sonata

Generated by Doxygen 1.5.7

Fri Nov 28 10:52:02 2008

Contents

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:

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

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 $\rightarrow$ 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 $\rightarrow$ 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.

```
                          {0};
  int bytes_left = 0;
  int param_size;
  int param_count, addr_param_count = 0;

  switch(direction) {
  case SN_TO_GLOBAL: /* does not contain global addresses */
    g_addr = sm->ip_hdr->ip_dst;
    bytes_left = 0; /* force exit */
    break;
  case SN_TO_LOCAL:
    g_addr = sm->ip_hdr->ip_src;
    param_count = 1;
    switch(sm->msg) {
    case SN_SCTP_INIT:
      bytes_left = sm->chunk_length - sizeof(struct sctp_init_chunk);
      param = (struct sctp_paramhdr *)((char *)sm->sctpchnk.Init + sizeof(struct sctp_init));
      break;
    case SN_SCTP_INITACK:
      bytes_left = sm->chunk_length - sizeof(struct sctp_init_ack_chunk);
      param = (struct sctp_paramhdr *)((char *)sm->sctpchnk.InitAck + sizeof(struct sctp_init_ack));
      break;
    case SN_SCTP_ASCONF:
      bytes_left = sm->chunk_length;
      param = sm->sctpchnk.Asconf;
      break;
    }
  }
  if (bytes_left >= SN_MIN_PARAM_SIZE)
    param_size = SCTP_SIZE32(ntohs(param->param_length));
  else
    param_size = bytes_left+1; /* force skip loop */

  if ((assoc->state == SN_ID) && ((sm->msg == SN_SCTP_INIT) || (bytes_left < SN_MIN_PARAM_SIZE))) {/*
    G_Addr = (struct sctp_GlobalAddress *) sn_malloc(sizeof(struct sctp_GlobalAddress));
    if (G_Addr == NULL) {/* out of resources */
      SN_LOG(SN_LOG_EVENT,
             logsctperror("AddGlobalIPAddress: No resources for adding global address - revert to no
                          sm->sctp_hdr->v_tag,  0, direction));
      assoc->num_Gaddr = 0; /* don't track any more for this assoc*/
      sysctl_track_global_addresses=0;
      return;
    }
    G_Addr->g_addr = g_addr;
    if (!Add_Global_Address_to_List(assoc, G_Addr))
      SN_LOG(SN_LOG_EVENT,
             logsctperror("AddGlobalIPAddress: Address already in list",
                          sm->sctp_hdr->v_tag,  assoc->num_Gaddr, direction));
  }

  /* step through parameters */
  while((bytes_left >= param_size) && (bytes_left >= sizeof(struct sctp_ipv4addr_param))) {
    if (assoc->num_Gaddr >= sysctl_track_global_addresses) {
      SN_LOG(SN_LOG_EVENT,
             logsctperror("AddGlobalIPAddress: Maximum Number of addresses reached",
                          sm->sctp_hdr->v_tag,  sysctl_track_global_addresses, direction));
      return;
    }
    switch(ntohs(param->param_type)) {
    case SCTP_ADD_IP_ADDRESS:
      /* skip to address parameter - leave param_size so bytes left will be calculated properly*/
      param = (struct sctp_paramhdr *) &((struct sctp_asconf_addrv4_param *) param)->addrp;
    case SCTP_IPV4_ADDRESS:
      ipv4_param = (struct sctp_ipv4addr_param *) param;
      /* add addresses to association */
      G_Addr = (struct sctp_GlobalAddress *) sn_malloc(sizeof(struct sctp_GlobalAddress));
      if (G_Addr == NULL) {/* out of resources */
        SN_LOG(SN_LOG_EVENT,
               logsctperror("AddGlobalIPAddress: No resources for adding global address - revert to r
                            sm->sctp_hdr->v_tag,  0, direction));
        assoc->num_Gaddr = 0; /* don't track any more for this assoc*/
        sysctl_track_global_addresses=0;
        return;
      }
      /* add address */
      addr_param_count++;
      if ((sm->msg == SN_SCTP_ASCONF) && (ipv4_param->addr == INADDR_ANY)) { /* use packet address */
        G_Addr->g_addr = g_addr;
        if (!Add_Global_Address_to_List(assoc, G_Addr))
          SN_LOG(SN_LOG_EVENT,
                 logsctperror("AddGlobalIPAddress: Address already in list",
                              sm->sctp_hdr->v_tag,  assoc->num_Gaddr, direction));
        return; /*shouldn't be any other addresses if the zero address is given*/
      } else {
        G_Addr->g_addr.s_addr = ipv4_param->addr;
        if (!Add_Global_Address_to_List(assoc, G_Addr))
          SN_LOG(SN_LOG_EVENT,
                 logsctperror("AddGlobalIPAddress: Address already in list",
                              sm->sctp_hdr->v_tag,  assoc->num_Gaddr, direction));
      }
    }

    bytes_left -= param_size;
    if (bytes_left < SN_MIN_PARAM_SIZE)
      break;

    param = SN_SCTP_NEXTPARAM(param);
    param_size = SCTP_SIZE32(ntohs(param->param_length));
    if (++param_count > sysctl_param_proc_limit) {
      SN_LOG(SN_LOG_EVENT,
             logsctperror("Parameter parse limit exceeded (AddGlobalIPAddress)",
                          sm->sctp_hdr->v_tag, sysctl_param_proc_limit, direction));
      break; /* limit exceeded*/
    }
  }
  if (addr_param_count == 0) {
    SN_LOG(SN_LOG_DETAIL,
           logsctperror("AddGlobalIPAddress: no address parameters to add",
                        sm->sctp_hdr->v_tag, assoc->num_Gaddr, direction));
  }
```

```
}

