NAT_PKT | SN_DROP_PKT

Definition at line 1890 of file alias_sctp.c.

Referenced by ProcessSctpMsg().

```
1890             : /* a packet containing a SHUTDOWN-ACK chunk */
1891     assoc->state = SN_CL;
1892     sctp_ResetTimeout(la,assoc, SN_C_T(la));
1893     return(SN_NAT_PKT);
1894 case SN_SCTP_ABORT: /* a packet containing an ABORT chunk */
1895     assoc->state = SN_RM; /* Mark for removal*/
1896     return(SN_NAT_PKT);
1897 case SN_SCTP_ASCONF: /* a packet containing an ASCONF chunk*/
1898     if ((direction == SN_TO_LOCAL) && assoc->num_Gaddr) /*If tracking global addresses for this assoc
1899     switch(IsADDorDEL(la,sm,direction)) {
1900     case SCTP_ADD_IP_ADDRESS:
1901         AddGlobalIPAddresses(sm, assoc, direction);
1902         break;
1903     case SCTP_DEL_IP_ADDRESS:
1904         RmGlobalIPAddresses(sm, assoc, direction);
1905         break;
1906     } /* fall through to default */
1907 default:
1908     sctp_ResetTimeout(la,assoc, SN_U_T(la));
1909     return(SN_NAT_PKT); /* forward packet */
1910 }
1911 return(SN_DROP_PKT); /* shouldn't get here very bad: log, drop and hope for the best */
1912 }
1913 }
```

5.4 Logging Functionality

Defines

- #define `SN_LOG_LOW` 0
- #define `SN_LOG_EVENT` 1
- #define `SN_LOG_INFO` 2
- #define `SN_LOG_DETAIL` 3
- #define `SN_LOG_DEBUG` 4
- #define `SN_LOG_DEBUG_MAX` 5
- #define `SN_LOG(level, action) if (sysctl_log_level >= level) { action; }`

Functions

- static void `logsctpperror` (char *errmsg, uint32_t vtag, int error, int direction)
Log sctp nat errors.
- static void `logsctpparse` (int direction, struct `sctp_nat_msg` *sm)
Log what the parser parsed.
- static void `logsctpassoc` (struct `sctp_nat_assoc` *assoc, char *s)
Log an SCTP association's details.
- static void `logSctpGlobal` (struct `libalias` *la)
Output Global table to log.
- static void `logSctpLocal` (struct `libalias` *la)
Output Local table to log.
- static void `logTimerQ` (struct `libalias` *la)
Output timer queue to log.
- static void `SctpAliasLog` (FILE *stream, const char *format,...)
Sctp NAT logging function.

5.4.1 Detailed Description

Define various log levels and a macro to call specified log functions only if the current log level (`sysctl_log_level`) matches the specified level

The logging functions provide logging of different items ranging from logging a simple message, through logging an association details to logging the current state of the NAT tables

5.4.2 Define Documentation

5.4.2.1 #define `SN_LOG(level, action) if (sysctl_log_level >= level) { action; }`

Perform log action ONLY if the current log level meets the specified log level

Definition at line 295 of file alias_sctp.c.

Referenced by RmSctpAssoc().

5.4.2.2 #define SN_LOG_DEBUG 4

Definition at line 292 of file alias_sctp.c.

5.4.2.3 #define SN_LOG_DEBUG_MAX 5

Definition at line 293 of file alias_sctp.c.

5.4.2.4 #define SN_LOG_DETAIL 3

Definition at line 291 of file alias_sctp.c.

5.4.2.5 #define SN_LOG_EVENT 1

Definition at line 289 of file alias_sctp.c.

5.4.2.6 #define SN_LOG_INFO 2

Definition at line 290 of file alias_sctp.c.

5.4.2.7 #define SN_LOG_LOW 0

Definition at line 288 of file alias_sctp.c.

Referenced by RmSctpAssoc().

5.4.3 Function Documentation

5.4.3.1 static void logsetpassoc (struct sctp_nat_assoc * assoc, char * s) [static]

Log an SCTP association's details.

Parameters:

assoc pointer to sctp association

s Character that indicates the state of processing for this packet

Definition at line 2562 of file alias_sctp.c.

References SN_CL, SN_ID, SN_INa, SN_INi, SN_RM, and SN_UP.

```

2563                                     {
2564     case SN_ID:
2565         sp = "ID ";
2566         break;
2567     case SN_INi:
2568         sp = "INi ";

```

```

2569     break;
2570 case SN_INa:
2571     sp = "INa ";
2572     break;
2573 case SN_UP:
2574     sp = "UP ";
2575     break;
2576 case SN_CL:
2577     sp = "CL ";
2578     break;
2579 case SN_RM:
2580     sp = "RM ";
2581     break;
2582 default:
2583     sp = "***ERROR***";
2584     break;
2585 }
2586 SctpAliasLog("%sAssoc: %s exp=%u la=%s lv=%u lp=%u gv=%u gp=%u tbl=%d\n",
2587             s, sp, assoc->exp, inet_ntoa(assoc->l_addr), ntohl(assoc->l_vtag),
2588             ntohs(assoc->l_port), ntohl(assoc->g_vtag), ntohs(assoc->g_port),
2589             assoc->TableRegister);
2590 /* list global addresses */
2591 LIST_FOREACH(G_Addr, &(assoc->Gaddr), list_Gaddr) {
2592     SctpAliasLog("\t\ttga=%s\n", inet_ntoa(G_Addr->g_addr));
2593 }
2594 }
2595

```

5.4.3.2 static void logsetperror (char * *errmsg*, uint32_t *vtag*, int *error*, int *direction*) [static]

Log sctp nat errors.

Parameters:

- errmsg*** Error message to be logged
- vtag*** Current Vtag
- error*** Error number
- direction*** Direction of packet

Definition at line 2487 of file alias_sctp.c.

References SN_TO_GLOBAL, and SN_TO_LOCAL.

Referenced by RmSctpAssoc().

```

2487     {
2488 case SN_TO_LOCAL:
2489     dir = 'L';
2490     break;
2491 case SN_TO_GLOBAL:
2492     dir = 'G';
2493     break;
2494 default:
2495     dir = '*';
2496     break;
2497 }
2498 SctpAliasLog("->%c %s (vt=%u) %d\n", dir, errmsg, ntohl(vtag), error);
2499 }
2500

```

5.4.3.3 static void logSctpGlobal (struct libalias * *la*) [static]

Output Global table to log.

Parameters:

la Pointer to the relevant libalias instance

Definition at line 2604 of file alias_sctp.c.

```

2607                                     {
2608     LIST_FOREACH(assoc, &la->sctpTableGlobal[i], list_G) {
2609         logsctpassoc(assoc, " ");
2610     }
2611 }
2612 }
2613

```

5.4.3.4 static void logSctpLocal (struct libalias * *la*) [static]

Output Local table to log.

Parameters:

la Pointer to the relevant libalias instance

Definition at line 2622 of file alias_sctp.c.

```

2625                                     {
2626     LIST_FOREACH(assoc, &la->sctpTableLocal[i], list_L) {
2627         logsctpassoc(assoc, " ");
2628     }
2629 }
2630 }
2631

```

5.4.3.5 static void logsctpparse (int *direction*, struct sctp_nat_msg * *sm*) [static]

Log what the parser parsed.

Parameters:

direction Direction of packet

sm Pointer to sctp message information

Definition at line 2511 of file alias_sctp.c.

References SN_TO_GLOBAL, and SN_TO_LOCAL.

```

2511                                     {
2512     case SN_TO_LOCAL:
2513         ploc = "TO_LOCAL -";
2514         break;
2515     case SN_TO_GLOBAL:
2516         ploc = "TO_GLOBAL -";

```



```

2517     break;
2518 default:
2519     ploc = "";
2520 }
2521 switch(sm->msg) {
2522 case SN_SCTP_INIT:
2523     pstate = "Init";
2524     break;
2525 case SN_SCTP_INITACK:
2526     pstate = "InitAck";
2527     break;
2528 case SN_SCTP_ABORT:
2529     pstate = "Abort";
2530     break;
2531 case SN_SCTP_SHUTACK:
2532     pstate = "ShutAck";
2533     break;
2534 case SN_SCTP_SHUTCOMP:
2535     pstate = "ShutComp";
2536     break;
2537 case SN_SCTP_ASCONF:
2538     pstate = "Asconf";
2539     break;
2540 case SN_SCTP_ASCONFACK:
2541     pstate = "AsconfAck";
2542     break;
2543 case SN_SCTP_OTHER:
2544     pstate = "Other";
2545     break;
2546 default:
2547     pstate = "***ERROR***";
2548     break;
2549 }
2550 SctpAliasLog("Parsed: %s %s\n", ploc, pstate);
2551 }
2552

```

5.4.3.6 static void logTimerQ (struct libalias *la) [static]

Output timer queue to log.

Parameters:

la Pointer to the relevant libalias instance

Definition at line 2640 of file alias_sctp.c.

```

2644     {
2645     LIST_FOREACH(assoc, &la->sctpNatTimer.TimerQ[i], timer_Q) {
2646         snprintf(buf, 50, " l=%u ", i);
2647         //SctpAliasLog(la->logDesc, " l=%d ", i);
2648         logsockpassoc(assoc, buf);
2649     }
2650 }
2651 }
2652

```

5.4.3.7 static void SctpAliasLog (FILE *stream, const char *format, ...) [static]

Sctp NAT logging function.

This function is based on a similar function in alias_db.c

Parameters:

str/stream logging descriptor

format printf type string

Definition at line 2678 of file alias_sctp.c.

5.5 Hash Table Macros and Functions

Defines

- #define `SN_MIN_HASH_SIZE` 101
- #define `SN_MAX_HASH_SIZE` 1000001
- #define `SN_DEFAULT_HASH_SIZE` 2003
- #define `SN_LOCAL_TBL` 0x01
- #define `SN_GLOBAL_TBL` 0x02
- #define `SN_BOTH_TBL` 0x03
- #define `SN_WAIT_TOLOCAL` 0x10
- #define `SN_WAIT_TOGLOBAL` 0x20
- #define `SN_NULL_TBL` 0x00
- #define `SN_MAX_GLOBAL_ADDRESSES` 100
- #define `SN_ADD_OK` 0
- #define `SN_ADD_CLASH` 1
- #define `SN_TABLE_HASH`(vtag, port, size) (((u_int) vtag + (u_int) port) % (u_int) size)

Functions

- static struct `sctp_nat_assoc` * `FindSctpLocal` (struct libalias *la, struct in_addr l_addr, struct in_addr g_addr, uint32_t l_vtag, uint16_t l_port, uint16_t g_port)
Find the SCTP association given the local address, port and vtag.
- static struct `sctp_nat_assoc` * `FindSctpGlobalClash` (struct libalias *la, struct `sctp_nat_assoc` *Cassoc)
Check for Global Clash.
- static struct `sctp_nat_assoc` * `FindSctpGlobal` (struct libalias *la, struct in_addr g_addr, uint32_t g_vtag, uint16_t g_port, uint16_t l_port, int *partial_match)
Find the SCTP association given the global port and vtag.
- static struct `sctp_nat_assoc` * `FindSctpLocalT` (struct libalias *la, struct in_addr g_addr, uint32_t l_vtag, uint16_t g_port, uint16_t l_port)
Find the SCTP association for a T-Flag message (given the global port and local vtag).
- static struct `sctp_nat_assoc` * `FindSctpGlobalT` (struct libalias *la, struct in_addr g_addr, uint32_t g_vtag, uint16_t l_port, uint16_t g_port)
Find the SCTP association for a T-Flag message (given the local port and global vtag).
- static int `AddSctpAssocLocal` (struct libalias *la, struct `sctp_nat_assoc` *assoc, struct in_addr g_addr)
Add the sctp association information to the local look up table.
- static int `AddSctpAssocGlobal` (struct libalias *la, struct `sctp_nat_assoc` *assoc)
Add the sctp association information to the global look up table.
- static void `RmSctpAssoc` (struct libalias *la, struct `sctp_nat_assoc` *assoc)
Remove the sctp association information from the look up table.

- static void `freeGlobalAddressList` (struct `sctp_nat_assoc` *assoc)
free the Global Address List memory

5.5.1 Detailed Description

Defines minimum/maximum/default values for the hash table size

The Hash functions facilitate searching the NAT Hash Tables for associations as well as adding/removing associations from the table(s).

5.5.2 Define Documentation

5.5.2.1 #define SN_ADD_CLASH 1

Clash when trying to add the assoc. info to the table

Definition at line 315 of file `alias_sctp.c`.

5.5.2.2 #define SN_ADD_OK 0

Association added to the table

Definition at line 314 of file `alias_sctp.c`.

5.5.2.3 #define SN_BOTH_TBL 0x03

assoc in both tables

Definition at line 308 of file `alias_sctp.c`.

5.5.2.4 #define SN_DEFAULT_HASH_SIZE 2003

A reasonable default size for the hash tables

Definition at line 304 of file `alias_sctp.c`.

5.5.2.5 #define SN_GLOBAL_TBL 0x02

assoc in global table

Definition at line 307 of file `alias_sctp.c`.

5.5.2.6 #define SN_LOCAL_TBL 0x01

assoc in local table

Definition at line 306 of file `alias_sctp.c`.

5.5.2.7 #define SN_MAX_GLOBAL_ADDRESSES 100

absolute maximum global address count

Definition at line 312 of file alias_sctp.c.

5.5.2.8 #define SN_MAX_HASH_SIZE 1000001

Maximum hash table size (NB must be less than max int)

Definition at line 303 of file alias_sctp.c.

5.5.2.9 #define SN_MIN_HASH_SIZE 101

Minimum hash table size (set to stop users choosing stupid values)

Definition at line 302 of file alias_sctp.c.

5.5.2.10 #define SN_NULL_TBL 0x00

assoc in No table

Definition at line 311 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.5.2.11 #define SN_TABLE_HASH(vtag, port, size) (((u_int) vtag + (u_int) port) % (u_int) size)

Calculate the hash table lookup position

Definition at line 317 of file alias_sctp.c.

Referenced by FindSctpGlobal(), FindSctpGlobalClash(), FindSctpGlobalT(), FindSctpLocal(), and FindSctpLocalT().

