Alias_sctp Version 0.2: SCTP NAT implementation in IPFW

David Hayes, Jason But
Centre for Advanced Internet Architectures, Technical Report 081128A
Swinburne University of Technology
Melbourne, Australia
david.hayes@ieee.org, jbut@swin.edu.au

***Abstract*—Alias_sctp is part of the SONATA[1] project to develop and release a BSD licensed implementation of a Network Address Translation (NAT) module that supports the Stream Control Transmission Protocol (SCTP). Arriving SCTP packets are first parsed for information that may be important to the association's state in the NAT. This information along with the SCTP message is then passed to the state machine for further processing for a final decision on whether to NAT the packet, and whether the association's state in the NAT should change.**

Alias_sctp is implemented as a patch for the `libalias` kernel module, which works with the FreeBSD IPFW2 NAT module. Many of alias_sctp's parameters can be configured dynamically through the FreeBSD sysctl interface. Some minor modifications to the storage and management of IP addresses will need to be made when `libalias` supports IPv6. Alias_sctp 0.2 is a fully functional IPv4 SCTP NAT.

I. Introduction

Alias_sctp 0.2.x is a fully functional Stream Control Transmission Protocol (SCTP) Network Address Translation module for FreeBSD. The module works with the IPFW2 NAT kernel module, patching the libalias kernel module. This technical report now obsoletes the original Alias_sctp 0.1 report [2].

SCTP[3] is a reliable transport protocol operating on top of the Internet Protocol (IP). It was originally designed to transport telephony signalling but its application is broad, providing improved UDP and TCP type functionality.

Standard TCP/UDP type NAT implementations use *<address,ports>* tuples to identify each connection. This will not work with SCTP because:

- SCTP's checksum requires calculation of a CRC-32C over the entire SCTP message. Therefore, any changes made, such as a port number, require a full recalculation of the checksum.
- SCTP supports multi-homing of endpoints. If a NAT was to change the port number to avoid a lookup table conflict, this would require synchronisation with other NATs which may be connected though alternate paths through the network.
- SCTP's messages can contain a number of control chunks, which need to be scanned to determine whether the incoming packet will change the state of the association in the NAT.

This report outlines the design of `alias_sctp` 0.2.x. Following a description of the terminology used in the report, the details of the SCTP NAT design are described. These include the packet parser, state machine, look up tables, and timer queue designs. Following this are some notes on the implementation in the FreeBSD ipfw2 libalias context and changes that will be required for IPv6 functionality when this is supported by `libalias`. The report concludes with a summary of achievements and suggestions for further enhancements and analysis.

II. Terminology

The operational states and packet processing sequences in alias_sctp are described using ITU-T Message Sequence Charts (MSC) [4]. Table I describes some of the notation used in the MSCs.

The SCTP protocol unit is termed a *message*. Chunks within an SCTP message are identified in **UPPERCASE-BOLD**, while parameters within chunks are bold upper and lower case.

III. Packet parser

Figure 1 gives an overview of the SCTP NAT. When a packet arrives, the state of the timer queue is checked, with any resulting timeouts processed. Following this, packets are parsed for relevant SCTP information. The appropriate association state is then searched for in the