```

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.

```
                            {
    struct sctp_paramhdr ph;/* type=SCTP_VTAG_PARAM */
    uint32_t local_vtag;
    uint32_t remote_vtag;
  }                     __attribute__((packed));

  struct sctp_vtag_param *vtag_param;
  struct sctp_paramhdr *param;
  int bytes_left;
  int param_size;
  int param_count;

  param_count = 1;
  param = sm->sctpchnk.Asconf;
  param_size = SCTP_SIZE32(ntohs(param->param_length));
  bytes_left = sm->chunk_length;
  /* step through Asconf parameters */
  while((bytes_left >= param_size) && (bytes_left >= SN_VTAG_PARAM_SIZE)) {
    if (ntohs(param->param_type) == SCTP_VTAG_PARAM) {
      vtag_param = (struct sctp_vtag_param *) param;
      switch(direction) {
        /* The Internet draft is a little ambigious as to order of these vtags.
           We think it is this way around. If we are wrong, the order will need
           to be changed. */
      case SN_TO_GLOBAL:
        *g_vtag = vtag_param->local_vtag;
        *l_vtag = vtag_param->remote_vtag;
        break;
      case SN_TO_LOCAL:
        *g_vtag = vtag_param->remote_vtag;
        *l_vtag = vtag_param->local_vtag;
        break;
      }
      return(1); /* found */
    }

    bytes_left -= param_size;
    if (bytes_left < SN_MIN_PARAM_SIZE) return(0);

    param = SN_SCTP_NEXTPARAM(param);
    param_size = SCTP_SIZE32(ntohs(param->param_length));
    if (++param_count > sysctl_param_proc_limit) {
      SN_LOG(SN_LOG_EVENT,
             logsctperror("Parameter parse limit exceeded (GetAsconfVtags)",
                          sm->sctp_hdr->v_tag, sysctl_param_proc_limit, direction));
      return(0); /* not found limit exceeded*/
    }
  }
  return(0); /* not found */
}

```

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.

```
                                                {
    if (ntohs(param->param_type) == SCTP_ADD_IP_ADDRESS)
      return(SCTP_ADD_IP_ADDRESS);
    else if (ntohs(param->param_type) == SCTP_DEL_IP_ADDRESS)
      return(SCTP_DEL_IP_ADDRESS);
    /* check others just in case */
    bytes_left -= param_size;
    if (bytes_left >= SN_MIN_PARAM_SIZE) {
      param = SN_SCTP_NEXTPARAM(param);
    } else {
      return(0); /*Neither found */
    }
    param_size = SCTP_SIZE32(ntohs(param->param_length));
    if (bytes_left < param_size) return(0);

    if (++param_count > sysctl_param_proc_limit) {
      SN_LOG(SN_LOG_EVENT,
             logsctperror("Parameter parse limit exceeded IsADDorDEL)",
                          sm->sctp_hdr->v_tag, sysctl_param_proc_limit, direction));
      return(0); /* not found limit exceeded*/
    }
  }
  return(0);  /*Neither found */
}

/* ---------------------------------------------------------------------
 *                            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.

```
                                                {
    if (ntohs(param->param_type) == SCTP_SUCCESS_REPORT)
      return(1); /* success - but can't match correlation IDs - should only be one */
    /* check others just in case */
    bytes_left -= param_size;
    if (bytes_left >= SN_MIN_PARAM_SIZE) {
      param = SN_SCTP_NEXTPARAM(param);
    } else {
      return(0);
    }
    param_size = SCTP_SIZE32(ntohs(param->param_length));
    if (bytes_left < param_size) return(0);

    if (++param_count > sysctl_param_proc_limit) {
      SN_LOG(SN_LOG_EVENT,
             logsctperror("Parameter parse limit exceeded (IsASCONFack)",
                          sm->sctp_hdr->v_tag, sysctl_param_proc_limit, direction));
      return(0); /* not found limit exceeded*/
    }
  }
  return(0); /* not success */
}

```

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.

```
                                     {
  param_size = SCTP_SIZE32(ntohs(param->param_length));
} else {
  SN_LOG(SN_LOG_EVENT,
         logsctperror("RmGlobalIPAddress: truncated packet - cannot remove IP addresses",
                      sm->sctp_hdr->v_tag, sysctl_track_global_addresses, direction));
  return;
}

/* step through Asconf parameters */
while((bytes_left >= param_size) && (bytes_left >= sizeof(struct sctp_ipv4addr_param))) {
  if (ntohs(param->param_type) == SCTP_DEL_IP_ADDRESS) {
    asconf_ipv4_param = (struct sctp_asconf_addrv4_param *) param;
    if (asconf_ipv4_param->addrp.addr == INADDR_ANY) { /* remove all bar pkt address */
      LIST_FOREACH_SAFE(G_Addr, &(assoc->Gaddr), list_Gaddr, G_Addr_tmp) {
        if(G_Addr->g_addr.s_addr != sm->ip_hdr->ip_src.s_addr) {
          if (assoc->num_Gaddr > 1) { /* only delete if more than one */
            LIST_REMOVE(G_Addr, list_Gaddr);
            sn_free(G_Addr);
            assoc->num_Gaddr--;
          } else {
            SN_LOG(SN_LOG_EVENT,
                   logsctperror("RmGlobalIPAddress: Request to remove last IP address (didn't)",
                                sm->sctp_hdr->v_tag, assoc->num_Gaddr, direction));
          }
        }
      }
      return; /*shouldn't be any other addresses if the zero address is given*/
    } else {
      LIST_FOREACH_SAFE(G_Addr, &(assoc->Gaddr), list_Gaddr, G_Addr_tmp) {
        if(G_Addr->g_addr.s_addr == asconf_ipv4_param->addrp.addr) {
          if (assoc->num_Gaddr > 1) { /* only delete if more than one */
            LIST_REMOVE(G_Addr, list_Gaddr);
            sn_free(G_Addr);
            assoc->num_Gaddr--;
            break; /* Since add only adds new addresses, there should be no double entries */
          } else {
            SN_LOG(SN_LOG_EVENT,
                   logsctperror("RmGlobalIPAddress: Request to remove last IP address (didn't)",
                                sm->sctp_hdr->v_tag, assoc->num_Gaddr, direction));
          }
        }
      }
    }
  }
  bytes_left -= param_size;
  if (bytes_left == 0) return;
  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 addresse
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_∗

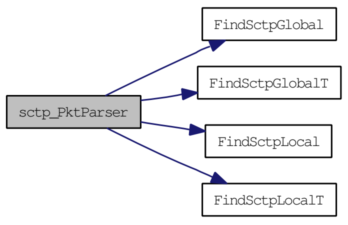
Definition at line 1006 of file alias_sctp.c.