5.5.2.12 #define SN_WAIT_TOGLOBAL 0x20

assoc waiting for TOLOCAL asconf ACK

Definition at line 310 of file alias_sctp.c.

5.5.2.13 #define SN_WAIT_TOLOCAL 0x10

assoc waiting for TOLOCAL asconf ACK

Definition at line 309 of file alias_sctp.c.

5.5.3 Function Documentation**5.5.3.1 static int AddSctpAssocGlobal (struct libalias * *la*, struct sctp_nat_assoc * *assoc*)**
[static]

Add the sctp association information to the global look up table.

Searches the global look-up table for an existing association with the same details. If a match exists and is ONLY in the global look-up table then this is a repeated INIT packet, we need to remove this association from the look-up table and add the new association

The new association is added to the head of the list and state is updated

Parameters:

la Pointer to the relevant libalias instance
assoc pointer to sctp association

Returns:

SN_ADD_OK | SN_ADD_CLASH

Definition at line 2246 of file alias_sctp.c.

```

2249         {
2250     if ((found->TableRegister == SN_GLOBAL_TBL) &&
2251         (found->l_addr.s_addr == assoc->l_addr.s_addr) && (found->l_port == assoc->l_port)) { /* rese
2252     RmSctpAssoc(la, found);
2253     sctp_RmTimeOut(la, found);
2254     freeGlobalAddressList(found);
2255     sn_free(found);
2256     } else
2257     return(SN_ADD_CLASH);
2258 }
2259
2260 LIST_INSERT_HEAD(&la->sctpTableGlobal[SN_TABLE_HASH(assoc->g_vtag, assoc->g_port, la->sctpNatTableS
2261     assoc, list_G);
2262 assoc->TableRegister |= SN_GLOBAL_TBL;
2263 la->sctpLinkCount++; //increment link count
2264
2265 if (assoc->TableRegister == SN_BOTH_TBL) {
2266     /* libalias log -- controlled by libalias */
2267     if (la->packetAliasMode & PKT_ALIAS_LOG)
2268         SctpShowAliasStats(la);
2269
2270     SN_LOG(SN_LOG_INFO, logsctpassoc(assoc, "^"));
2271 }
2272
2273 return(SN_ADD_OK);
2274 }
2275

```

5.5.3.2 static int AddSctpAssocLocal (struct libalias * *la*, struct sctp_nat_assoc * *assoc*, struct in_addr *g_addr*) [static]

Add the sctp association information to the local look up table.

Searches the local look-up table for an existing association with the same details. If a match exists and is ONLY in the local look-up table then this is a repeated INIT packet, we need to remove this association from the look-up table and add the new association

The new association is added to the head of the list and state is updated

Parameters:

la Pointer to the relevant libalias instance
assoc pointer to sctp association

g_addr global address

Returns:

SN_ADD_OK | SN_ADD_CLASH

Definition at line 2189 of file alias_sctp.c.

```

2194                                     :
2195 * - the local receiver if receiving it for the first time will establish
2196 *   an association with the new global host
2197 * - if receiving an init from a different global address after sending a
2198 *   lost initack it will send an initack to the new global host, the first
2199 *   association attempt will then be blocked if retried.
2200 */
2201 if (found != NULL) {
2202     if ((found->TableRegister == SN_LOCAL_TBL) && (found->g_port == assoc->g_port)) { /* resent messa
2203         RmSctpAssoc(la, found);
2204         sctp_RmTimeOut(la, found);
2205         freeGlobalAddressList(found);
2206         sn_free(found);
2207     } else
2208         return(SN_ADD_CLASH);
2209 }
2210
2211 LIST_INSERT_HEAD(&la->sctpTableLocal[SN_TABLE_HASH(assoc->l_vtag, assoc->l_port, la->sctpNatTableSi
2212                 assoc, list_L);
2213 assoc->TableRegister |= SN_LOCAL_TBL;
2214 la->sctpLinkCount++; //increment link count
2215
2216 if (assoc->TableRegister == SN_BOTH_TBL) {
2217     /* libalias log -- controlled by libalias */
2218     if (la->packetAliasMode & PKT_ALIAS_LOG)
2219         SctpShowAliasStats(la);
2220
2221     SN_LOG(SN_LOG_INFO, logsctpassoc(assoc, "^"));
2222 }
2223
2224 return(SN_ADD_OK);
2225 }
2226

```

5.5.3.3 `static struct sctp_nat_assoc * FindSctpGlobal (struct libalias * la, struct in_addr g_addr, uint32_t g_vtag, uint16_t g_port, uint16_t l_port, int * partial_match)` [static, read]

Find the SCTP association given the global port and vtag.

Searches the global look-up table for the association entry matching the provided global <address:ports:vtag> tuple

If all but the global address match it sets *partial_match* to 1 to indicate a partial match. If the NAT is tracking global IP addresses for this association, the NAT may respond with an ERRORM to request the missing address to be added.

Parameters:

la Pointer to the relevant libalias instance

g_addr global address

g_vtag global vtag

g_port global port

l_port local port

Returns:

pointer to association or NULL

Definition at line 2065 of file alias_sctp.c.

References sctp_GlobalAddress::g_addr, sctp_nat_assoc::g_port, sctp_nat_assoc::g_vtag, sctp_nat_-assoc::l_port, sctp_nat_assoc::num_Gaddr, and SN_TABLE_HASH.

Referenced by sctp_PktParser().

```

2069             { /* an init packet, vtag==0 */
2070     i = SN_TABLE_HASH(g_vtag, g_port, la->sctpNatTableSize);
2071     LIST_FOREACH(assoc, &la->sctpTableGlobal[i], list_G) {
2072         if ((assoc->g_vtag == g_vtag) && (assoc->g_port == g_port) && (assoc->l_port == l_port)) {
2073             *partial_match = 1;
2074             if (assoc->num_Gaddr) {
2075                 LIST_FOREACH(G_Addr, &(assoc->Gaddr), list_Gaddr) {
2076                     if (G_Addr->g_addr.s_addr == g_addr.s_addr)
2077                         return(assoc);
2078                 }
2079             } else {
2080                 return(assoc);
2081             }
2082         }
2083     }
2084 }
2085 return(NULL);
2086 }
2087

```

5.5.3.4 static struct sctp_nat_assoc * FindSctpGlobalClash (struct libalias * la, struct sctp_nat_assoc * Cassoc) [static, read]

Check for Global Clash.

Searches the global look-up table for the association entry matching the provided global <(addresses):ports:vtag> tuple

Parameters:

la Pointer to the relevant libalias instance

Cassoc association being checked for a clash

Returns:

pointer to association or NULL

Definition at line 2018 of file alias_sctp.c.

References sctp_GlobalAddress::g_addr, sctp_nat_assoc::g_port, sctp_nat_assoc::g_vtag, sctp_nat_-assoc::l_port, sctp_nat_assoc::num_Gaddr, and SN_TABLE_HASH.

```

2022             { /* an init packet, vtag==0 */
2023     i = SN_TABLE_HASH(Cassoc->g_vtag, Cassoc->g_port, la->sctpNatTableSize);

```



```

2024     LIST_FOREACH(assoc, &la->sctpTableGlobal[i], list_G) {
2025         if ((assoc->g_vtag == Cassoc->g_vtag) && (assoc->g_port == Cassoc->g_port) && (assoc->l_port ==
2026             if (assoc->num_Gaddr) {
2027                 LIST_FOREACH(G_AddrC, &(Cassoc->Gaddr), list_Gaddr) {
2028                     LIST_FOREACH(G_Addr, &(assoc->Gaddr), list_Gaddr) {
2029                         if(G_Addr->g_addr.s_addr == G_AddrC->g_addr.s_addr)
2030                             return(assoc);
2031                     }
2032                 }
2033             } else {
2034                 return(assoc);
2035             }
2036         }
2037     }
2038 }
2039 return(NULL);
2040 }
2041

```

5.5.3.5 static struct sctp_nat_assoc * FindSctpGlobalT (struct libalias * *la*, struct in_addr *g_addr*, uint32_t *g_vtag*, uint16_t *l_port*, uint16_t *g_port*) [static, read]

Find the SCTP association for a T-Flag message (given the local port and global vtag).

Searches the global look-up table for a unique association entry matching the provided local port and global vtag information

Parameters:

la Pointer to the relevant libalias instance
g_addr global address
g_vtag global vtag
l_port local port
g_port global port

Returns:

pointer to association or NULL

Definition at line 2148 of file alias_sctp.c.

References sctp_GlobalAddress::g_addr, sctp_nat_assoc::g_port, sctp_nat_assoc::l_port, sctp_nat_assoc::l_vtag, sctp_nat_assoc::num_Gaddr, and SN_TABLE_HASH.

Referenced by sctp_PktParser().

```

2151         { /* an init packet, vtag==0 */
2152     i = SN_TABLE_HASH(g_vtag, l_port, la->sctpNatTableSize);
2153     LIST_FOREACH(assoc, &la->sctpTableLocal[i], list_L) {
2154         if ((assoc->l_vtag == g_vtag) && (assoc->l_port == l_port) && (assoc->g_port == g_port)) {
2155             if (assoc->num_Gaddr) {
2156                 LIST_FOREACH(G_Addr, &(assoc->Gaddr), list_Gaddr) {
2157                     if(G_Addr->g_addr.s_addr == g_addr.s_addr)
2158                         return(assoc);
2159                 }
2160             } else {
2161                 return(assoc);
2162             }
2163         }

```

```

2164     }
2165   }
2166   return(NULL);
2167 }
2168

```

5.5.3.6 `static struct sctp_nat_assoc * FindSctpLocal (struct libalias * la, struct in_addr l_addr, struct in_addr g_addr, uint32_t l_vtag, uint16_t l_port, uint16_t g_port)` [static, read]

Find the SCTP association given the local address, port and vtag.

Searches the local look-up table for the association entry matching the provided local <address:ports:vtag> tuple

Parameters:

la Pointer to the relevant libalias instance

l_addr local address

g_addr global address

l_vtag local Vtag

l_port local Port

g_port global Port

Returns:

pointer to association or NULL

Definition at line 1981 of file alias_sctp.c.

References `sctp_GlobalAddress::g_addr`, `sctp_nat_assoc::g_port`, `sctp_nat_assoc::l_addr`, `sctp_nat_assoc::l_port`, `sctp_nat_assoc::l_vtag`, `sctp_nat_assoc::num_Gaddr`, and `SN_TABLE_HASH`.

Referenced by `sctp_PktParser()`.

```

1984     { /* an init packet, vtag==0 */
1985     i = SN_TABLE_HASH(l_vtag, l_port, la->sctpNatTableSize);
1986     LIST_FOREACH(assoc, &la->sctpTableLocal[i], list_L) {
1987         if ((assoc->l_vtag == l_vtag) && (assoc->l_port == l_port) && (assoc->g_port == g_port)\
1988             && (assoc->l_addr.s_addr == l_addr.s_addr)) {
1989             if (assoc->num_Gaddr) {
1990                 LIST_FOREACH(G_Addr, &(assoc->Gaddr), list_Gaddr) {
1991                     if(G_Addr->g_addr.s_addr == g_addr.s_addr)
1992                         return(assoc);
1993                 }
1994             } else {
1995                 return(assoc);
1996             }
1997         }
1998     }
1999 }
2000 return(NULL);
2001 }
2002

```

5.5.3.7 `static struct sctp_nat_assoc * FindSctpLocalT (struct libalias * la, struct in_addr g_addr, uint32_t l_vtag, uint16_t g_port, uint16_t l_port)` [static, read]

Find the SCTP association for a T-Flag message (given the global port and local vtag).

Searches the local look-up table for a unique association entry matching the provided global port and local vtag information

Parameters:

la Pointer to the relevant libalias instance

g_addr global address

l_vtag local Vtag

g_port global Port

l_port local Port

Returns:

pointer to association or NULL

Definition at line 2106 of file alias_sctp.c.

References `sctp_GlobalAddress::g_addr`, `sctp_nat_assoc::g_port`, `sctp_nat_assoc::g_vtag`, `sctp_nat_assoc::l_port`, `sctp_nat_assoc::num_Gaddr`, and `SN_TABLE_HASH`.

Referenced by `sctp_PktParser()`.

```

2110             { /* an init packet, vtag==0 */
2111     i = SN_TABLE_HASH(l_vtag, g_port, la->sctpNatTableSize);
2112     LIST_FOREACH(assoc, &la->sctpTableGlobal[i], list_G) {
2113         if ((assoc->g_vtag == l_vtag) && (assoc->g_port == g_port) && (assoc->l_port == l_port)) {
2114             if (assoc->num_Gaddr) {
2115                 LIST_FOREACH(G_Addr, &(assoc->Gaddr), list_Gaddr) {
2116                     if (G_Addr->g_addr.s_addr == G_Addr->g_addr.s_addr)
2117                         return(assoc); /* full match */
2118                 }
2119             } else {
2120                 if (++cnt > 1) return(NULL);
2121                 lastmatch = assoc;
2122             }
2123         }
2124     }
2125 }
2126 /* If there is more than one match we do not know which local address to send to */
2127 return( cnt ? lastmatch : NULL );
2128 }
2129
```

5.5.3.8 `static void freeGlobalAddressList (struct sctp_nat_assoc * assoc)` [static]

free the Global Address List memory

`freeGlobalAddressList` deletes all global IP addresses in an associations global IP address list.

Parameters:

assoc

Definition at line 2334 of file alias_sctp.c.

```

2336             {
2337     gaddr2 = LIST_NEXT(gaddr1, list_Gaddr);
2338     sn_free(gaddr1);
2339     gaddr1 = gaddr2;
2340 }
2341 }
2342 /* -----
2343 *                               TIMER QUEUE CODE
2344 * -----

```

5.5.3.9 static void RmSctpAssoc (struct libalias * *la*, struct sctp_nat_assoc * *assoc*) [static]

Remove the sctp association information from the look up table.

For each of the two (local/global) look-up tables, remove the association from that table IF it has been registered in that table.