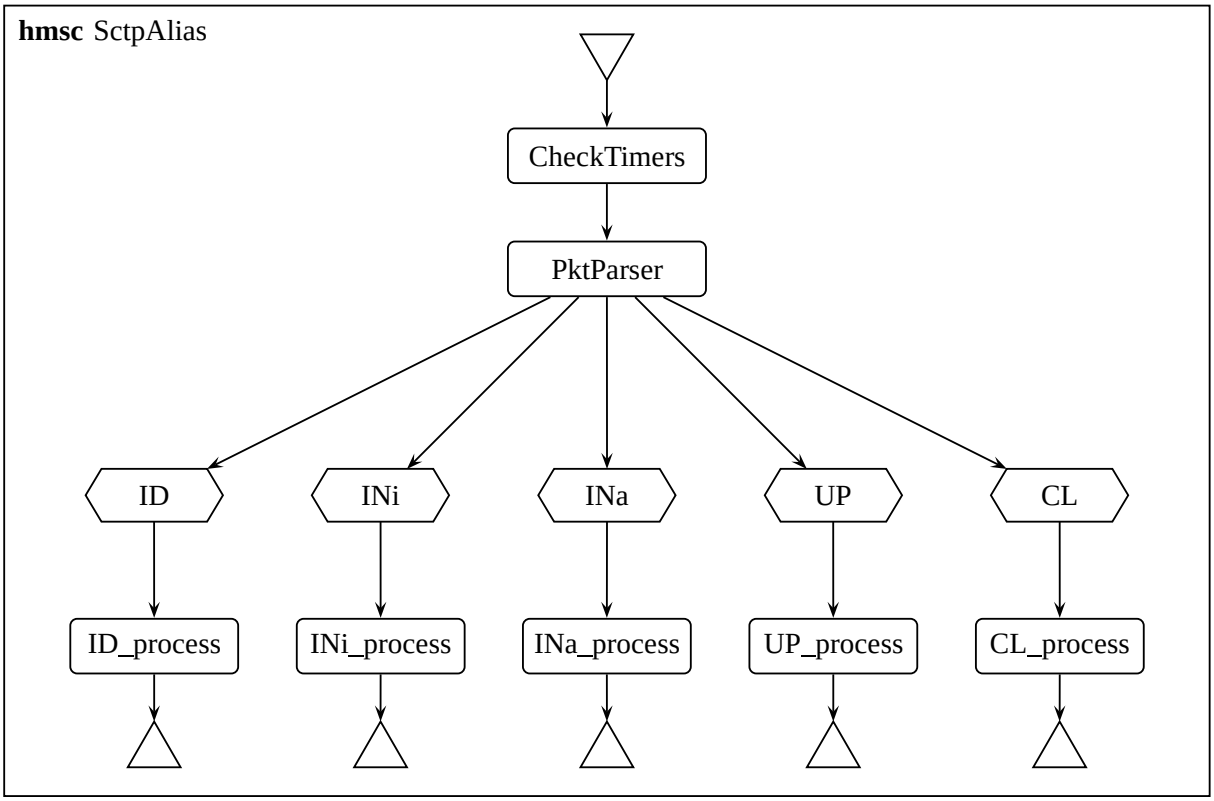


Fig. 1. SCTP NAT overview

sctp(*)	any sctp message
[sctp(*)]	The sending of this message is dependent on the NATs configuration
l_sctp(*)	sctp message that has been received from a local address.
g_sctp(*)	sctp message that is from a global address outside the NAT device.
r_sctp(*)	sctp message reply back to the sender, usually under error. conditions.
f_sctp(*)	forward of sctp packet through NAT.
{l,g}_vtag	Set appropriate local or global verification tag
init_tag	Initiation Tag
*_T	Any timer
LT	Look up Table
assoc_state	Stored state of an SCTP association

TABLE I
SCTP MSC SYNTAX

lookup tables. The packet is processed according to the recorded state of the SCTP association.

A. SCTP messages

SCTP's multi-streaming capability makes parsing packets more difficult than would be the case with a TCP/UDP NAT. An IP packet contains a single SCTP message. However this message may contain many control and data streams delineated as chunks (see Figure 2). In addition, each control chunk may contain a number of parameters, some of which are relevant to the NAT. Generally the NAT parses just enough information in the `PktParser` to determine the current state of the association the packet belongs to. Further parsing of the chunk and parameters is done as required by the state machine.

SCTP uses a Verification tag (vtag), to differentiate associations. The vtag is randomly generated by the endpoints during the association initialisation process and should be used as part of the tuple to identify associations. `Alias_sctp` uses a tuple consisting of <*vtag, ports, [global IP address]*> to identify associations. The use of the global IP addresses is optional [5] (see [6] for a full discussion on pros and cons of tracking global IP addresses). Using vtags within the identification tuple allows the NAT to leave port numbers unchanged. There is then no need to recalculate the SCTP checksum.

B. Parsing mechanism

Figure 3 outlines the operation of the `PktParser`. First, the source and destination addresses are obtained