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;
```

```
int bytes_left; /* bytes left in ip packet */
int chunk_length;
int chunk_count;
int partial_match = 0;
//  mbuf *mp;
//  int mlen;

//  mlen = SCTP_HEADER_LEN(i_pak);
//  mp = SCTP_HEADER_TO_CHAIN(i_pak); /* does nothing in bsd since header and chain not separate */

/*
 * Note, that if the VTag is zero, it must be an INIT
 * Also, I am only interested in the content of INIT and ADDIP chunks
 */

// no mbuf stuff from Paolo yet so ...
sm->ip_hdr = pip;
/* remove ip header length from the bytes_left */
bytes_left = ntohs(pip->ip_len) - (pip->ip_hl << 2);

/* Check SCTP header length and move to first chunk */
if (bytes_left < sizeof(struct sctphdr)) {
  sm->sctp_hdr = NULL;
  return(SN_PARSE_ERROR_IPSHL); /* packet not long enough*/
}

sm->sctp_hdr = sctp_hdr = (struct sctphdr *) ip_next(pip);
bytes_left -= sizeof(struct sctphdr);

/* Check for valid ports (zero valued ports would find partially initialised associations */
if (sctp_hdr->src_port == 0 || sctp_hdr->dest_port == 0)
  return(SN_PARSE_ERROR_PORT);

/* Check length of first chunk */
if (bytes_left < SN_MIN_CHUNK_SIZE) /* malformed chunk - could cause endless loop*/
  return(SN_PARSE_ERROR_CHHL); /* packet not long enough for this chunk */

/* First chunk */
chunk_hdr = SN_SCTP_FIRSTCHUNK(sctp_hdr);

chunk_length = SCTP_SIZE32(ntohs(chunk_hdr->chunk_length));
if ((chunk_length < SN_MIN_CHUNK_SIZE) || (chunk_length > bytes_left)) /* malformed chunk - could c
  return(SN_PARSE_ERROR_CHHL);

if ((chunk_hdr->chunk_flags & SCTP_HAD_NO_TCB) &&
    ((chunk_hdr->chunk_type == SCTP_ABORT_ASSOCIATION) ||
     (chunk_hdr->chunk_type == SCTP_SHUTDOWN_COMPLETE))) {
  /* T-Bit set */
  if (direction == SN_TO_LOCAL)
    *passoc = FindSctpGlobalT(la,  pip->ip_src, sctp_hdr->v_tag, sctp_hdr->dest_port, sctp_hdr->src
  else
    *passoc = FindSctpLocalT(la, pip->ip_dst, sctp_hdr->v_tag, sctp_hdr->dest_port, sctp_hdr->src_p
} else {
  /* Proper v_tag settings */
  if (direction == SN_TO_LOCAL)
    *passoc = FindSctpGlobal(la, pip->ip_src, sctp_hdr->v_tag, sctp_hdr->src_port, sctp_hdr->dest_p
  else
    *passoc = FindSctpLocal(la, pip->ip_src,  pip->ip_dst, sctp_hdr->v_tag, sctp_hdr->src_port, sct
}

chunk_count = 1;
/* Real packet parsing occurs below */
sm->msg = SN_SCTP_OTHER;/* Initialise to largest value*/
sm->chunk_length = 0; /* only care about length for key chunks */
while (IS_SCTP_CONTROL(chunk_hdr)) {
  switch(chunk_hdr->chunk_type) {
  case SCTP_INITIATION:
    if (chunk_length < sizeof(struct sctp_init_chunk)) /* malformed chunk*/
      return(SN_PARSE_ERROR_CHHL);
    sm->msg = SN_SCTP_INIT;
    sm->sctpchnk.Init = (struct sctp_init *) ((char *) chunk_hdr + sizeof(struct sctp_chunkhdr));
    sm->chunk_length = chunk_length;
    /* if no existing association, create a new one */
    if (*passoc == NULL) {
      if (sctp_hdr->v_tag == 0){ //Init requires vtag=0
        *passoc = (struct sctp_nat_assoc *) sn_malloc(sizeof(struct sctp_nat_assoc));
        if (*passoc == NULL) {/* out of resources */
          return(SN_PARSE_ERROR_AS_MALLOC);
        }
        /* Initialise association - malloc initialises memory to zeros */
        (*passoc)->state = SN_ID;
        LIST_INIT(&((*passoc)->Gaddr)); /* always initialise to avoid memory problems */
        (*passoc)->TableRegister = SN_NULL_TBL;
        return(SN_PARSE_OK);
      }
      return(SN_PARSE_ERROR_VTAG);
    }
    return(SN_PARSE_ERROR_LOOKUP);
  case SCTP_INITIATION_ACK:
    if (chunk_length < sizeof(struct sctp_init_ack_chunk)) /* malformed chunk*/
      return(SN_PARSE_ERROR_CHHL);
    sm->msg = SN_SCTP_INITACK;
    sm->sctpchnk.InitAck = (struct sctp_init_ack *) ((char *) chunk_hdr + sizeof(struct sctp_chunkh
    sm->chunk_length = chunk_length;
    return ((*passoc == NULL)?(SN_PARSE_ERROR_LOOKUP):(SN_PARSE_OK));
  case SCTP_ABORT_ASSOCIATION: /* access only minimum sized chunk */
    sm->msg = SN_SCTP_ABORT;
    sm->chunk_length = chunk_length;
    return ((*passoc == NULL)?(SN_PARSE_ERROR_LOOKUP_ABORT):(SN_PARSE_OK));
  case SCTP_SHUTDOWN_ACK:
   if (chunk_length < sizeof(struct sctp_shutdown_ack_chunk)) /* malformed chunk*/
      return(SN_PARSE_ERROR_CHHL);
    if (sm->msg > SN_SCTP_SHUTACK) {
      sm->msg = SN_SCTP_SHUTACK;
      sm->chunk_length = chunk_length;
    }
    break;
  case SCTP_SHUTDOWN_COMPLETE:  /* minimum sized chunk */
   if (sm->msg > SN_SCTP_SHUTCOMP) {
      sm->msg = SN_SCTP_SHUTCOMP;
      sm->chunk_length = chunk_length;
    }
    return ((*passoc == NULL)?(SN_PARSE_ERROR_LOOKUP):(SN_PARSE_OK));
  case SCTP_ASCONF:
    if (sm->msg > SN_SCTP_ASCONF) {
      if (chunk_length < (sizeof(struct  sctp_asconf_chunk) + sizeof(struct  sctp_ipv4addr_param)))
        return(SN_PARSE_ERROR_CHHL);
      //leave parameter searching to later, if required
      param_hdr = (struct sctp_paramhdr *) ((char *) chunk_hdr + sizeof(struct sctp_asconf_chunk));
      if (ntohs(param_hdr->param_type) == SCTP_IPV4_ADDRESS) {
        if ((*passoc == NULL) && (direction == SN_TO_LOCAL)) { /* AddIP with no association */
          /* try look up with the ASCONF packet's alternative address */
          ipv4addr.s_addr = ((struct sctp_ipv4addr_param *) param_hdr)->addr;
          *passoc = FindSctpGlobal(la, ipv4addr, sctp_hdr->v_tag, sctp_hdr->src_port, sctp_hdr->des
        }
        param_hdr = (struct sctp_paramhdr *)
          ((char *) param_hdr + sizeof(struct sctp_ipv4addr_param)); /*asconf's compulsory address
        sm->chunk_length = chunk_length - sizeof(struct  sctp_asconf_chunk) - sizeof(struct  sctp_i
      } else {
        if (chunk_length < (sizeof(struct  sctp_asconf_chunk) + sizeof(struct  sctp_ipv6addr_param)
          return(SN_PARSE_ERROR_CHHL);
        param_hdr = (struct sctp_paramhdr *)
          ((char *) param_hdr + sizeof(struct sctp_ipv6addr_param)); /*asconf's compulsory address
        sm->chunk_length = chunk_length - sizeof(struct  sctp_asconf_chunk) - sizeof(struct  sctp_i
      }
      sm->msg = SN_SCTP_ASCONF;
      sm->sctpchnk.Asconf = param_hdr;