NOTE: The calling code is responsible for freeing memory allocated to the association structure itself

NOTE: The association is NOT removed from the timer queue

Parameters:

la Pointer to the relevant libalias instance

assoc pointer to sctp association

Definition at line 2294 of file alias_sctp.c.

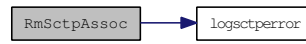
References logsctperror(), SN_LOG, SN_LOG_LOW, and SN_TO_NODIR.

```

2294             {
2295     /* very bad, log and die*/
2296     SN_LOG(SN_LOG_LOW,
2297     logsctperror("ERROR: alias_sctp:RmSctpAssoc(NULL)\n", 0, 0, SN_TO_NODIR));
2298     return;
2299 }
2300 /* log if association is fully up and now closing */
2301 if (assoc->TableRegister == SN_BOTH_TBL) {
2302     SN_LOG(SN_LOG_INFO, logsctpassoc(assoc, "$"));
2303 }
2304 LIBALIAS_LOCK_ASSERT(la);
2305 if (assoc->TableRegister & SN_LOCAL_TBL) {
2306     assoc->TableRegister ^= SN_LOCAL_TBL;
2307     la->sctpLinkCount--; //decrement link count
2308     LIST_REMOVE(assoc, list_L);
2309 }
2310
2311 if (assoc->TableRegister & SN_GLOBAL_TBL) {
2312     assoc->TableRegister ^= SN_GLOBAL_TBL;
2313     la->sctpLinkCount--; //decrement link count
2314     LIST_REMOVE(assoc, list_G);
2315 }
2316 // sn_free(assoc); //Don't remove now, remove if needed later
2317 /* libalias logging -- controlled by libalias log definition */
2318 if (la->packetAliasMode & PKT_ALIAS_LOG)
2319     SctpShowAliasStats(la);
2320 }
2321

```

Here is the call graph for this function:



5.6 Timer Queue Macros and Functions

Defines

- `#define SN_MIN_TIMER 1`
- `#define SN_MAX_TIMER 600`
- `#define SN_TIMER_QUEUE_SIZE SN_MAX_TIMER+2`
- `#define SN_I_T(la) (la → timeStamp + sysctl_init_timer)`
- `#define SN_U_T(la) (la → timeStamp + sysctl_up_timer)`
- `#define SN_C_T(la) (la → timeStamp + sysctl_shutdown_timer)`
- `#define SN_X_T(la) (la → timeStamp + sysctl_holddown_timer)`

Functions

- static void `sctp_AddTimeOut` (struct libalias *la, struct `sctp_nat_assoc` *assoc)
Add an association timeout to the timer queue.
- static void `sctp_RmTimeOut` (struct libalias *la, struct `sctp_nat_assoc` *assoc)
Remove an association from timer queue.
- static void `sctp_ResetTimeOut` (struct libalias *la, struct `sctp_nat_assoc` *assoc, int newexp)
Reset timer in timer queue.
- void `sctp_CheckTimers` (struct libalias *la)
Check timer Q against current time.

5.6.1 Detailed Description

Timer macros set minimum/maximum timeout values and calculate timer expiry times for the provided libalias instance

The timer queue management functions are designed to operate efficiently with a minimum of interaction with the queues.

Once a timeout is set in the queue it will not be altered in the queue unless it has to be changed to a shorter time (usually only for aborts and closing). On a queue timeout, the real expiry time is checked, and if not less than the timeout it is requeued (O(1)) at its later time. This is especially important for normal packets sent during an association. When a timer expires, it is updated to its new expiration time if necessary, or processed as a timeout. This means that while in UP state, the timing queue is only altered every U_T (every few minutes) for a particular association.

5.6.2 Define Documentation

5.6.2.1 `#define SN_C_T(la) (la → timeStamp + sysctl_shutdown_timer)`

CL State expiration time in seconds

Definition at line 331 of file alias_sctp.c.

5.6.2.2 #define SN_I_T(la) (la → timeStamp + sysctl_init_timer)

INIT State expiration time in seconds

Definition at line 329 of file alias_sctp.c.

5.6.2.3 #define SN_MAX_TIMER 600

Definition at line 326 of file alias_sctp.c.

5.6.2.4 #define SN_MIN_TIMER 1

Definition at line 325 of file alias_sctp.c.

5.6.2.5 #define SN_TIMER_QUEUE_SIZE SN_MAX_TIMER+2

Definition at line 327 of file alias_sctp.c.

5.6.2.6 #define SN_U_T(la) (la → timeStamp + sysctl_up_timer)

UP State expiration time in seconds

Definition at line 330 of file alias_sctp.c.

5.6.2.7 #define SN_X_T(la) (la → timeStamp + sysctl_holddown_timer)

Wait after a shutdown complete in seconds

Definition at line 332 of file alias_sctp.c.

5.6.3 Function Documentation**5.6.3.1 static void sctp_AddTimeOut (struct libalias * la, struct sctp_nat_assoc * assoc)**
[static]

Add an association timeout to the timer queue.

Determine the location in the queue to add the timeout and insert the association into the list at that queue position

Parameters:

la

assoc

Definition at line 2373 of file alias_sctp.c.

```
2392 {
```

5.6.3.2 void sctp_CheckTimers (struct libalias * la)

Check timer Q against current time.

Loop through each entry in the timer queue since the last time we processed the timer queue until now (the current time). For each association in the event list, we remove it from that position in the timer queue and check if it has really expired. If so we:

- Log the timer expiry
- Remove the association from the NAT tables
- Release the memory used by the association

If the timer hasn't really expired we place the association into its new correct position in the timer queue.

Parameters:

la Pointer to the relevant libalias instance

Definition at line 2441 of file alias_sctp.c.

```

2443                                     {
2444     while (!LIST_EMPTY(&la->sctpNatTimer.TimerQ[la->sctpNatTimer.cur_loc])) {
2445         assoc = LIST_FIRST(&la->sctpNatTimer.TimerQ[la->sctpNatTimer.cur_loc]);
2446         //SLIST_REMOVE_HEAD(&la->sctpNatTimer.TimerQ[la->sctpNatTimer.cur_loc], timer_Q);
2447         LIST_REMOVE(assoc, timer_Q);
2448         if (la->timeStamp >= assoc->exp) { /* state expired */
2449             SN_LOG(((assoc->state == SN_CL)?(SN_LOG_DEBUG):(SN_LOG_INFO)),
2450                 logsctperror("Timer Expired", assoc->g_vtag, assoc->state, SN_TO_NODIR));
2451             RmSctpAssoc(la, assoc);
2452             freeGlobalAddressList(assoc);
2453             sn_free(assoc);
2454         } else { /* state not expired, reschedule timer*/
2455             sctp_AddTimeOut(la, assoc);
2456         }
2457     }
2458     /* Goto next location in the timer queue*/
2459     ++la->sctpNatTimer.loc_time;
2460     if (++la->sctpNatTimer.cur_loc >= SN_TIMER_QUEUE_SIZE)
2461         la->sctpNatTimer.cur_loc = 0;
2462 }
2463 }
2464
2465 /* -----
2466 *                                     LOGGING CODE

```

5.6.3.3 static void sctp_ResetTimeOut (struct libalias * la, struct sctp_nat_assoc * assoc, int newexp) [static]

Reset timer in timer queue.

Reset the actual timeout for the specified association. If it is earlier than the existing timeout, then remove and re-install the association into the queue

Parameters:

la Pointer to the relevant libalias instance

assoc pointer to sctp association

newexp New expiration time

Definition at line 2413 of file alias_sctp.c.

```
2416     {
2417     assoc->exp = newexp;
2418     }
2419 }
2420
```

5.6.3.4 static void sctp_RmTimeOut (struct libalias * *la*, struct sctp_nat_assoc * *assoc*) [static]

Remove an association from timer queue.

This is an O(1) operation to remove the association pointer from its current position in the timer queue

Parameters:

la Pointer to the relevant libalias instance

assoc pointer to sctp association

Definition at line 2394 of file alias_sctp.c.

```
2411 {
```

5.7 SysCtl Variable and callback function declarations

Defines

- #define [SN_NO_ERROR_ON_OOTB](#) 0
- #define [SN_LOCAL_ERROR_ON_OOTB](#) 1
- #define [SN_LOCALandPARTIAL_ERROR_ON_OOTB](#) 2
- #define [SN_ERROR_ON_OOTB](#) 3

Functions

- int [sysctl_chg_loglevel](#) (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.fw.sctp.log_level
- int [sysctl_chg_timer](#) (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.fw.sctp.(init_timer|up_timer|shutdown_timer)
- int [sysctl_chg_hashtable_size](#) (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.alias.sctp.hashtable_size
- int [sysctl_chg_error_on_ootb](#) (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.alias.sctp.error_on_ootb
- int [sysctl_chg_accept_global_ootb_addip](#) (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.alias.sctp.accept_global_ootb_addip
- int [sysctl_chg_initialising_chunk_proc_limit](#) (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.alias.sctp.initialising_chunk_proc_limit
- int [sysctl_chg_chunk_proc_limit](#) (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.alias.sctp.chunk_proc_limit
- int [sysctl_chg_param_proc_limit](#) (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.alias.sctp.param_proc_limit
- int [sysctl_chg_track_global_addresses](#) (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.alias.sctp.track_global_addresses

Variables

- static u_int [sysctl_log_level](#) = 0
net.inet.ip.alias.sctp.log_level
- static u_int [sysctl_init_timer](#) = 15
net.inet.ip.alias.sctp.init_timer
- static u_int [sysctl_up_timer](#) = 300
net.inet.ip.alias.sctp.up_timer

- static u_int `sysctl_shutdown_timer` = 15
net.inet.ip.alias.sctp.shutdown_timer
- static u_int `sysctl_holddown_timer` = 0
net.inet.ip.alias.sctp.holddown_timer
- static u_int `sysctl_hashtable_size` = SN_DEFAULT_HASH_SIZE
net.inet.ip.alias.sctp.hashtable_size
- static u_int `sysctl_error_on_ootb` = 1
net.inet.ip.alias.sctp.error_on_ootb
- static u_int `sysctl_accept_global_ootb_addip` = 0
net.inet.ip.alias.sctp.accept_global_ootb_addip
- static u_int `sysctl_initialising_chunk_proc_limit` = 2
net.inet.ip.alias.sctp.initialising_chunk_proc_limit
- static u_int `sysctl_chunk_proc_limit` = 5
net.inet.ip.alias.sctp.param_proc_limit
- static u_int `sysctl_param_proc_limit` = 25
net.inet.ip.alias.sctp.param_proc_limit
- static u_int `sysctl_track_global_addresses` = 0
net.inet.ip.alias.sctp.track_global_addresses

5.7.1 Detailed Description

Sysctl variables to modify NAT functionality in real-time along with associated functions to manage modifications to the sysctl variables

5.7.2 Define Documentation

5.7.2.1 #define SN_ERROR_ON_OOTB 3

Send errorM on out of the blue packets

Definition at line 385 of file alias_sctp.c.

5.7.2.2 #define SN_LOCAL_ERROR_ON_OOTB 1

Send only local errorM on out of the blue packets

Definition at line 383 of file alias_sctp.c.

5.7.2.3 #define SN_LOCALandPARTIAL_ERROR_ON_OOTB 2

Send local errorM and global errorM for out of the blue packets only if partial match found

Definition at line 384 of file alias_sctp.c.

5.7.2.4 #define SN_NO_ERROR_ON_OOTB 0

Send no errorM on out of the blue packets

Definition at line 382 of file alias_sctp.c.

5.7.3 Function Documentation**5.7.3.1 int sysctl_chg_accept_global_ootb_addip (SYSCTL_HANDLER_ARGS)**

sysctl callback for changing net.inet.ip.alias.sctp.accept_global_ootb_addip

If set to 1 the NAT will accept ootb global addip messages for processing (Security risk) Default is 0, only responding to local ootb AddIP messages

Definition at line 538 of file alias_sctp.c.

```
543                                     : 0;
544
545     return (0);
546 }
547
```

5.7.3.2 int sysctl_chg_chunk_proc_limit (SYSCTL_HANDLER_ARGS)

sysctl callback for changing net.inet.ip.alias.sctp.chunk_proc_limit

Updates the chunk_proc_limit sysctl variable. Number of chunks that should be processed to find key chunk: >= initialising_chunk_proc_limit (A high value is a DoS risk)

Definition at line 580 of file alias_sctp.c.

```
586
587
588     return (0);
589 }
590
591
```

: procli

5.7.3.3 int sysctl_chg_error_on_ootb (SYSCTL_HANDLER_ARGS)

sysctl callback for changing net.inet.ip.alias.sctp.error_on_ootb

Updates the error_on_clash sysctl variable. If set to 0, no ErrorM will be sent if there is a look up table clash If set to 1, an ErrorM is sent only to the local side If set to 2, an ErrorM is sent to the local side and global side if there is a partial association match If set to 3, an ErrorM is sent to both local and global sides (DoS) risk.

Definition at line 519 of file alias_sctp.c.

```

524                                     : flag;
525
526     return (0);
527 }
528

```

5.7.3.4 int sysctl_chg_hashtable_size (SYSCTL_HANDLER_ARGS)

sysctl callback for changing net.inet.ip.alias.sctp.hashtable_size

Updates the hashtable_size sysctl variable. The new value should be a prime number. We sanity check to ensure that the size is within the range SN_MIN_HASH_SIZE-SN_MAX_HASH_SIZE. We then check the provided number to see if it is prime. We approximate by checking that (2,3,5,7,11) are not factors, incrementing the user provided value until we find a suitable number.