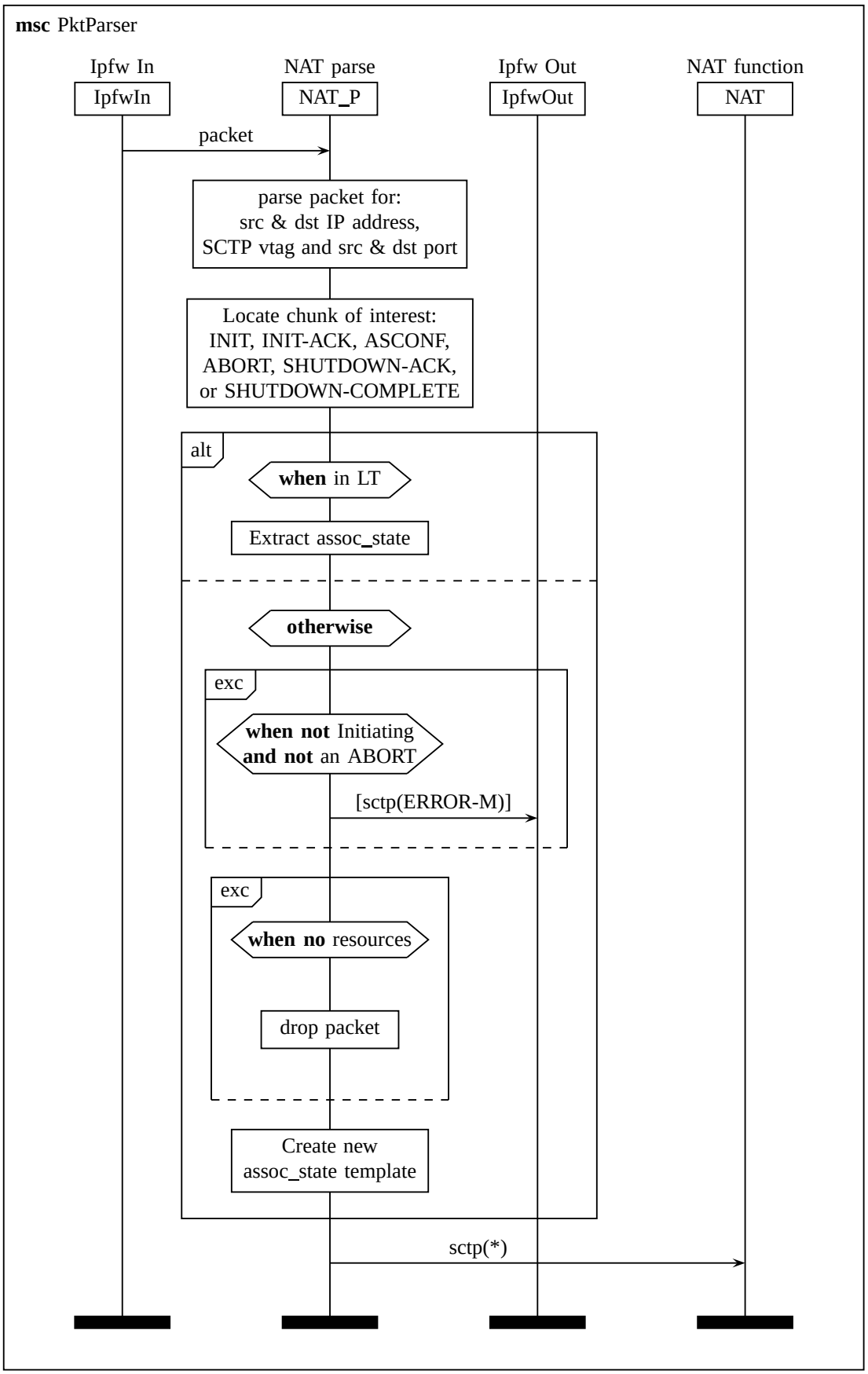
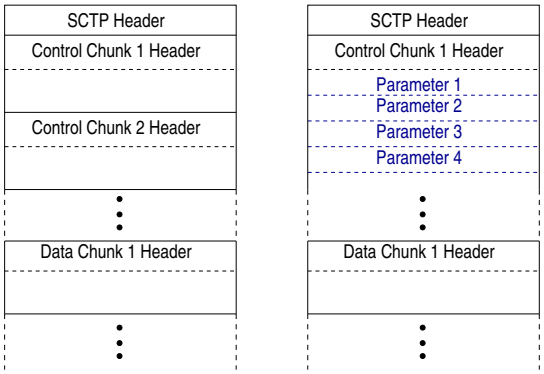


Fig. 3. Parsing and checking of an arriving packet



(a) Multiple chunks (b) Multiple parameters per chunk

Fig. 2. SCTP messages

from the IP header. SCTP's common header is then parsed to find the verification tag and source and destination port numbers. There is now enough information to search the association Look up Table (LT) to see if there is association state information matching this packet. If no matching association is found, parsing continues since this packet may contain an initiating control chunk.