      if (*passoc == NULL) { /* AddIP with no association */
        *passoc = (struct sctp_nat_assoc *) sn_malloc(sizeof(struct sctp_nat_assoc));
        if (*passoc == NULL) {/* out of resources */
          return(SN_PARSE_ERROR_AS_MALLOC);
        }
        /* Initialise association  - malloc initialises memory to zeros */
        (*passoc)->state = SN_ID;
        LIST_INIT(&((*passoc)->Gaddr)); /* always initialise to avoid memory problems */
        (*passoc)->TableRegister = SN_NULL_TBL;
        return(SN_PARSE_OK);
      }
    }
    break;
  case SCTP_ASCONF_ACK:
    if (sm->msg > SN_SCTP_ASCONFACK) {
      if (chunk_length < sizeof(struct  sctp_asconf_ack_chunk)) /* malformed chunk*/
        return(SN_PARSE_ERROR_CHHL);
      //leave parameter searching to later, if required
      param_hdr = (struct sctp_paramhdr *) ((char *) chunk_hdr
                                      + sizeof(struct sctp_asconf_ack_chunk));
      sm->msg = SN_SCTP_ASCONFACK;
      sm->sctpchnk.Asconf = param_hdr;
      sm->chunk_length = chunk_length - sizeof(struct sctp_asconf_ack_chunk);
    }
    break;
  default:
    break; /* do nothing*/
  }

  /* if no association is found exit - we need to find an Init or AddIP within sysctl_initialising_
  if ((*passoc == NULL) && (chunk_count >= sysctl_initialising_chunk_proc_limit))
    return(SN_PARSE_ERROR_LOOKUP);

  /* finished with this chunk, on to the next chunk*/
  bytes_left-= chunk_length;

  /* Is this the end of the packet ? */
  if (bytes_left == 0)
    return(SN_PARSE_OK);

  /* Are there enough bytes in packet to at least retrieve length of next chunk ? */
  if (bytes_left < SN_MIN_CHUNK_SIZE)
    return(SN_PARSE_ERROR_CHHL);

  chunk_hdr = SN_SCTP_NEXTCHUNK(chunk_hdr);

  /* Is the chunk long enough to not cause endless look and are there enough bytes in packet to rea
  chunk_length = SCTP_SIZE32(ntohs(chunk_hdr->chunk_length));
  if ((chunk_length < SN_MIN_CHUNK_SIZE) || (chunk_length > bytes_left))
    return(SN_PARSE_ERROR_CHHL);
  if(++chunk_count > sysctl_chunk_proc_limit)
    return(SN_PARSE_OK); /* limit for processing chunks, take what we get */
}

if (*passoc == NULL)
    return (partial_match)?(SN_PARSE_ERROR_PARTIALLOOKUP):(SN_PARSE_ERROR_LOOKUP);
else
  return(SN_PARSE_OK);
}

```

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().

```
                        :        /* a packet containing a SHUTDOWN-COMPLETE chunk */
    assoc->state = SN_CL;  /* Stay in Close state until timeout */
    if (sysctl_holddown_timer > 0)
      sctp_ResetTimeOut(la, assoc, SN_X_T(la));/* allow to stay open for Tbit packets*/
    else
      assoc->state = SN_RM;/* Mark for removal*/
    return(SN_NAT_PKT);
  case SN_SCTP_SHUTACK:          /* a packet containing a SHUTDOWN-ACK chunk */
    assoc->state = SN_CL;  /* Stay in Close state until timeout */
    sctp_ResetTimeOut(la, assoc, SN_C_T(la));
    return(SN_NAT_PKT);
  case SN_SCTP_ABORT:            /* a packet containing an ABORT chunk */
    assoc->state = SN_RM;/* Mark for removal*/
    return(SN_NAT_PKT);
  default:
    return(SN_DROP_PKT);
  }
  return(SN_DROP_PKT);/* shouldn't get here very bad: log, drop and hope for the best */
}

/* ----------------------------------------------------------------------
 *                           HASH TABLE CODE
```

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:

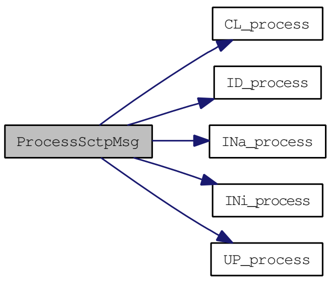
SN_DROP_PKT | SN_NAT_PKT | SN_REPLY_ABORT | SN_REPLY_ERROR | SN_-PROCESSING_ERROR

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().