Definition at line 491 of file alias_sctp.c.

```

496                                     : ((size > SN_MAX_HASH_SIZE) ? (SN_MAX_HASH_SIZE) : (s
497
498     size |= 0x00000001; /* make odd */
499
500     for(;;((size % 3) == 0) || ((size % 5) == 0) || ((size % 7) == 0) || ((size % 11) == 0)); size+=2);
501     sysctl_hashtable_size = size;
502
503     return (0);
504 }
505

```

5.7.3.5 int sysctl_chg_initialising_chunk_proc_limit (SYSCTL_HANDLER_ARGS)

sysctl callback for changing net.inet.ip.alias.sctp.initialising_chunk_proc_limit

Updates the initialising_chunk_proc_limit sysctl variable. Number of chunks that should be processed if there is no current association found: > 0 (A high value is a DoS risk)

Definition at line 558 of file alias_sctp.c.

```

563                                     : proclimit;
564     sysctl_chunk_proc_limit =
565         (sysctl_chunk_proc_limit < sysctl_initialising_chunk_proc_limit) ? sysctl_initialising_chunk_proc_
566
567     return (0);
568 }
569

```

5.7.3.6 int sysctl_chg_loglevel (SYSCTL_HANDLER_ARGS)

sysctl callback for changing net.inet.ip.fw.sctp.log_level

Updates the variable sysctl_log_level to the provided value and ensures it is in the valid range (SN_LOG_LOW -> SN_LOG_DEBUG)

Definition at line 441 of file alias_sctp.c.

```

446                                     : (level);
447     sysctl_log_level = (level < SN_LOG_LOW) ? (SN_LOG_LOW) : (level);
448

```

```

449     return (0);
450 }
451

```

5.7.3.7 int sysctl_chg_param_proc_limit (SYSCTL_HANDLER_ARGS)

sysctl callback for changing net.inet.ip.alias.sctp.param_proc_limit

Updates the param_proc_limit sysctl variable. Number of parameters that should be processed to find key parameters: > 1 (A high value is a DoS risk)

Definition at line 602 of file alias_sctp.c.

```

608                                     : proclimit;
609
610     return (0);
611 }
612

```

5.7.3.8 int sysctl_chg_timer (SYSCTL_HANDLER_ARGS)

sysctl callback for changing net.inet.ip.fw.sctp.(init_timer|up_timer|shutdown_timer)

Updates the timer-based sysctl variables. The new values are sanity-checked to make sure that they are within the range SN_MIN_TIMER-SN_MAX_TIMER. The holddown timer is allowed to be 0

Definition at line 462 of file alias_sctp.c.

```

467                                     : (timer);
468
469     if (((u_int *)arg1) != &sysctl_holddown_timer)
470     {
471         timer = (timer < SN_MIN_TIMER) ? (SN_MIN_TIMER) : (timer);
472     }
473
474     *(u_int *)arg1 = timer;
475
476     return (0);
477 }
478

```

5.7.3.9 int sysctl_chg_track_global_addresses (SYSCTL_HANDLER_ARGS)

sysctl callback for changing net.inet.ip.alias.sctp.track_global_addresses

Configures the global address tracking option within the NAT (0 - Global tracking is disabled, > 0 - enables tracking but limits the number of global IP addresses to this value)

Definition at line 623 of file alias_sctp.c.

```

628
629
630     return (0);
631 }
632
633
634 /* -----

```

5.7.4 Variable Documentation

5.7.4.1 `u_int sysctl_accept_global_ootb_addip = 0` [static]

net.inet.ip.alias.sctp.accept_global_ootb_addip

NAT responset to receipt of global OOTB AddIP (0 - No response, 1 - NAT will accept OOTB global AddIP messages for processing (Security risk))

Definition at line 371 of file alias_sctp.c.

5.7.4.2 `u_int sysctl_chunk_proc_limit = 5` [static]

net.inet.ip.alias.sctp.param_proc_limit

A limit on the number of chunks that should be searched (DoS prevention)

Definition at line 375 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.7.4.3 `u_int sysctl_error_on_ootb = 1` [static]

net.inet.ip.alias.sctp.error_on_ootb

NAT response to receipt of OOTB packet (0 - No response, 1 - NAT will send ErrorM only to local side, 2 - NAT will send local ErrorM and global ErrorM if there was a partial association match 3 - NAT will send ErrorM to both local and global)

Definition at line 366 of file alias_sctp.c.

5.7.4.4 `u_int sysctl_hashtable_size = SN_DEFAULT_HASH_SIZE` [static]

net.inet.ip.alias.sctp.hashtable_size

Sets the hash table size for any NEW NAT instances (existing instances retain their existing Hash Table

Definition at line 364 of file alias_sctp.c.

5.7.4.5 `u_int sysctl_holddown_timer = 0` [static]

net.inet.ip.alias.sctp.holddown_timer

Seconds to hold an association in the table after it has been shutdown (to allow for lost SHUTDOWN-COMPLETES)

Definition at line 362 of file alias_sctp.c.

5.7.4.6 `u_int sysctl_init_timer = 15` [static]

net.inet.ip.alias.sctp.init_timer

Seconds to hold an association in the table waiting for an INIT-ACK or AddIP-ACK

Definition at line 356 of file alias_sctp.c.

5.7.4.7 u_int sysctl_initialising_chunk_proc_limit = 2 [static]

net.inet.ip.alias.sctp.initialising_chunk_proc_limit

A limit on the number of chunks that should be searched if there is no matching association (DoS prevention)

Definition at line 373 of file alias_sctp.c.

Referenced by sctp_PktParser().

5.7.4.8 u_int sysctl_log_level = 0 [static]

net.inet.ip.alias.sctp.log_level

Stores the current level of logging

Definition at line 354 of file alias_sctp.c.

5.7.4.9 u_int sysctl_param_proc_limit = 25 [static]

net.inet.ip.alias.sctp.param_proc_limit

A limit on the number of parameters (in chunks) that should be searched (DoS prevention)

Definition at line 377 of file alias_sctp.c.

5.7.4.10 u_int sysctl_shutdown_timer = 15 [static]

net.inet.ip.alias.sctp.shutdown_timer

Seconds to hold an association in the table waiting for a SHUTDOWN-COMPLETE

Definition at line 360 of file alias_sctp.c.

5.7.4.11 u_int sysctl_track_global_addresses = 0 [static]

net.inet.ip.alias.sctp.track_global_addresses

Configures the global address tracking option within the NAT (0 - Global tracking is disabled, > 0 - enables tracking but limits the number of global IP addresses to this value) If set to >=1 the NAT will track that many global IP addresses. This may reduce look up table conflicts, but increases processing

Definition at line 379 of file alias_sctp.c.

5.7.4.12 u_int sysctl_up_timer = 300 [static]

net.inet.ip.alias.sctp.up_timer

Seconds to hold an association in the table while no packets are transmitted

Definition at line 358 of file alias_sctp.c.

Chapter 6

Data Structure Documentation

6.1 sctp_GlobalAddress Struct Reference

```
#include <alias_sctp.h>
```

Public Member Functions

- [LIST_ENTRY \(sctp_GlobalAddress\)](#) list_Gaddr

Data Fields

- struct in_addr [g_addr](#)

6.1.1 Detailed Description

Definition at line 154 of file alias_sctp.h.

6.1.2 Member Function Documentation

6.1.2.1 sctp_GlobalAddress::LIST_ENTRY (sctp_GlobalAddress)

Linked list of pointers for Global table

6.1.3 Field Documentation

6.1.3.1 struct in_addr sctp_GlobalAddress::g_addr [read]

Definition at line 155 of file alias_sctp.h.

Referenced by FindSctpGlobal(), FindSctpGlobalClash(), FindSctpGlobalT(), FindSctpLocal(), and FindSctpLocalT().

The documentation for this struct was generated from the following file:

- [alias_sctp.h](#)

6.2 sctp_nat_assoc Struct Reference

sctp association information

```
#include <alias_sctp.h>
```

Public Member Functions

- [LIST_HEAD](#) (sctpGlobalAddresshead, [sctp_GlobalAddress](#)) Gaddr
- [LIST_ENTRY](#) (sctp_nat_assoc) list_L
- [LIST_ENTRY](#) (sctp_nat_assoc) list_G
- [LIST_ENTRY](#) (sctp_nat_assoc) timer_Q

Data Fields

- [uint32_t l_vtag](#)
- [uint16_t l_port](#)
- [uint32_t g_vtag](#)
- [uint16_t g_port](#)
- [struct in_addr l_addr](#)
- [struct in_addr a_addr](#)
- [int state](#)
- [int TableRegister](#)
- [int exp](#)
- [int exp_loc](#)
- [int num_Gaddr](#)

6.2.1 Detailed Description

sctp association information

Structure that contains information about a particular sctp association currently under Network Address Translation. Information is stored in network byte order (as is libalias)***

Definition at line 135 of file alias_sctp.h.

6.2.2 Member Function Documentation

6.2.2.1 sctp_nat_assoc::LIST_ENTRY (sctp_nat_assoc)

Linked list of pointers for timer Q

6.2.2.2 sctp_nat_assoc::LIST_ENTRY (sctp_nat_assoc)

Linked list of pointers for Global table

6.2.2.3 sctp_nat_assoc::LIST_ENTRY (sctp_nat_assoc)

Linked list of pointers for Local table

6.2.2.4 sctp_nat_assoc::LIST_HEAD (sctpGlobalAddresshead, sctp_GlobalAddress)

List of global addresses

6.2.3 Field Documentation

6.2.3.1 struct in_addr sctp_nat_assoc::a_addr [read]

alias ip address

Definition at line 141 of file alias_sctp.h.

6.2.3.2 int sctp_nat_assoc::exp

timer expiration in seconds from uptime

Definition at line 144 of file alias_sctp.h.

6.2.3.3 int sctp_nat_assoc::exp_loc

current location in timer_Q

Definition at line 145 of file alias_sctp.h.

6.2.3.4 uint16_t sctp_nat_assoc::g_port

global side port number

Definition at line 139 of file alias_sctp.h.

Referenced by FindSctpGlobal(), FindSctpGlobalClash(), FindSctpGlobalT(), FindSctpLocal(), and FindSctpLocalT().

6.2.3.5 uint32_t sctp_nat_assoc::g_vtag

global side verification tag

Definition at line 138 of file alias_sctp.h.

Referenced by FindSctpGlobal(), FindSctpGlobalClash(), and FindSctpLocalT().

6.2.3.6 struct in_addr sctp_nat_assoc::l_addr [read]

local ip address

Definition at line 140 of file alias_sctp.h.

Referenced by FindSctpLocal().

6.2.3.7 uint16_t sctp_nat_assoc::l_port

local side port number

Definition at line 137 of file alias_sctp.h.

Referenced by FindSctpGlobal(), FindSctpGlobalClash(), FindSctpGlobalT(), FindSctpLocal(), and FindSctpLocalT().

6.2.3.8 uint32_t sctp_nat_assoc::l_vtag

local side verification tag

Definition at line 136 of file alias_sctp.h.

Referenced by FindSctpGlobalT(), and FindSctpLocal().

6.2.3.9 int sctp_nat_assoc::num_Gaddr

number of global IP addresses in the list

Definition at line 146 of file alias_sctp.h.

Referenced by FindSctpGlobal(), FindSctpGlobalClash(), FindSctpGlobalT(), FindSctpLocal(), and FindSctpLocalT().

6.2.3.10 int sctp_nat_assoc::state

current state of NAT association

Definition at line 142 of file alias_sctp.h.

Referenced by ProcessSctpMsg().

6.2.3.11 int sctp_nat_assoc::TableRegister

stores which look up tables association is registered in

Definition at line 143 of file alias_sctp.h.

The documentation for this struct was generated from the following file:

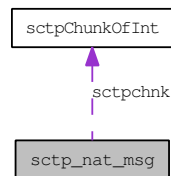
- [alias_sctp.h](#)

6.3 sctp_nat_msg Struct Reference

SCTP message.

```
#include <alias_sctp.h>
```

Collaboration diagram for sctp_nat_msg:



Data Fields

- uint16_t msg
- struct ip * ip_hdr
- struct sctphdr * sctp_hdr
- union sctpChunkOfInt sctpchnk
- int chunk_length

6.3.1 Detailed Description

SCTP message.

Structure containing the relevant information from the SCTP message

Definition at line 176 of file alias_sctp.h.

6.3.2 Field Documentation

6.3.2.1 int sctp_nat_msg::chunk_length

length of chunk of interest

Definition at line 185 of file alias_sctp.h.

Referenced by sctp_PktParser().

6.3.2.2 struct ip* sctp_nat_msg::ip_hdr [read]

pointer to ip packet header

Definition at line 181 of file alias_sctp.h.

Referenced by sctp_PktParser().

6.3.2.3 uint16_t sctp_nat_msg::msg

one of the key messages defined above

Definition at line 177 of file alias_sctp.h.

Referenced by sctp_PktParser().

6.3.2.4 struct sctphdr* sctp_nat_msg::sctp_hdr [read]

pointer to sctp common header

Definition at line 183 of file alias_sctp.h.

Referenced by sctp_PktParser().

6.3.2.5 union sctpChunkOfInt sctp_nat_msg::sctpchnk [write]

union of pointers to the chunk of interest

Definition at line 184 of file alias_sctp.h.

Referenced by sctp_PktParser().

The documentation for this struct was generated from the following file:

- [alias_sctp.h](#)

6.4 sctp_nat_timer Struct Reference

sctp nat timer queue structure

```
#include <alias_sctp.h>
```

Public Member Functions

- [LIST_HEAD](#) (sctpTimerQ, [sctp_nat_assoc](#))*TimerQ

Data Fields

- [int loc_time](#)
- [int cur_loc](#)

6.4.1 Detailed Description

sctp nat timer queue structure

Definition at line 194 of file [alias_sctp.h](#).

6.4.2 Member Function Documentation

6.4.2.1 [sctp_nat_timer::LIST_HEAD](#) (sctpTimerQ, [sctp_nat_assoc](#))

List of associations at this position in the timer Q

6.4.3 Field Documentation

6.4.3.1 [int sctp_nat_timer::cur_loc](#)

index of the current location in the circular queue

Definition at line 196 of file [alias_sctp.h](#).

6.4.3.2 [int sctp_nat_timer::loc_time](#)

time in seconds for the current location in the queue

Definition at line 195 of file [alias_sctp.h](#).

The documentation for this struct was generated from the following file:

- [alias_sctp.h](#)

6.5 sctpChunkOfInt Union Reference

SCTP chunk of interest.

```
#include <alias_sctp.h>
```

Data Fields

- struct sctp_init * [Init](#)
- struct sctp_init_ack * [InitAck](#)
- struct sctp_paramhdr * [Asconf](#)

6.5.1 Detailed Description

SCTP chunk of interest.

The only chunks whose contents are of any interest are the INIT and ASCONF_AddIP

Definition at line 164 of file alias_sctp.h.

6.5.2 Field Documentation

6.5.2.1 struct sctp_paramhdr* sctpChunkOfInt::Asconf [read]

Pointer to ASCONF chunk

Definition at line 167 of file alias_sctp.h.

Referenced by sctp_PktParser().

6.5.2.2 struct sctp_init* sctpChunkOfInt::Init [read]

Pointer to Init Chunk

Definition at line 165 of file alias_sctp.h.

Referenced by sctp_PktParser().

6.5.2.3 struct sctp_init_ack* sctpChunkOfInt::InitAck [read]

Pointer to Init Chunk

Definition at line 166 of file alias_sctp.h.

Referenced by sctp_PktParser().

The documentation for this union was generated from the following file:

- [alias_sctp.h](#)

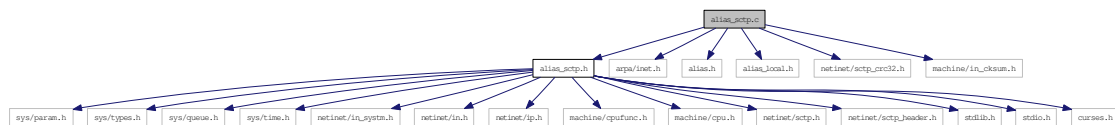
Chapter 7

File Documentation

7.1 alias_sctp.c File Reference

```
#include "alias_sctp.h"  
#include <arpa/inet.h>  
#include "alias.h"  
#include "alias_local.h"  
#include <netinet/sctp_crc32.h>  
#include <machine/in_cksum.h>
```

Include dependency graph for alias_sctp.c:



Defines

- #define [sn_malloc\(x\)](#) malloc(x)
- #define [sn_calloc\(n, x\)](#) calloc(n, x)
- #define [sn_free\(x\)](#) free(x)
- #define [SN_SCTP_FIRSTCHUNK\(sctphead\)](#) (struct sctp_chunkhdr *)(((char *)sctphead) + sizeof(struct sctphdr))
- #define [SN_SCTP_NEXTCHUNK\(chunkhead\)](#) (struct sctp_chunkhdr *)(((char *)chunkhead) + SCTP_SIZE32(ntohs(chunkhead → chunk_length)))
- #define [SN_SCTP_NEXTPARAM\(param\)](#) (struct sctp_paramhdr *)(((char *)param) + SCTP_SIZE32(ntohs(param → param_length)))
- #define [SN_MIN_CHUNK_SIZE](#) 4
- #define [SN_MIN_PARAM_SIZE](#) 4
- #define [SN_VTAG_PARAM_SIZE](#) 12
- #define [SN_ASCONFACK_PARAM_SIZE](#) 8
- #define [SN_PARSE_OK](#) 0

- #define `SN_PARSE_ERROR_IPSHL` 1
- #define `SN_PARSE_ERROR_AS_MALLOC` 2
- #define `SN_PARSE_ERROR_CHHL` 3
- #define `SN_PARSE_ERROR_DIR` 4
- #define `SN_PARSE_ERROR_VTAG` 5
- #define `SN_PARSE_ERROR_CHUNK` 6
- #define `SN_PARSE_ERROR_PORT` 7
- #define `SN_PARSE_ERROR_LOOKUP` 8
- #define `SN_PARSE_ERROR_PARTIALLOOKUP` 9
- #define `SN_PARSE_ERROR_LOOKUP_ABORT` 10
- #define `SN_SCTP_ABORT` 0x0000
- #define `SN_SCTP_INIT` 0x0001
- #define `SN_SCTP_INITACK` 0x0002
- #define `SN_SCTP_SHUTCOMP` 0x0010
- #define `SN_SCTP_SHUTACK` 0x0020
- #define `SN_SCTP_ASCONF` 0x0100
- #define `SN_SCTP_ASCONFACK` 0x0200
- #define `SN_SCTP_OTHER` 0xFFFF
- #define `SN_ID` 0x0000
- #define `SN_INi` 0x0010
- #define `SN_INa` 0x0020
- #define `SN_UP` 0x0100
- #define `SN_CL` 0x1000
- #define `SN_RM` 0x2000
- #define `SN_LOG_LOW` 0
- #define `SN_LOG_EVENT` 1
- #define `SN_LOG_INFO` 2
- #define `SN_LOG_DETAIL` 3
- #define `SN_LOG_DEBUG` 4
- #define `SN_LOG_DEBUG_MAX` 5
- #define `SN_LOG(level, action)` if (`sysctl_log_level` >= level) { action; }
- #define `SN_MIN_HASH_SIZE` 101
- #define `SN_MAX_HASH_SIZE` 1000001
- #define `SN_DEFAULT_HASH_SIZE` 2003
- #define `SN_LOCAL_TBL` 0x01
- #define `SN_GLOBAL_TBL` 0x02
- #define `SN_BOTH_TBL` 0x03
- #define `SN_WAIT_TOLOCAL` 0x10
- #define `SN_WAIT_TOGLOBAL` 0x20
- #define `SN_NULL_TBL` 0x00
- #define `SN_MAX_GLOBAL_ADDRESSES` 100
- #define `SN_ADD_OK` 0
- #define `SN_ADD_CLASH` 1
- #define `SN_TABLE_HASH(vtag, port, size)` (((u_int) vtag + (u_int) port) % (u_int) size)
- #define `SN_MIN_TIMER` 1
- #define `SN_MAX_TIMER` 600
- #define `SN_TIMER_QUEUE_SIZE` SN_MAX_TIMER+2
- #define `SN_I_T(la)` (la → timeStamp + `sysctl_init_timer`)
- #define `SN_U_T(la)` (la → timeStamp + `sysctl_up_timer`)
- #define `SN_C_T(la)` (la → timeStamp + `sysctl_shutdown_timer`)

- #define `SN_X_T(la)` (`la` → `timeStamp + systcl_holddown_timer`)
- #define `SN_NO_ERROR_ON_OOTB` 0
- #define `SN_LOCAL_ERROR_ON_OOTB` 1
- #define `SN_LOCALandPARTIAL_ERROR_ON_OOTB` 2
- #define `SN_ERROR_ON_OOTB` 3
- #define `SCTP_MIDDLEBOX_FLAG` 0x02
- #define `SCTP_NAT_TABLE_COLLISION` 0x00b0
- #define `SCTP_MISSING_NAT` 0x00b1
- #define `SCTP_VTAG_PARAM` 0xC007

Functions

- static int `sctp_PktParser` (struct `libalias *la`, int `direction`, struct `ip *pip`, struct `sctp_nat_msg *sm`, struct `sctp_nat_assoc **passoc`)
Parses SCTP packets for the key SCTP chunk that will be processed.
- static int `GetAsconfVtags` (struct `libalias *la`, struct `sctp_nat_msg *sm`, uint32_t `*l_vtag`, uint32_t `*g_vtag`, int `direction`)
Extract Vtags from Asconf Chunk.
- static int `IsASCONFack` (struct `libalias *la`, struct `sctp_nat_msg *sm`, int `direction`)
Check that ASCONF was successful.
- static void `AddGlobalIPAddresses` (struct `sctp_nat_msg *sm`, struct `sctp_nat_assoc *assoc`, int `direction`)
AddGlobalIPAddresses from Init,InitAck,or AddIP packets.
- static int `Add_Global_Address_to_List` (struct `sctp_nat_assoc *assoc`, struct `sctp_GlobalAddress *G_addr`)
Add_Global_Address_to_List.
- static void `RmGlobalIPAddresses` (struct `sctp_nat_msg *sm`, struct `sctp_nat_assoc *assoc`, int `direction`)
RmGlobalIPAddresses from DelIP packets.
- static int `IsADDorDEL` (struct `libalias *la`, struct `sctp_nat_msg *sm`, int `direction`)
Check to see if ASCONF contains an Add IP or Del IP parameter.
- static int `ProcessSctpMsg` (struct `libalias *la`, int `direction`, struct `sctp_nat_msg *sm`, struct `sctp_nat_assoc *assoc`)
Process SCTP message.
- static int `ID_process` (struct `libalias *la`, int `direction`, struct `sctp_nat_assoc *assoc`, struct `sctp_nat_msg *sm`)
Process SCTP message while in the Idle state.
- static int `INi_process` (struct `libalias *la`, int `direction`, struct `sctp_nat_assoc *assoc`, struct `sctp_nat_msg *sm`)
Process SCTP message while waiting for an INIT-ACK message.

- static int `INa_process` (struct libalias *la, int direction, struct `sctp_nat_assoc` *assoc, struct `sctp_nat_msg` *sm)
Process SCTP message while waiting for an AddIp-ACK message.
- static int `UP_process` (struct libalias *la, int direction, struct `sctp_nat_assoc` *assoc, struct `sctp_nat_msg` *sm)
Process SCTP messages while association is UP redirecting packets.
- static int `CL_process` (struct libalias *la, int direction, struct `sctp_nat_assoc` *assoc, struct `sctp_nat_msg` *sm)
Process SCTP message while association is in the process of closing.
- static void `TxAbortErrorM` (struct libalias *la, struct `sctp_nat_msg` *sm, struct `sctp_nat_assoc` *assoc, int sndrply, int direction)
Send an AbortM or ErrorM.
- static struct `sctp_nat_assoc` * `FindSctpLocal` (struct libalias *la, struct in_addr l_addr, struct in_addr g_addr, uint32_t l_vtag, uint16_t l_port, uint16_t g_port)
Find the SCTP association given the local address, port and vtag.
- static struct `sctp_nat_assoc` * `FindSctpGlobal` (struct libalias *la, struct in_addr g_addr, uint32_t g_vtag, uint16_t g_port, uint16_t l_port, int *partial_match)
Find the SCTP association given the global port and vtag.
- static struct `sctp_nat_assoc` * `FindSctpGlobalClash` (struct libalias *la, struct `sctp_nat_assoc` *Cassoc)
Check for Global Clash.
- static struct `sctp_nat_assoc` * `FindSctpLocalT` (struct libalias *la, struct in_addr g_addr, uint32_t l_vtag, uint16_t g_port, uint16_t l_port)
Find the SCTP association for a T-Flag message (given the global port and local vtag).
- static struct `sctp_nat_assoc` * `FindSctpGlobalT` (struct libalias *la, struct in_addr g_addr, uint32_t g_vtag, uint16_t l_port, uint16_t g_port)
Find the SCTP association for a T-Flag message (given the local port and global vtag).
- static int `AddSctpAssocLocal` (struct libalias *la, struct `sctp_nat_assoc` *assoc, struct in_addr g_addr)
Add the sctp association information to the local look up table.
- static int `AddSctpAssocGlobal` (struct libalias *la, struct `sctp_nat_assoc` *assoc)
Add the sctp association information to the global look up table.
- static void `RmSctpAssoc` (struct libalias *la, struct `sctp_nat_assoc` *assoc)
Remove the sctp association information from the look up table.
- static void `freeGlobalAddressList` (struct `sctp_nat_assoc` *assoc)
free the Global Address List memory
- static void `sctp_AddTimeOut` (struct libalias *la, struct `sctp_nat_assoc` *assoc)
Add an association timeout to the timer queue.

- static void `sctp_RmTimeOut` (struct libalias *la, struct `sctp_nat_assoc` *assoc)
Remove an association from timer queue.
- static void `sctp_ResetTimeOut` (struct libalias *la, struct `sctp_nat_assoc` *assoc, int newexp)
Reset timer in timer queue.
- void `sctp_CheckTimers` (struct libalias *la)
Check timer Q against current time.
- static void `logsctpperror` (char *errmsg, uint32_t vtag, int error, int direction)
Log sctp nat errors.
- static void `logsctpparse` (int direction, struct `sctp_nat_msg` *sm)
Log what the parser parsed.
- static void `logsctppassoc` (struct `sctp_nat_assoc` *assoc, char *s)
Log an SCTP association's details.
- static void `logTimerQ` (struct libalias *la)
Output timer queue to log.
- static void `logSctpGlobal` (struct libalias *la)
Output Global table to log.
- static void `logSctpLocal` (struct libalias *la)
Output Local table to log.
- void `SctpShowAliasStats` (struct libalias *la)
Log current statistics for the libalias instance.
- struct in_addr `FindSctpRedirectAddress` (struct libalias *la, struct `sctp_nat_msg` *sm)
Find the address to redirect incoming packets.
- int `sysctl_chg_loglevel` (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.fw.sctp.log_level
- int `sysctl_chg_timer` (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.fw.sctp.(init_timer|up_timer|shutdown_timer)
- int `sysctl_chg_hashtable_size` (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.alias.sctp.hashtable_size
- int `sysctl_chg_error_on_ootb` (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.alias.sctp.error_on_ootb
- int `sysctl_chg_accept_global_ootb_addip` (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.alias.sctp.accept_global_ootb_addip
- int `sysctl_chg_initialising_chunk_proc_limit` (SYSCTL_HANDLER_ARGS)

sysctl callback for changing net.inet.ip.alias.sctp.initialising_chunk_proc_limit

- int [sysctl_chg_chunk_proc_limit](#) (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.alias.sctp.chunk_proc_limit
- int [sysctl_chg_param_proc_limit](#) (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.alias.sctp.param_proc_limit
- int [sysctl_chg_track_global_addresses](#) (SYSCTL_HANDLER_ARGS)
sysctl callback for changing net.inet.ip.alias.sctp.track_global_addresses
- void [AliasSctpInit](#) (struct libalias *la)
Initialises the SCTP NAT Implementation.
- void [AliasSctpTerm](#) (struct libalias *la)
Cleans-up the SCTP NAT Implementation prior to unloading.
- int [SctpAlias](#) (struct libalias *la, struct ip *pip, int direction)
Handles SCTP packets passed from libalias.
- static void [SctpAliasLog](#) (FILE *stream, const char *format,...)
Sctp NAT logging function.