```
                         {
  case SN_ID: /* Idle */
    rtnval = ID_process(la, direction, assoc, sm);
    if (rtnval != SN_NAT_PKT) {
      assoc->state = SN_RM;/* Mark for removal*/
    }
    return(rtnval);
  case SN_INi: /* Initialising - Init */
    return(INi_process(la, direction, assoc, sm));
  case SN_INa: /* Initialising - AddIP */
    return(INa_process(la, direction, assoc, sm));
  case SN_UP:  /* Association UP */
    return(UP_process(la, direction, assoc, sm));
  case SN_CL:  /* Association Closing */
    return(CL_process(la, direction, assoc, sm));
  }
  return(SN_PROCESSING_ERROR);
}
```

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 logsctperror (char ∗errormsg, 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 logsctpassoc (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.

```
                        {
  case SN_ID:
    sp = "ID ";
    break;
  case SN_INi:
    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\tga=%s\n",inet_ntoa(G_Addr->g_addr));
2593   }
2594 }
2595
```

5.4.3.2 static void logsctperror (char * *errormsg*, uint32_t *vtag*, int *error*, int *direction*) [static]

Log sctp nat errors.

Parameters:

errormsg 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, errormsg, 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 -";
```

```
    break;
  default:
    ploc = "";
  }
  switch(sm->msg) {
  case SN_SCTP_INIT:
    pstate = "Init";
    break;
  case SN_SCTP_INITACK:
    pstate = "InitAck";
    break;
  case SN_SCTP_ABORT:
    pstate = "Abort";
    break;
  case SN_SCTP_SHUTACK:
    pstate = "ShutAck";
    break;
  case SN_SCTP_SHUTCOMP:
    pstate = "ShutComp";
    break;
  case SN_SCTP_ASCONF:
    pstate = "Asconf";
    break;
 case SN_SCTP_ASCONFACK:
    pstate = "AsconfAck";
    break;
  case SN_SCTP_OTHER:
    pstate = "Other";
    break;
  default:
    pstate = "***ERROR***";
    break;
  }
  SctpAliasLog("Parsed: %s %s\n", ploc, pstate);
}

```

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.

```
                                        {
    LIST_FOREACH(assoc, &la->sctpNatTimer.TimerQ[i], timer_Q) {
      snprintf(buf, 50, " l=%u ",i);
      //SctpAliasLog(la->logDesc," l=%d ",i);
      logsctpassoc(assoc, buf);
    }
  }
}

```

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.

```
                     {
    if ((found->TableRegister == SN_GLOBAL_TBL) &&                      \
        (found->l_addr.s_addr == assoc->l_addr.s_addr) && (found->l_port == assoc->l_port)) { /* rese
      RmSctpAssoc(la, found);
      sctp_RmTimeOut(la, found);
      freeGlobalAddressList(found);
      sn_free(found);
    } else
      return(SN_ADD_CLASH);
  }

  LIST_INSERT_HEAD(&la->sctpTableGlobal[SN_TABLE_HASH(assoc->g_vtag, assoc->g_port, la->sctpNatTableS
                   assoc, list_G);
  assoc->TableRegister |= SN_GLOBAL_TBL;
  la->sctpLinkCount++; //increment link count

  if (assoc->TableRegister == SN_BOTH_TBL) {
    /* libalias log -- controlled by libalias */
    if (la->packetAliasMode & PKT_ALIAS_LOG)
      SctpShowAliasStats(la);

    SN_LOG(SN_LOG_INFO, logsctpassoc(assoc, "^"));
  }

  return(SN_ADD_OK);
}

```

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.

```
                                    :
   *  - the local receiver if receiving it for the first time will establish
   *    an association with the new global host
   *  - if receiving an init from a different global address after sending a
   *    lost initack it will send an initack to the new global host, the first
   *    association attempt will then be blocked if retried.
   */
  if (found != NULL) {
    if ((found->TableRegister == SN_LOCAL_TBL) && (found->g_port == assoc->g_port)) { /* resent messa
      RmSctpAssoc(la, found);
      sctp_RmTimeOut(la, found);
      freeGlobalAddressList(found);
      sn_free(found);
    } else
      return(SN_ADD_CLASH);
  }

  LIST_INSERT_HEAD(&la->sctpTableLocal[SN_TABLE_HASH(assoc->l_vtag, assoc->l_port, la->sctpNatTableSi
                   assoc, list_L);
  assoc->TableRegister |= SN_LOCAL_TBL;
  la->sctpLinkCount++; //increment link count

  if (assoc->TableRegister == SN_BOTH_TBL) {
    /* libalias log -- controlled by libalias */
    if (la->packetAliasMode & PKT_ALIAS_LOG)
      SctpShowAliasStats(la);

    SN_LOG(SN_LOG_INFO, logsctpassoc(assoc, "^"));
  }

  return(SN_ADD_OK);
}

```

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().

```
                  { /* an init packet, vtag==0 */
    i = SN_TABLE_HASH(g_vtag, g_port, la->sctpNatTableSize);
    LIST_FOREACH(assoc, &la->sctpTableGlobal[i], list_G) {
      if ((assoc->g_vtag == g_vtag) && (assoc->g_port == g_port) && (assoc->l_port == l_port)) {
        *partial_match = 1;
        if (assoc->num_Gaddr) {
          LIST_FOREACH(G_Addr, &(assoc->Gaddr), list_Gaddr) {
            if(G_Addr->g_addr.s_addr == g_addr.s_addr)
              return(assoc);
          }
        } else {
          return(assoc);
        }
      }
    }
  }
  return(NULL);
}

```

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.

```
                          { /* an init packet, vtag==0 */
    i = SN_TABLE_HASH(Cassoc->g_vtag, Cassoc->g_port, la->sctpNatTableSize);
    LIST_FOREACH(assoc, &la->sctpTableGlobal[i], list_G) {
      if ((assoc->g_vtag == Cassoc->g_vtag) && (assoc->g_port == Cassoc->g_port) && (assoc->l_port ==
        if (assoc->num_Gaddr) {
          LIST_FOREACH(G_AddrC, &(Cassoc->Gaddr), list_Gaddr) {
            LIST_FOREACH(G_Addr, &(assoc->Gaddr), list_Gaddr) {
              if(G_Addr->g_addr.s_addr == G_AddrC->g_addr.s_addr)
                return(assoc);
            }
          }
        } else {
          return(assoc);
        }
      }
    }
  }
  return(NULL);
}
```