Variables

- static u_int [sysctl_log_level](#) = 0
net.inet.ip.alias.sctp.log_level
- static u_int [sysctl_init_timer](#) = 15
net.inet.ip.alias.sctp.init_timer
- static u_int [sysctl_up_timer](#) = 300
net.inet.ip.alias.sctp.up_timer
- static u_int [sysctl_shutdown_timer](#) = 15
net.inet.ip.alias.sctp.shutdown_timer
- static u_int [sysctl_holddown_timer](#) = 0
net.inet.ip.alias.sctp.holddown_timer
- static u_int [sysctl_hashtable_size](#) = SN_DEFAULT_HASH_SIZE
net.inet.ip.alias.sctp.hashtable_size
- static u_int [sysctl_error_on_ootb](#) = 1
net.inet.ip.alias.sctp.error_on_ootb
- static u_int [sysctl_accept_global_ootb_addip](#) = 0
net.inet.ip.alias.sctp.accept_global_ootb_addip

- static u_int `sysctl_initialising_chunk_proc_limit` = 2

net.inet.ip.alias.sctp.initialising_chunk_proc_limit

- static u_int `sysctl_chunk_proc_limit` = 5

net.inet.ip.alias.sctp.param_proc_limit

- static u_int `sysctl_param_proc_limit` = 25

net.inet.ip.alias.sctp.param_proc_limit

- static u_int `sysctl_track_global_addresses` = 0

net.inet.ip.alias.sctp.track_global_addresses

7.1.1 Detailed Description

Copyright (c) 2008, Centre for Advanced Internet Architectures Swinburne University of Technology, Melbourne, Australia (CRICOS number 00111D).

`alias_sctp` forms part of the `libalias` kernel module to handle Network Address Translation (NAT) for the SCTP protocol.

This software was developed by David A. Hayes and Jason But

Development is part of the CAIA SONATA project, proposed by Jason But and Grenville Armitage: <http://caia.swin.edu.au/urp/sonata/>

This project has been made possible in part by a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The names of the authors, the "Centre for Advanced Internet Architectures" and "Swinburne University of Technology" may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Definition in file `alias_sctp.c`.

7.1.2 Define Documentation

7.1.2.1 #define SCTP_MIDDLEBOX_FLAG 0x02

7.1.2.2 #define SCTP_MISSING_NAT 0x00b1

7.1.2.3 #define SCTP_NAT_TABLE_COLLISION 0x00b0

7.1.2.4 #define SCTP_VTAG_PARAM 0xC007

7.1.2.5 #define sn_calloc(n, x) calloc(n, x)

Definition at line 217 of file alias_sctp.c.

7.1.2.6 #define sn_free(x) free(x)

Definition at line 218 of file alias_sctp.c.

7.1.2.7 #define sn_malloc(x) malloc(x)

Definition at line 216 of file alias_sctp.c.

Referenced by sctp_PktParser().

7.1.3 Function Documentation

7.1.3.1 static int Add_Global_Address_to_List (struct sctp_nat_assoc * *assoc*, struct sctp_GlobalAddress * *G_addr*) [static]

Add_Global_Address_to_List.

Adds a global IP address to an associations address list, if it is not already there. The first address added us usually the packet's address, and is most likely to be used, so it is added at the beginning. Subsequent addresses are added after this one.

Parameters:

assoc Pointer to the association this SCTP Message belongs to

G_addr Pointer to the global address to add

Returns:

1 - success | 0 - fail

Definition at line 1427 of file alias_sctp.c.

```

1428                                     {
1429     LIST_INSERT_HEAD(&(assoc->Gaddr), G_addr, list_Gaddr); /* add new address to beginning of list*/
1430 } else {
1431     LIST_FOREACH(iter_G_Addr, &(assoc->Gaddr), list_Gaddr) {
1432         if (G_addr->g_addr.s_addr == iter_G_Addr->g_addr.s_addr)
1433             return(0); /* already exists, so don't add */
1434     }
1435     LIST_INSERT_AFTER(first_G_Addr, G_addr, list_Gaddr); /* add address to end of list*/

```



```

1436 }
1437 assoc->num_Gaddr++;
1438 return(1); /* success */
1439 }
1440

```

7.1.3.2 void AliasSctpInit (struct libalias * *la*)

Initialises the SCTP NAT Implementation.

Creates the look-up tables and the timer queue and initialises all state variables

Parameters:

la Pointer to the relevant libalias instance

Definition at line 649 of file alias_sctp.c.

```

652                                     :%d)\n", la->sctpNatTableSize));
653 la->sctpTableLocal = sn_calloc(la->sctpNatTableSize, sizeof(struct sctpNatTableL));
654 la->sctpTableGlobal = sn_calloc(la->sctpNatTableSize, sizeof(struct sctpNatTableG));
655 la->sctpNatTimer.TimerQ = sn_calloc(SN_TIMER_QUEUE_SIZE, sizeof(struct sctpTimerQ));
656 /* Initialise hash table */
657 for (i = 0; i < la->sctpNatTableSize; i++) {
658     LIST_INIT(&la->sctpTableLocal[i]);
659     LIST_INIT(&la->sctpTableGlobal[i]);
660 }
661
662 /* Initialise circular timer Q*/
663 for (i = 0; i < SN_TIMER_QUEUE_SIZE; i++)
664     LIST_INIT(&la->sctpNatTimer.TimerQ[i]);
665 #ifdef _KERNEL
666 la->sctpNatTimer.loc_time=time_uptime; /* la->timeStamp is not set yet */
667 #else
668 la->sctpNatTimer.loc_time=la->timeStamp;
669 #endif
670 la->sctpNatTimer.cur_loc = 0;
671 la->sctpLinkCount = 0;
672 }
673

```

7.1.3.3 void AliasSctpTerm (struct libalias * *la*)

Cleans-up the SCTP NAT Implementation prior to unloading.

Removes all entries from the timer queue, freeing associations as it goes. We then free memory allocated to the look-up tables and the time queue

NOTE: We do not need to traverse the look-up tables as each association will always have an entry in the timer queue, freeing this memory once will free all memory allocated to entries in the look-up tables

Parameters:

la Pointer to the relevant libalias instance

Definition at line 689 of file alias_sctp.c.

```

694                                     {
695     assoc1 = LIST_FIRST(&la->sctpNatTimer.TimerQ[i]);
696     while (assoc1 != NULL) {
697         freeGlobalAddressList(assoc1);
698         assoc2 = LIST_NEXT(assoc1, timer_Q);
699         sn_free(assoc1);
700         assoc1 = assoc2;
701     }
702 }
703
704 sn_free(la->sctpTableLocal);
705 sn_free(la->sctpTableGlobal);
706 sn_free(la->sctpNatTimer.TimerQ);
707 }
708

```

7.1.3.4 int SctpAlias (struct libalias * *la*, struct ip * *pip*, int *direction*)

Handles SCTP packets passed from libalias.

This function needs to actually NAT/drop packets and possibly create and send AbortM or ErrorM packets in response. The process involves:

- Validating the direction parameter passed by the caller
- Checking and handling any expired timers for the NAT
- Calling [sctp_PktParser\(\)](#) to parse the packet
- Call [ProcessSctpMsg\(\)](#) to decide the appropriate outcome and to update the NAT tables
- Based on the return code either:
 - NAT the packet
 - Construct and send an ErrorM|AbortM packet
 - Mark the association for removal from the tables
- Potentially remove the association from all lookup tables
- Return the appropriate result to libalias

Parameters:

la Pointer to the relevant libalias instance

pip Pointer to IP packet to process

direction SN_TO_LOCAL | SN_TO_GLOBAL

Returns:

PKT_ALIAS_OK | PKT_ALIAS_IGNORE | PKT_ALIAS_ERROR

Definition at line 736 of file alias_sctp.c.

```

739                                     {
740     SctpAliasLog("ERROR: Invalid direction\n");
741     return (PKT_ALIAS_ERROR);
742 }
743

```

```

744 sctp_CheckTimers(la); /* Check timers */
745
746 /* Parse the packet */
747 rtnval = sctp_PktParser(la, direction, pip, &msg, &assoc); //using *char (change to mbuf when get co
748 switch (rtnval) {
749 case SN_PARSE_OK:
750     break;
751 case SN_PARSE_ERROR_CHHL:
752     /* Not an error if there is a chunk length parsing error and this is a fragmented packet */
753     if (ntohs(pip->ip_off) & IP_MF) {
754         rtnval = SN_PARSE_OK;
755         break;
756     }
757     SN_LOG(SN_LOG_EVENT,
758           logsctperror("SN_PARSE_ERROR", msg.sctp_hdr->v_tag, rtnval, direction));
759     return(PKT_ALIAS_ERROR);
760 case SN_PARSE_ERROR_PARTIALLOOKUP:
761     if (sysctl_error_on_ootb > SN_LOCALandPARTIAL_ERROR_ON_OOTB) {
762         SN_LOG(SN_LOG_EVENT,
763               logsctperror("SN_PARSE_ERROR", msg.sctp_hdr->v_tag, rtnval, direction));
764         return(PKT_ALIAS_ERROR);
765     }
766 case SN_PARSE_ERROR_LOOKUP:
767     if (sysctl_error_on_ootb == SN_ERROR_ON_OOTB ||
768         (sysctl_error_on_ootb == SN_LOCALandPARTIAL_ERROR_ON_OOTB && direction == SN_TO_LOCAL) ||
769         (sysctl_error_on_ootb == SN_LOCAL_ERROR_ON_OOTB && direction == SN_TO_GLOBAL)) {
770         TxAbortErrorM(la, &msg, assoc, SN_REFLECT_ERROR, direction); /*NB assoc=NULL */
771         return(PKT_ALIAS_RESPOND);
772     }
773 default:
774     SN_LOG(SN_LOG_EVENT,
775           logsctperror("SN_PARSE_ERROR", msg.sctp_hdr->v_tag, rtnval, direction));
776     return(PKT_ALIAS_ERROR);
777 }
778
779 SN_LOG(SN_LOG_DETAIL,
780       logsctpassoc(assoc, "*");
781       logsctpparse(direction, &msg);
782       );
783
784 /* Process the SCTP message */
785 rtnval = ProcessSctpMsg(la, direction, &msg, assoc);
786
787 SN_LOG(SN_LOG_DEBUG_MAX,
788       logsctpassoc(assoc, "-");
789       logSctpLocal(la);
790       logSctpGlobal(la);
791       );
792 SN_LOG(SN_LOG_DEBUG, logTimerQ(la));
793
794 switch(rtnval){
795 case SN_NAT_PKT:
796     switch(direction) {
797     case SN_TO_LOCAL:
798         DifferentialChecksum(&(msg.ip_hdr->ip_sum),
799                             &(assoc->l_addr), &(msg.ip_hdr->ip_dst), 2);
800         msg.ip_hdr->ip_dst = assoc->l_addr; /* change dst address to local address*/
801         break;
802     case SN_TO_GLOBAL:
803         DifferentialChecksum(&(msg.ip_hdr->ip_sum),
804                             &(assoc->a_addr), &(msg.ip_hdr->ip_src), 2);
805         msg.ip_hdr->ip_src = assoc->a_addr; /* change src to alias addr*/
806         break;
807     default:
808         rtnval = SN_DROP_PKT; /* shouldn't get here, but if it does drop packet */
809         SN_LOG(SN_LOG_LOW, logsctperror("ERROR: Invalid direction", msg.sctp_hdr->v_tag, rtnval, directi
810         break;

```

```

811     }
812     break;
813 case SN_DROP_PKT:
814     SN_LOG(SN_LOG_DETAIL, logsctperror("SN_DROP_PKT", msg.sctp_hdr->v_tag, rtnval, direction));
815     break;
816 case SN_REPLY_ABORT:
817 case SN_REPLY_ERROR:
818 case SN_SEND_ABORT:
819     TxAbortErrorM(la, &msg, assoc, rtnval, direction);
820     break;
821 default:
822     // big error, remove association and go to idle and write log messages
823     SN_LOG(SN_LOG_LOW, logsctperror("SN_PROCESSING_ERROR", msg.sctp_hdr->v_tag, rtnval, direction));
824     assoc->state=SN_RM; /* Mark for removal*/
825     break;
826 }
827
828 /* Remove association if tagged for removal */
829 if (assoc->state == SN_RM) {
830     if (assoc->TableRegister) {
831         sctp_RmTimeOut(la, assoc);
832         RmSctpAssoc(la, assoc);
833     }
834     LIBALIAS_LOCK_ASSERT(la);
835     freeGlobalAddressList(assoc);
836     sn_free(assoc);
837 }
838 switch(rtnval) {
839 case SN_NAT_PKT:
840     return(PKT_ALIAS_OK);
841 case SN_SEND_ABORT:
842     return(PKT_ALIAS_OK);
843 case SN_REPLY_ABORT:
844 case SN_REPLY_ERROR:
845 case SN_REFLECT_ERROR:
846     return(PKT_ALIAS_RESPOND);
847 case SN_DROP_PKT:
848 default:
849     return(PKT_ALIAS_ERROR);
850 }
851 }
852

```

7.1.3.5 static void TxAbortErrorM (struct libalias * la, struct sctp_nat_msg * sm, struct sctp_nat_assoc * assoc, int sndrply, int direction) [static]

Send an AbortM or ErrorM.

We construct the new SCTP packet to send in place of the existing packet we have been asked to NAT. This function can only be called if the original packet was successfully parsed as a valid SCTP packet.

An AbortM (without cause) packet is the smallest SCTP packet available and as such there is always space in the existing packet buffer to fit the AbortM packet. An ErrorM packet is 4 bytes longer than the (the error cause is not optional). An ErrorM is sent in response to an AddIP when the Vtag/address combination, if added, will produce a conflict in the association look up tables. It may also be used for an unexpected packet - a packet with no matching association in the NAT table and we are requesting an AddIP so we can add it. The smallest valid SCTP packet while the association is in an up-state is a Heartbeat packet, which is big enough to be transformed to an ErrorM.

We create a temporary character array to store the packet as we are constructing it. We then populate the array with appropriate values based on:

- Packet type (AbortM | ErrorM)

- Initial packet direction (SN_TO_LOCAL | SN_TO_GLOBAL)
- NAT response (Send packet | Reply packet)

Once complete, we copy the contents of the temporary packet over the original SCTP packet we were asked to NAT

Parameters:

la Pointer to the relevant libalias instance

sm Pointer to sctp message information

assoc Pointer to current association details

sndrply SN_SEND_ABORT | SN_REPLY_ABORT | SN_REPLY_ERROR

direction SN_TO_LOCAL | SN_TO_GLOBAL

Definition at line 890 of file alias_sctp.c.

```

894                                     { /* short packet, cannot send error cause */
895     include_error_cause = 0;
896     ip_size = ip_size - sizeof(struct sctp_error_cause);
897     sctp_size = sctp_size - sizeof(struct sctp_error_cause);
898 }
899 /* Assign header pointers packet */
900 struct ip* ip = (struct ip *) tmp_ip;
901 struct sctphdr* sctp_hdr = (struct sctphdr *) ((char *) ip + sizeof(*ip));
902 struct sctp_chunkhdr* chunk_hdr = (struct sctp_chunkhdr *) ((char *) sctp_hdr + sizeof(*sctp_hdr));
903 struct sctp_error_cause* error_cause = (struct sctp_error_cause *) ((char *) chunk_hdr + sizeof(*chu
904
905 /* construct ip header */
906 ip->ip_v = sm->ip_hdr->ip_v;
907 ip->ip_hl = 5; /* 5*32 bit words */
908 ip->ip_tos = 0;
909 ip->ip_len = htons(ip_size);
910 ip->ip_id = sm->ip_hdr->ip_id;
911 ip->ip_off = 0;
912 ip->ip_ttl = 255;
913 ip->ip_p = IPPROTO_SCTP;
914 /*
915 The definitions below should be removed when they make it into the SCTP stack
916 */
917 #define SCTP_MIDDLEBOX_FLAG 0x02
918 #define SCTP_NAT_TABLE_COLLISION 0x00b0
919 #define SCTP_MISSING_NAT 0x00b1
920 chunk_hdr->chunk_type = (sndrply & SN_TX_ABORT) ? SCTP_ABORT_ASSOCIATION : SCTP_OPERATION_ERROR;
921 chunk_hdr->chunk_flags = SCTP_MIDDLEBOX_FLAG;
922 if (include_error_cause) {
923     error_cause->code = htons((sndrply & SN_REFLECT_ERROR) ? SCTP_MISSING_NAT : SCTP_NAT_TABLE_COLLISION);
924     error_cause->length = htons(sizeof(struct sctp_error_cause));
925     chunk_hdr->chunk_length = htons(sizeof(*chunk_hdr) + sizeof(struct sctp_error_cause));
926 } else {
927     chunk_hdr->chunk_length = htons(sizeof(*chunk_hdr));
928 }
929
930 /* set specific values */
931 switch(sndrply) {
932 case SN_REFLECT_ERROR:
933     chunk_hdr->chunk_flags |= SCTP_HAD_NO_TCB; /* set Tbit */
934     sctp_hdr->v_tag = sm->sctp_hdr->v_tag;
935     break;
936 case SN_REPLY_ERROR:
937     sctp_hdr->v_tag = (direction == SN_TO_LOCAL) ? assoc->g_vtag : assoc->l_vtag ;
938     break;

```

```

939 case SN_SEND_ABORT:
940     sctp_hdr->v_tag = sm->sctp_hdr->v_tag;
941     break;
942 case SN_REPLY_ABORT:
943     sctp_hdr->v_tag = sm->sctpchnk.Init->initiate_tag;
944     break;
945 }
946
947 /* Set send/reply values */
948 if (sndrply == SN_SEND_ABORT) { /*pass through NAT */
949     ip->ip_src = (direction == SN_TO_LOCAL) ? sm->ip_hdr->ip_src : assoc->a_addr;
950     ip->ip_dst = (direction == SN_TO_LOCAL) ? assoc->l_addr : sm->ip_hdr->ip_dst;
951     sctp_hdr->src_port = sm->sctp_hdr->src_port;
952     sctp_hdr->dest_port = sm->sctp_hdr->dest_port;
953 } else { /* reply and reflect */
954     ip->ip_src = sm->ip_hdr->ip_dst;
955     ip->ip_dst = sm->ip_hdr->ip_src;
956     sctp_hdr->src_port = sm->sctp_hdr->dest_port;
957     sctp_hdr->dest_port = sm->sctp_hdr->src_port;
958 }
959
960 /* Calculate IP header checksum */
961 ip->ip_sum = in_cksum_hdr(ip);
962
963 /* calculate SCTP header CRC32 */
964 sctp_hdr->checksum = 0;
965 sctp_hdr->checksum = sctp_csum_finalize(update_crc32(0xffffffff, (unsigned char *) sctp_hdr, sctp_si
966
967 memcpy(sm->ip_hdr, ip, ip_size);
968
969 SN_LOG(SN_LOG_EVENT, SctpAliasLog("%s %s 0x%x (->%s:%u vtag=0x%x crc=0x%x)\n",
970     ((sndrply == SN_SEND_ABORT) ? "Sending" : "Replying"),
971     ((sndrply & SN_TX_ERROR) ? "ErrorM" : "AbortM"),
972     (include_error_cause ? ntohs(error_cause->code) : 0),
973     inet_ntoa(ip->ip_dst), ntohs(sctp_hdr->dest_port),
974     ntohl(sctp_hdr->v_tag), ntohl(sctp_hdr->checksum)));
975 }
976
977 /* -----
978 *

```

PACKET PARSER CODE

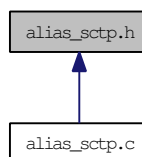
7.2 alias_sctp.h File Reference

```
#include <sys/param.h>
#include <sys/types.h>
#include <sys/queue.h>
#include <sys/time.h>
#include <netinet/in_system.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <machine/cpufunc.h>
#include <machine/cpu.h>
#include <netinet/sctp.h>
#include <netinet/sctp_header.h>
#include <stdlib.h>
#include <stdio.h>
#include <curses.h>
```

Include dependency graph for alias_sctp.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct [sctp_nat_assoc](#)
sctp association information
- struct [sctp_GlobalAddress](#)
- union [sctpChunkOfInt](#)
SCTP chunk of interest.
- struct [sctp_nat_msg](#)
SCTP message.
- struct [sctp_nat_timer](#)

sctp nat timer queue structure

Defines

- #define `SCTP_PACKED` `__attribute__((packed))`
- #define `SCTP_UNUSED` `__attribute__((unused))`
- #define `LINK_SCTP` `IPPROTO_SCTP`
- #define `SN_TO_LOCAL` `0`
- #define `SN_TO_GLOBAL` `1`
- #define `SN_TO_NODIR` `99`
- #define `SN_NAT_PKT` `0x0000`
- #define `SN_DROP_PKT` `0x0001`
- #define `SN_PROCESSING_ERROR` `0x0003`
- #define `SN_REPLY_ABORT` `0x0010`
- #define `SN_SEND_ABORT` `0x0020`
- #define `SN_TX_ABORT` `0x0030`
- #define `SN_REFLECT_ERROR` `0x0100`
- #define `SN_REPLY_ERROR` `0x0200`
- #define `SN_TX_ERROR` `0x0300`
- #define `PKT_ALIAS_RESPOND` `0x1000`

7.2.1 Detailed Description

Copyright (c) 2008, Centre for Advanced Internet Architectures Swinburne University of Technology, Melbourne, Australia (CRICOS number 00111D).

`Alias_sctp` forms part of the `libalias` kernel module to handle Network Address Translation (NAT) for the SCTP protocol.

This software was developed by David A. Hayes with leadership and advice from Jason But

Development is part of the CAIA SONATA project, proposed by Jason But and Grenville Armitage: <http://caia.swin.edu.au/urp/sonata/>

This project has been made possible in part by a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met: 1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. 2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution. 3. The names of the authors, the "Centre for Advanced Internet Architectures" and "Swinburne University of Technology" may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHORS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE AUTHORS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND

ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Definition in file [alias_sctp.h](#).

7.2.2 Define Documentation

7.2.2.1 #define LINK_SCTP IPPROTO_SCTP

Definition at line 105 of file [alias_sctp.h](#).

7.2.2.2 #define PKT_ALIAS_RESPOND 0x1000

Signal to libalias that there is a response packet to send

Definition at line 123 of file [alias_sctp.h](#).

7.2.2.3 #define SCTP_PACKED __attribute__((packed))

These are defined in [sctp_os_bsd.h](#), but it can't be included due to its local file inclusion, so I'm defining them here.

Definition at line 87 of file [alias_sctp.h](#).

7.2.2.4 #define SCTP_UNUSED __attribute__((unused))

Definition at line 90 of file [alias_sctp.h](#).

7.2.2.5 #define SN_DROP_PKT 0x0001

drop packet (don't forward it)

Definition at line 113 of file [alias_sctp.h](#).

7.2.2.6 #define SN_NAT_PKT 0x0000

Network Address Translate packet

Definition at line 112 of file [alias_sctp.h](#).

Referenced by [ProcessSctpMsg\(\)](#).

7.2.2.7 #define SN_PROCESSING_ERROR 0x0003

Packet processing error

Definition at line 114 of file [alias_sctp.h](#).

7.2.2.8 #define SN_REFLECT_ERROR 0x0100

Reply with ERROR to sender on OOTB packet Tbit set

Definition at line 118 of file alias_sctp.h.

7.2.2.9 #define SN_REPLY_ABORT 0x0010

Reply with ABORT to sender (don't forward it)

Definition at line 115 of file alias_sctp.h.

7.2.2.10 #define SN_REPLY_ERROR 0x0200

Reply with ERROR to sender on ASCONF clash

Definition at line 119 of file alias_sctp.h.

7.2.2.11 #define SN_SEND_ABORT 0x0020

Send ABORT to destination

Definition at line 116 of file alias_sctp.h.

7.2.2.12 #define SN_TO_GLOBAL 1

packet traveling from local to global

Definition at line 109 of file alias_sctp.h.

Referenced by logscpperror(), and logscpparse().

7.2.2.13 #define SN_TO_LOCAL 0

packet traveling from global to local

Definition at line 108 of file alias_sctp.h.

Referenced by logscpperror(), logscpparse(), and sctp_PktParser().

7.2.2.14 #define SN_TO_NODIR 99

used where direction is not important

Definition at line 110 of file alias_sctp.h.

Referenced by RmSctpAssoc().

7.2.2.15 #define SN_TX_ABORT 0x0030

mask for transmitting abort

Definition at line 117 of file alias_sctp.h.

7.2.2.16 #define SN_TX_ERROR 0x0300

mask for transmitting error

Definition at line 120 of file alias_sctp.h.

Index

- a_addr
 - sctp_nat_assoc, 65
- Add_Global_Address_to_List
 - alias_sctp.c, 78
- AddGlobalIPAddresses
 - packet_parser, 16
- AddSctpAssocGlobal
 - Hash, 43
- AddSctpAssocLocal
 - Hash, 44
- alias_sctp.c, 71
 - Add_Global_Address_to_List, 78
 - AliasSctpInit, 79
 - AliasSctpTerm, 79
 - SCTP_MIDDLEBOX_FLAG, 78
 - SCTP_MISSING_NAT, 78
 - SCTP_NAT_TABLE_COLLISION, 78
 - SCTP_VTAG_PARAM, 78
 - SctpAlias, 80
 - sn_calloc, 78
 - sn_free, 78
 - sn_malloc, 78
 - TxAbortErrorM, 82
- alias_sctp.h, 85
 - LINK_SCTP, 87
 - PKT_ALIAS_RESPOND, 87
 - SCTP_PACKED, 87
 - SCTP_UNUSED, 87
 - SN_DROP_PKT, 87
 - SN_NAT_PKT, 87
 - SN_PROCESSING_ERROR, 87
 - SN_REFLECT_ERROR, 87
 - SN_REPLY_ABORT, 88
 - SN_REPLY_ERROR, 88
 - SN_SEND_ABORT, 88
 - SN_TO_GLOBAL, 88
 - SN_TO_LOCAL, 88
 - SN_TO_NODIR, 88
 - SN_TX_ABORT, 88
 - SN_TX_ERROR, 88
- AliasSctpInit
 - alias_sctp.c, 79
- AliasSctpTerm
 - alias_sctp.c, 79
- Asconf
 - sctpChunkOfInt, 70
- chunk_length
 - sctp_nat_msg, 67
- CL_process
 - state_machine, 28
- cur_loc
 - sctp_nat_timer, 69
- exp
 - sctp_nat_assoc, 65
- exp_loc
 - sctp_nat_assoc, 65
- external
 - FindSctpRedirectAddress, 9
 - SctpShowAliasStats, 10
- External code changes and modifications, 9
- FindSctpGlobal
 - Hash, 45
- FindSctpGlobalClash
 - Hash, 46
- FindSctpGlobalT
 - Hash, 47
- FindSctpLocal
 - Hash, 48
- FindSctpLocalT
 - Hash, 48
- FindSctpRedirectAddress
 - external, 9
- freeGlobalAddressList
 - Hash, 49
- g_addr
 - sctp_GlobalAddress, 63
- g_port
 - sctp_nat_assoc, 65
- g_vtag
 - sctp_nat_assoc, 65
- GetAsconfVtags
 - packet_parser, 18
- Hash
 - AddSctpAssocGlobal, 43
 - AddSctpAssocLocal, 44
 - FindSctpGlobal, 45