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().

```
                    { /* an init packet, vtag==0 */
    i = SN_TABLE_HASH(g_vtag, l_port, la->sctpNatTableSize);
    LIST_FOREACH(assoc, &la->sctpTableLocal[i], list_L) {
      if ((assoc->l_vtag == g_vtag) && (assoc->l_port == l_port) && (assoc->g_port == g_port)) {
        if (assoc->num_Gaddr) {
          LIST_FOREACH(G_Addr, &(assoc->Gaddr), list_Gaddr) {
            if(G_Addr->g_addr.s_addr == g_addr.s_addr)
              return(assoc);
          }
        } else {
          return(assoc);
        }
      }
    }
  }
  return(NULL);
}

```

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().

```
                   { /* an init packet, vtag==0 */
    i = SN_TABLE_HASH(l_vtag, l_port, la->sctpNatTableSize);
    LIST_FOREACH(assoc, &la->sctpTableLocal[i], list_L) {
      if ((assoc->l_vtag == l_vtag) && (assoc->l_port == l_port) && (assoc->g_port == g_port)\
          && (assoc->l_addr.s_addr == l_addr.s_addr)) {
        if (assoc->num_Gaddr) {
          LIST_FOREACH(G_Addr, &(assoc->Gaddr), list_Gaddr) {
            if(G_Addr->g_addr.s_addr == g_addr.s_addr)
              return(assoc);
          }
        } else {
          return(assoc);
        }
      }
    }
  }
  return(NULL);
}

```

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().

```
                    { /* an init packet, vtag==0 */
    i = SN_TABLE_HASH(l_vtag, g_port, la->sctpNatTableSize);
    LIST_FOREACH(assoc, &la->sctpTableGlobal[i], list_G) {
      if ((assoc->g_vtag == l_vtag) && (assoc->g_port == g_port) && (assoc->l_port == l_port)) {
        if (assoc->num_Gaddr) {
          LIST_FOREACH(G_Addr, &(assoc->Gaddr), list_Gaddr) {
            if(G_Addr->g_addr.s_addr == G_Addr->g_addr.s_addr)
              return(assoc); /* full match */
          }
        } else {
          if (++cnt > 1) return(NULL);
          lastmatch = assoc;
        }
      }
    }
  }
  /* If there is more than one match we do not know which local address to send to */
  return( cnt ? lastmatch : NULL );
}

```

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

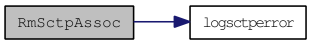
assoc pointer to sctp association

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 leq 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.

```
                                                   {
    while (!LIST_EMPTY(&la->sctpNatTimer.TimerQ[la->sctpNatTimer.cur_loc])) {
      assoc = LIST_FIRST(&la->sctpNatTimer.TimerQ[la->sctpNatTimer.cur_loc]);
      //SLIST_REMOVE_HEAD(&la->sctpNatTimer.TimerQ[la->sctpNatTimer.cur_loc], timer_Q);
      LIST_REMOVE(assoc, timer_Q);
      if (la->timeStamp >= assoc->exp) { /* state expired */
        SN_LOG(((assoc->state == SN_CL)?(SN_LOG_DEBUG):(SN_LOG_INFO)),
               logsctperror("Timer Expired", assoc->g_vtag, assoc->state, SN_TO_NODIR));
        RmSctpAssoc(la, assoc);
        freeGlobalAddressList(assoc);
        sn_free(assoc);
      } else {/* state not expired, reschedule timer*/
        sctp_AddTimeOut(la, assoc);
      }
    }
    /* Goto next location in the timer queue*/
    ++la->sctpNatTimer.loc_time;
    if (++la->sctpNatTimer.cur_loc >= SN_TIMER_QUEUE_SIZE)
      la->sctpNatTimer.cur_loc = 0;
  }
}

/* ----------------------------------------------------------------------
 *                             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.

```
        {
    assoc->exp = newexp;
  }
}

```

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.

```
{
```

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.

```
                                                : 0;

  return (0);
}

```

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.

```
                                                                          : procli

  return (0);
}


```

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.

```
                                                                          : flag;

  return (0);
}

```

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.

```
                                                  :((size > SN_MAX_HASH_SIZE)?(SN_MAX_HASH_SIZE):(s

  size |= 0x00000001; /* make odd */

  for(;(((size % 3) == 0) || ((size % 5) == 0) || ((size % 7) == 0) || ((size % 11) == 0)); size+=2);
  sysctl_hashtable_size = size;

  return (0);
}

```

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.

```
                                                        : proclimit;
  sysctl_chunk_proc_limit =
    (sysctl_chunk_proc_limit < sysctl_initialising_chunk_proc_limit) ? sysctl_initialising_chunk_proc_

  return (0);
}

```

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.

```
                                                                :(level);
  sysctl_log_level = (level < SN_LOG_LOW)?(SN_LOG_LOW):(level);

  return (0);
}

```

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.

```
                      : proclimit;

  return (0);
}

```

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.

```
                                               :(timer);

  if (((u_int *)arg1) != &sysctl_holddown_timer)
  {
    timer = (timer < SN_MIN_TIMER)?(SN_MIN_TIMER):(timer);
  }

  *(u_int *)arg1 = timer;

  return (0);
}

```

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.

```
                                                                                        :

  return (0);
}


/* ----------------------------------------------------------------------
```

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 Find-SctpLocalT().

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 Find-SctpLocalT().

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 Find-SctpLocalT().

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:

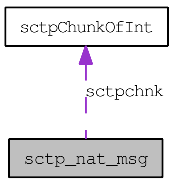
- alias_sctp.h

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

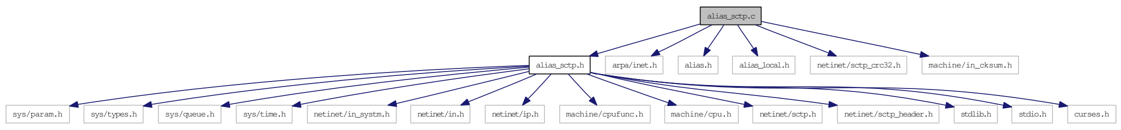
Chapter 7

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 + sysctl_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 logsctperror (char ∗errormsg, 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 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

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.

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.

```
                            {
    LIST_INSERT_HEAD(&(assoc->Gaddr), G_addr, list_Gaddr); /* add new address to beginning of list*/
  } else {
    LIST_FOREACH(iter_G_Addr, &(assoc->Gaddr), list_Gaddr) {
      if (G_addr->g_addr.s_addr == iter_G_Addr->g_addr.s_addr)
        return(0); /* already exists, so don't add */
    }
    LIST_INSERT_AFTER(first_G_Addr, G_addr, list_Gaddr); /* add address to end of list*/
  }
  assoc->num_Gaddr++;
  return(1); /* success */
}

```

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.