- FindSctpGlobalClash, 46
- FindSctpGlobalT, 47
- FindSctpLocal, 48
- FindSctpLocalT, 48
- freeGlobalAddressList, 49
- RmSctpAssoc, 50
- SN_ADD_CLASH, 42
- SN_ADD_OK, 42
- SN_BOTH_TBL, 42
- SN_DEFAULT_HASH_SIZE, 42
- SN_GLOBAL_TBL, 42
- SN_LOCAL_TBL, 42
- SN_MAX_GLOBAL_ADDRESSES, 42
- SN_MAX_HASH_SIZE, 43
- SN_MIN_HASH_SIZE, 43
- SN_NULL_TBL, 43
- SN_TABLE_HASH, 43
- SN_WAIT_TOGLOBAL, 43
- SN_WAIT_TOLOCAL, 43
- Hash Table Macros and Functions, 41
- ID_process
 - state_machine, 29
- INa_process
 - state_machine, 30
- INi_process
 - state_machine, 31
- Init
 - sctpChunkOfInt, 70
- InitAck
 - sctpChunkOfInt, 70
- ip_hdr
 - sctp_nat_msg, 67
- IsADDorDEL
 - packet_parser, 19
- IsASCONFack
 - packet_parser, 20
- l_addr
 - sctp_nat_assoc, 65
- l_port
 - sctp_nat_assoc, 65
- l_vtag
 - sctp_nat_assoc, 66
- LINK_SCTP
 - alias_sctp.h, 87
- LIST_ENTRY
 - sctp_GlobalAddress, 63
 - sctp_nat_assoc, 64
- LIST_HEAD
 - sctp_nat_assoc, 64
 - sctp_nat_timer, 69
- loc_time
 - sctp_nat_timer, 69
- Logging
 - logsctpassoc, 36
 - logsctperror, 37
 - logSctpGlobal, 37
 - logSctpLocal, 38
 - logsctpparse, 38
 - logTimerQ, 39
 - SctpAliasLog, 39
 - SN_LOG, 35
 - SN_LOG_DEBUG, 36
 - SN_LOG_DEBUG_MAX, 36
 - SN_LOG_DETAIL, 36
 - SN_LOG_EVENT, 36
 - SN_LOG_INFO, 36
 - SN_LOG_LOW, 36
- Logging Functionality, 35
- logsctpassoc
 - Logging, 36
- logsctperror
 - Logging, 37
- logSctpGlobal
 - Logging, 37
- logSctpLocal
 - Logging, 38
- logsctpparse
 - Logging, 38
- logTimerQ
 - Logging, 39
- msg
 - sctp_nat_msg, 67
- num_Gaddr
 - sctp_nat_assoc, 66
- packet_parser
 - AddGlobalIPAddresses, 16
 - GetAsconfVtags, 18
 - IsADDorDEL, 19
 - IsASCONFack, 20
 - RmGlobalIPAddresses, 20
 - sctp_PktParser, 22
 - SN_ASCONFACK_PARAM_SIZE, 12
 - SN_MIN_CHUNK_SIZE, 12
 - SN_MIN_PARAM_SIZE, 12
 - SN_PARSE_ERROR_AS_MALLOC, 12
 - SN_PARSE_ERROR_CHHL, 12
 - SN_PARSE_ERROR_CHUNK, 13
 - SN_PARSE_ERROR_DIR, 13
 - SN_PARSE_ERROR_IPSHL, 13
 - SN_PARSE_ERROR_LOOKUP, 13
 - SN_PARSE_ERROR_LOOKUP_ABORT, 13
 - SN_PARSE_ERROR_PARTIALLOOKUP, 13
 - SN_PARSE_ERROR_PORT, 13

- SN_PARSE_ERROR_VTAG, 14
- SN_PARSE_OK, 14
- SN_SCTP_ABORT, 14
- SN_SCTP_ASCONF, 14
- SN_SCTP_ASCONFACK, 14
- SN_SCTP_FIRSTCHUNK, 14
- SN_SCTP_INIT, 14
- SN_SCTP_INITACK, 15
- SN_SCTP_NEXTCHUNK, 15
- SN_SCTP_NEXTPARAM, 15
- SN_SCTP_OTHER, 15
- SN_SCTP_SHUTACK, 15
- SN_SCTP_SHUTCOMP, 15
- SN_VTAG_PARAM_SIZE, 15
- PKT_ALIAS_RESPOND
 - alias_sctp.h, 87
- ProcessSctpMsg
 - state_machine, 32
- RmGlobalIPAddresses
 - packet_parser, 20
- RmSctpAssoc
 - Hash, 50
- SCTP NAT State Machine, 27
- SCTP Packet Parsing, 11
- sctp_AddTimeOut
 - Timer, 53
- sctp_CheckTimers
 - Timer, 53
- sctp_GlobalAddress, 63
 - g_addr, 63
 - LIST_ENTRY, 63
- sctp_hdr
 - sctp_nat_msg, 68
- SCTP_MIDDLEBOX_FLAG
 - alias_sctp.c, 78
- SCTP_MISSING_NAT
 - alias_sctp.c, 78
- sctp_nat_assoc, 64
 - a_addr, 65
 - exp, 65
 - exp_loc, 65
 - g_port, 65
 - g_vtag, 65
 - l_addr, 65
 - l_port, 65
 - l_vtag, 66
 - LIST_ENTRY, 64
 - LIST_HEAD, 64
 - num_Gaddr, 66
 - state, 66
 - TableRegister, 66
- sctp_nat_msg, 67
 - chunk_length, 67
 - ip_hdr, 67
 - msg, 67
 - sctp_hdr, 68
 - sctpchnk, 68
- SCTP_NAT_TABLE_COLLISION
 - alias_sctp.c, 78
- sctp_nat_timer, 69
 - cur_loc, 69
 - LIST_HEAD, 69
 - loc_time, 69
- SCTP_PACKED
 - alias_sctp.h, 87
- sctp_PktParser
 - packet_parser, 22
- sctp_ResetTimeOut
 - Timer, 54
- sctp_RmTimeOut
 - Timer, 55
- SCTP_UNUSED
 - alias_sctp.h, 87
- SCTP_VTAG_PARAM
 - alias_sctp.c, 78
- SctpAlias
 - alias_sctp.c, 80
- SctpAliasLog
 - Logging, 39
- sctpchnk
 - sctp_nat_msg, 68
- sctpChunkOfInt, 70
 - Asconf, 70
 - Init, 70
 - InitAck, 70
- SctpShowAliasStats
 - external, 10
- SN_ADD_CLASH
 - Hash, 42
- SN_ADD_OK
 - Hash, 42
- SN_ASCONFACK_PARAM_SIZE
 - packet_parser, 12
- SN_BOTH_TBL
 - Hash, 42
- SN_C_T
 - Timer, 52
- sn_calloc
 - alias_sctp.c, 78
- SN_CL
 - state_machine, 28
- SN_DEFAULT_HASH_SIZE
 - Hash, 42
- SN_DROP_PKT
 - alias_sctp.h, 87
- SN_ERROR_ON_OOTB

- sysctl, 57
- sn_free
 - alias_sctp.c, 78
- SN_GLOBAL_TBL
 - Hash, 42
- SN_I_T
 - Timer, 52
- SN_ID
 - state_machine, 28
- SN_INa
 - state_machine, 28
- SN_INi
 - state_machine, 28
- SN_LOCAL_ERROR_ON_OOTB
 - sysctl, 57
- SN_LOCAL_TBL
 - Hash, 42
- SN_LOCALandPARTIAL_ERROR_ON_OOTB
 - sysctl, 57
- SN_LOG
 - Logging, 35
- SN_LOG_DEBUG
 - Logging, 36
- SN_LOG_DEBUG_MAX
 - Logging, 36
- SN_LOG_DETAIL
 - Logging, 36
- SN_LOG_EVENT
 - Logging, 36
- SN_LOG_INFO
 - Logging, 36
- SN_LOG_LOW
 - Logging, 36
- sn_malloc
 - alias_sctp.c, 78
- SN_MAX_GLOBAL_ADDRESSES
 - Hash, 42
- SN_MAX_HASH_SIZE
 - Hash, 43
- SN_MAX_TIMER
 - Timer, 53
- SN_MIN_CHUNK_SIZE
 - packet_parser, 12
- SN_MIN_HASH_SIZE
 - Hash, 43
- SN_MIN_PARAM_SIZE
 - packet_parser, 12
- SN_MIN_TIMER
 - Timer, 53
- SN_NAT_PKT
 - alias_sctp.h, 87
- SN_NO_ERROR_ON_OOTB
 - sysctl, 58
- SN_NULL_TBL
 - Hash, 43
- SN_PARSE_ERROR_AS_MALLOC
 - packet_parser, 12
- SN_PARSE_ERROR_CHHL
 - packet_parser, 12
- SN_PARSE_ERROR_CHUNK
 - packet_parser, 13
- SN_PARSE_ERROR_DIR
 - packet_parser, 13
- SN_PARSE_ERROR_IPSHL
 - packet_parser, 13
- SN_PARSE_ERROR_LOOKUP
 - packet_parser, 13
- SN_PARSE_ERROR_LOOKUP_ABORT
 - packet_parser, 13
- SN_PARSE_ERROR_PARTIALLOOKUP
 - packet_parser, 13
- SN_PARSE_ERROR_PORT
 - packet_parser, 13
- SN_PARSE_ERROR_VTAG
 - packet_parser, 14
- SN_PARSE_OK
 - packet_parser, 14
- SN_PROCESSING_ERROR
 - alias_sctp.h, 87
- SN_REFLECT_ERROR
 - alias_sctp.h, 87
- SN_REPLY_ABORT
 - alias_sctp.h, 88
- SN_REPLY_ERROR
 - alias_sctp.h, 88
- SN_RM
 - state_machine, 28
- SN_SCTP_ABORT
 - packet_parser, 14
- SN_SCTP_ASCONF
 - packet_parser, 14
- SN_SCTP_ASCONFACK
 - packet_parser, 14
- SN_SCTP_FIRSTCHUNK
 - packet_parser, 14
- SN_SCTP_INIT
 - packet_parser, 14
- SN_SCTP_INITACK
 - packet_parser, 15
- SN_SCTP_NEXTCHUNK
 - packet_parser, 15
- SN_SCTP_NEXTPARAM
 - packet_parser, 15
- SN_SCTP_OTHER
 - packet_parser, 15
- SN_SCTP_SHUTACK
 - packet_parser, 15
- SN_SCTP_SHUTCOMP

- packet_parser, 15
- SN_SEND_ABORT
 - alias_sctp.h, 88
- SN_TABLE_HASH
 - Hash, 43
- SN_TIMER_QUEUE_SIZE
 - Timer, 53
- SN_TO_GLOBAL
 - alias_sctp.h, 88
- SN_TO_LOCAL
 - alias_sctp.h, 88
- SN_TO_NODIR
 - alias_sctp.h, 88
- SN_TX_ABORT
 - alias_sctp.h, 88
- SN_TX_ERROR
 - alias_sctp.h, 88
- SN_U_T
 - Timer, 53
- SN_UP
 - state_machine, 28
- SN_VTAG_PARAM_SIZE
 - packet_parser, 15
- SN_WAIT_TOGLOBAL
 - Hash, 43
- SN_WAIT_TOLOCAL
 - Hash, 43
- SN_X_T
 - Timer, 53
- state
 - sctp_nat_assoc, 66
- state_machine
 - CL_process, 28
 - ID_process, 29
 - INa_process, 30
 - INi_process, 31
 - ProcessSctpMsg, 32
 - SN_CL, 28
 - SN_ID, 28
 - SN_INa, 28
 - SN_INi, 28
 - SN_RM, 28
 - SN_UP, 28
 - UP_process, 33
- sysctl
 - SN_ERROR_ON_OOTB, 57
 - SN_LOCAL_ERROR_ON_OOTB, 57
 - SN_LOCALandPARTIAL_ERROR_ON_OOTB, 57
 - SN_NO_ERROR_ON_OOTB, 58
 - sysctl_accept_global_ootb_addip, 61
 - sysctl_chg_accept_global_ootb_addip, 58
 - sysctl_chg_chunk_proc_limit, 58
 - sysctl_chg_error_on_ootb, 58
 - sysctl_chg_hashtable_size, 59
 - sysctl_chg_initialising_chunk_proc_limit, 59
 - sysctl_chg_loglevel, 59
 - sysctl_chg_param_proc_limit, 60
 - sysctl_chg_timer, 60
 - sysctl_chg_track_global_addresses, 60
 - sysctl_chunk_proc_limit, 61
 - sysctl_error_on_ootb, 61
 - sysctl_hashtable_size, 61
 - sysctl_holddown_timer, 61
 - sysctl_init_timer, 61
 - sysctl_initialising_chunk_proc_limit, 61
 - sysctl_log_level, 62
 - sysctl_param_proc_limit, 62
 - sysctl_shutdown_timer, 62
 - sysctl_track_global_addresses, 62
 - sysctl_up_timer, 62
- SysCtl Variable and callback function declarations, 56
 - sysctl_accept_global_ootb_addip
 - sysctl, 61
 - sysctl_chg_accept_global_ootb_addip
 - sysctl, 58
 - sysctl_chg_chunk_proc_limit
 - sysctl, 58
 - sysctl_chg_error_on_ootb
 - sysctl, 58
 - sysctl_chg_hashtable_size
 - sysctl, 59
 - sysctl_chg_initialising_chunk_proc_limit
 - sysctl, 59
 - sysctl_chg_loglevel
 - sysctl, 59
 - sysctl_chg_param_proc_limit
 - sysctl, 60
 - sysctl_chg_timer
 - sysctl, 60
 - sysctl_chg_track_global_addresses
 - sysctl, 60
 - sysctl_chunk_proc_limit
 - sysctl, 61
 - sysctl_error_on_ootb
 - sysctl, 61
 - sysctl_hashtable_size
 - sysctl, 61
 - sysctl_holddown_timer
 - sysctl, 61
 - sysctl_init_timer
 - sysctl, 61
 - sysctl_initialising_chunk_proc_limit
 - sysctl, 61
 - sysctl_log_level
 - sysctl, 62
 - sysctl_param_proc_limit

- sysctl, [62](#)
- sysctl_shutdown_timer
 - sysctl, [62](#)
- sysctl_track_global_addresses
 - sysctl, [62](#)
- sysctl_up_timer
 - sysctl, [62](#)
- TableRegister
 - sctp_nat_assoc, [66](#)
- Timer
 - sctp_AddTimeOut, [53](#)
 - sctp_CheckTimers, [53](#)
 - sctp_ResetTimeOut, [54](#)
 - sctp_RmTimeOut, [55](#)
 - SN_C_T, [52](#)
 - SN_I_T, [52](#)
 - SN_MAX_TIMER, [53](#)
 - SN_MIN_TIMER, [53](#)
 - SN_TIMER_QUEUE_SIZE, [53](#)
 - SN_U_T, [53](#)
 - SN_X_T, [53](#)
- Timer Queue Macros and Functions, [52](#)
- TxAbortErrorM
 - alias_sctp.c, [82](#)
- UP_process
 - state_machine, [33](#)