```
                                                                      :%d)\n", la->sctpNatTableSize));
  la->sctpTableLocal = sn_calloc(la->sctpNatTableSize, sizeof(struct sctpNatTableL));
  la->sctpTableGlobal = sn_calloc(la->sctpNatTableSize, sizeof(struct sctpNatTableG));
  la->sctpNatTimer.TimerQ = sn_calloc(SN_TIMER_QUEUE_SIZE, sizeof(struct sctpTimerQ));
  /* Initialise hash table */
  for (i = 0; i < la->sctpNatTableSize; i++) {
    LIST_INIT(&la->sctpTableLocal[i]);
    LIST_INIT(&la->sctpTableGlobal[i]);
  }

  /* Initialise circular timer Q*/
  for (i = 0; i < SN_TIMER_QUEUE_SIZE; i++)
    LIST_INIT(&la->sctpNatTimer.TimerQ[i]);
#ifdef _KERNEL
  la->sctpNatTimer.loc_time=time_uptime; /* la->timeStamp is not set yet */
#else
  la->sctpNatTimer.loc_time=la->timeStamp;
#endif
  la->sctpNatTimer.cur_loc = 0;
  la->sctpLinkCount = 0;
}

```

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.

```
                                             {
    assoc1 = LIST_FIRST(&la->sctpNatTimer.TimerQ[i]);
    while (assoc1 != NULL) {
      freeGlobalAddressList(assoc1);
      assoc2 = LIST_NEXT(assoc1, timer_Q);
      sn_free(assoc1);
      assoc1 = assoc2;
    }
  }

  sn_free(la->sctpTableLocal);
  sn_free(la->sctpTableGlobal);
  sn_free(la->sctpNatTimer.TimerQ);
}

```

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.

```
                                                                  {
    SctpAliasLog("ERROR: Invalid direction\n");
    return(PKT_ALIAS_ERROR);
  }

```

```
sctp_CheckTimers(la); /* Check timers */

/* Parse the packet */
rtnval = sctp_PktParser(la, direction, pip, &msg, &assoc); //using *char (change to mbuf when get co
switch (rtnval) {
case SN_PARSE_OK:
  break;
case SN_PARSE_ERROR_CHHL:
  /* Not an error if there is a chunk length parsing error and this is a fragmented packet */
  if (ntohs(pip->ip_off) & IP_MF) {
     rtnval = SN_PARSE_OK;
     break;
  }
  SN_LOG(SN_LOG_EVENT,
         logsctperror("SN_PARSE_ERROR", msg.sctp_hdr->v_tag, rtnval, direction));
  return(PKT_ALIAS_ERROR);
case SN_PARSE_ERROR_PARTIALLOOKUP:
  if (sysctl_error_on_ootb > SN_LOCALandPARTIAL_ERROR_ON_OOTB) {
    SN_LOG(SN_LOG_EVENT,
           logsctperror("SN_PARSE_ERROR", msg.sctp_hdr->v_tag, rtnval, direction));
    return(PKT_ALIAS_ERROR);
  }
case SN_PARSE_ERROR_LOOKUP:
  if (sysctl_error_on_ootb == SN_ERROR_ON_OOTB ||
      (sysctl_error_on_ootb == SN_LOCALandPARTIAL_ERROR_ON_OOTB && direction == SN_TO_LOCAL) ||
      (sysctl_error_on_ootb == SN_LOCAL_ERROR_ON_OOTB && direction == SN_TO_GLOBAL)) {
    TxAbortErrorM(la, &msg, assoc, SN_REFLECT_ERROR, direction); /*NB assoc=NULL */
    return(PKT_ALIAS_RESPOND);
  }
default:
   SN_LOG(SN_LOG_EVENT,
          logsctperror("SN_PARSE_ERROR", msg.sctp_hdr->v_tag, rtnval, direction));
   return(PKT_ALIAS_ERROR);
}

SN_LOG(SN_LOG_DETAIL,
       logsctpassoc(assoc, "*");
       logsctpparse(direction, &msg);
      );

/* Process the SCTP message */
rtnval = ProcessSctpMsg(la, direction, &msg, assoc);

SN_LOG(SN_LOG_DEBUG_MAX,
       logsctpassoc(assoc, "-");
       logSctpLocal(la);
       logSctpGlobal(la);
      );
SN_LOG(SN_LOG_DEBUG, logTimerQ(la));

switch(rtnval){
case SN_NAT_PKT:
  switch(direction) {
  case SN_TO_LOCAL:
    DifferentialChecksum(&(msg.ip_hdr->ip_sum),
                         &(assoc->l_addr), &(msg.ip_hdr->ip_dst), 2);
    msg.ip_hdr->ip_dst = assoc->l_addr; /* change dst address to local address*/
    break;
  case SN_TO_GLOBAL:
    DifferentialChecksum(&(msg.ip_hdr->ip_sum),
                         &(assoc->a_addr),  &(msg.ip_hdr->ip_src), 2);
    msg.ip_hdr->ip_src = assoc->a_addr; /* change src to alias addr*/
    break;
  default:
    rtnval = SN_DROP_PKT; /* shouldn't get here, but if it does drop packet */
    SN_LOG(SN_LOG_LOW, logsctperror("ERROR: Invalid direction", msg.sctp_hdr->v_tag, rtnval, directi
    break;
    }
    break;
  case SN_DROP_PKT:
    SN_LOG(SN_LOG_DETAIL, logsctperror("SN_DROP_PKT", msg.sctp_hdr->v_tag, rtnval, direction));
    break;
  case SN_REPLY_ABORT:
  case SN_REPLY_ERROR:
  case SN_SEND_ABORT:
    TxAbortErrorM(la, &msg, assoc, rtnval, direction);
    break;
  default:
    // big error, remove association and go to idle and write log messages
    SN_LOG(SN_LOG_LOW, logsctperror("SN_PROCESSING_ERROR", msg.sctp_hdr->v_tag, rtnval, direction));
    assoc->state=SN_RM;/* Mark for removal*/
    break;
  }

  /* Remove association if tagged for removal */
  if (assoc->state == SN_RM) {
    if (assoc->TableRegister) {
      sctp_RmTimeOut(la, assoc);
      RmSctpAssoc(la, assoc);
    }
    LIBALIAS_LOCK_ASSERT(la);
    freeGlobalAddressList(assoc);
    sn_free(assoc);
  }
  switch(rtnval) {
  case SN_NAT_PKT:
    return(PKT_ALIAS_OK);
  case SN_SEND_ABORT:
    return(PKT_ALIAS_OK);
  case SN_REPLY_ABORT:
  case SN_REPLY_ERROR:
  case SN_REFLECT_ERROR:
    return(PKT_ALIAS_RESPOND);
  case SN_DROP_PKT:
  default:
    return(PKT_ALIAS_ERROR);
  }
}

```

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

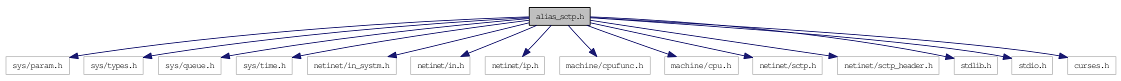
Definition at line 890 of file alias_sctp.c.

```
                                           { /* short packet, cannot send error cause */
    include_error_cause = 0;
    ip_size = ip_size -  sizeof(struct sctp_error_cause);
    sctp_size = sctp_size -  sizeof(struct sctp_error_cause);
  }
  /* Assign header pointers packet */
  struct ip* ip = (struct ip *) tmp_ip;
  struct sctphdr* sctp_hdr = (struct sctphdr *) ((char *) ip + sizeof(*ip));
  struct sctp_chunkhdr* chunk_hdr = (struct sctp_chunkhdr *) ((char *) sctp_hdr + sizeof(*sctp_hdr));
  struct sctp_error_cause* error_cause = (struct sctp_error_cause *) ((char *) chunk_hdr + sizeof(*chu

  /* construct ip header */
  ip->ip_v = sm->ip_hdr->ip_v;
  ip->ip_hl = 5; /* 5*32 bit words */
  ip->ip_tos = 0;
  ip->ip_len = htons(ip_size);
  ip->ip_id =  sm->ip_hdr->ip_id;
  ip->ip_off = 0;
  ip->ip_ttl = 255;
  ip->ip_p = IPPROTO_SCTP;
  /*
    The definitions below should be removed when they make it into the SCTP stack
  */
#define SCTP_MIDDLEBOX_FLAG 0x02
#define SCTP_NAT_TABLE_COLLISION 0x00b0
#define SCTP_MISSING_NAT 0x00b1
  chunk_hdr->chunk_type = (sndrply & SN_TX_ABORT) ? SCTP_ABORT_ASSOCIATION : SCTP_OPERATION_ERROR;
  chunk_hdr->chunk_flags = SCTP_MIDDLEBOX_FLAG;
  if (include_error_cause) {
    error_cause->code = htons((sndrply & SN_REFLECT_ERROR) ? SCTP_MISSING_NAT :  SCTP_NAT_TABLE_COLLIS
    error_cause->length = htons(sizeof(struct sctp_error_cause));
    chunk_hdr->chunk_length = htons(sizeof(*chunk_hdr) + sizeof(struct sctp_error_cause));
  } else {
    chunk_hdr->chunk_length = htons(sizeof(*chunk_hdr));
  }

  /* set specific values */
  switch(sndrply) {
  case SN_REFLECT_ERROR:
    chunk_hdr->chunk_flags |= SCTP_HAD_NO_TCB; /* set Tbit */
    sctp_hdr->v_tag =  sm->sctp_hdr->v_tag;
    break;
  case SN_REPLY_ERROR:
    sctp_hdr->v_tag = (direction == SN_TO_LOCAL) ? assoc->g_vtag :  assoc->l_vtag ;
    break;
  case SN_SEND_ABORT:
    sctp_hdr->v_tag =  sm->sctp_hdr->v_tag;
    break;
  case SN_REPLY_ABORT:
    sctp_hdr->v_tag = sm->sctpchnk.Init->initiate_tag;
    break;
  }

  /* Set send/reply values */
  if (sndrply == SN_SEND_ABORT) { /*pass through NAT */
    ip->ip_src = (direction == SN_TO_LOCAL) ? sm->ip_hdr->ip_src : assoc->a_addr;
    ip->ip_dst = (direction == SN_TO_LOCAL) ? assoc->l_addr : sm->ip_hdr->ip_dst;
    sctp_hdr->src_port = sm->sctp_hdr->src_port;
    sctp_hdr->dest_port = sm->sctp_hdr->dest_port;
  } else { /* reply and reflect */
    ip->ip_src = sm->ip_hdr->ip_dst;
    ip->ip_dst = sm->ip_hdr->ip_src;
    sctp_hdr->src_port = sm->sctp_hdr->dest_port;
    sctp_hdr->dest_port = sm->sctp_hdr->src_port;
  }

  /* Calculate IP header checksum */
  ip->ip_sum = in_cksum_hdr(ip);

  /* calculate SCTP header CRC32 */
  sctp_hdr->checksum = 0;
  sctp_hdr->checksum = sctp_csum_finalize(update_crc32(0xffffffff, (unsigned char *) sctp_hdr, sctp_si

  memcpy(sm->ip_hdr, ip, ip_size);

  SN_LOG(SN_LOG_EVENT,SctpAliasLog("%s %s 0x%x (->%s:%u vtag=0x%x crc=0x%x)\n",
                                   ((sndrply == SN_SEND_ABORT) ? "Sending" : "Replying"),
                                   ((sndrply & SN_TX_ERROR) ? "ErrorM" : "AbortM"),
                                   (include_error_cause ? ntohs(error_cause->code) : 0),
                                   inet_ntoa(ip->ip_dst),ntohs(sctp_hdr->dest_port),
                                   ntohl(sctp_hdr->v_tag), ntohl(sctp_hdr->checksum)));
}

/* ----------------------------------------------------------------------
 *                            PACKET PARSER CODE
```

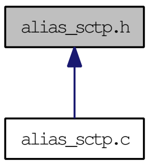
7.2 alias_sctp.h File Reference

```
#include <sys/param.h>
#include <sys/types.h>
#include <sys/queue.h>
#include <sys/time.h>
#include <netinet/in_systm.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 logsctperror(), and logsctpparse().

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 logsctperror(), logsctpparse(), 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