Control chunks are then scanned to find the most important to the state of the association in the NAT. Only the most important (in terms of NAT state) is referenced for processing in the state machine. The ranking of control chunks, by importance to the NAT, is as follows (1 is most important):

1) **ABORT** – Other chunks are meaningless
2) Initiation (**INIT**) – No other chunks are allowed in SCTP packet
3) Initiation Acknowledgement (**INIT-ACK**) – No other chunks are allowed in SCTP packet
4) Shutdown Acknowledgement (**SHUTDOWN-ACK**) – Other chunks are meaningless in SCTP packet
5) Shutdown Complete (**SHUTDOWN-COMPLETE**) – No other chunks are allowed in SCTP packet
6) Address Configuration (**ASCONF**) – contains instructions to add or delete IP addresses.
7) Address Configuration Acknowledgement (**ASCONF-ACK**)

If the packet was initiating a new association, a new `assoc_state` structure (see code segment 1) is created, otherwise the matching `assoc_state` is used. This information is passed to the state machine.

If no match was found, an **ERROR-M** chunk may be sent to indicate a NAT table look up failure if allowed by the current NAT policy (see section VII-B2).

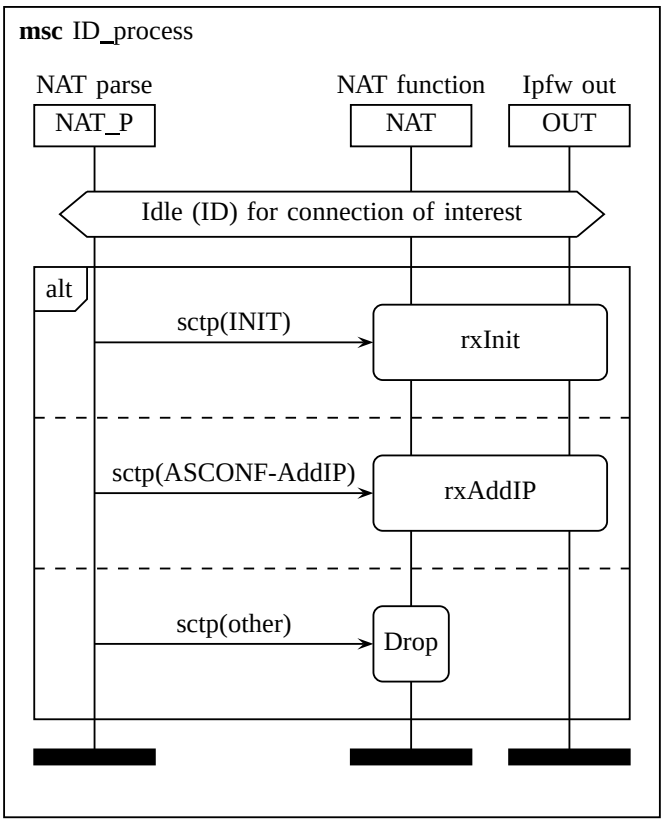


Fig. 4. Initiate dynamic NAT rule

IV. SCTP NAT STATE MACHINE

After enough information has been gleaned from an incoming SCTP message to determine its associations' current state in the NAT, the packet is passed to the relevant part of the state machine (see Figure 1). This section looks at the mechanism in each of the two main states (Idle (ID) and UP), and three transitory states (INi, INa, and CL) which wait for confirmation before moving to either ID or UP. INi and INa are progressed to from ID on receipt of an **INIT** or **ASCONF-AddIP** respectively. CL (Closing) is progressed to from UP on receipt of a **SHUTDOWN-ACK**.

A. No matching association – Idle (ID)

If there is no matching association, the state for the arriving message is Idle (ID) (see figure 4). Receipt of an SCTP message with an **INIT** chunk, or if permitted an **ASCONF-AddIP** chunk, starts the association initialisation process for the NAT. No other SCTP messages are permitted to pass through the NAT for this association while in ID.

Code segment 1 The SCTP association state stored in the look up tables

```
struct sctp_nat_assoc {
  uint32_t l_vtag; /**< local side verification tag */
  uint16_t l_port; /**< local side port number */
  uint32_t g_vtag; /**< global side verification tag */
  uint16_t g_port; /**< global side port number */
  struct in_addr l_addr; /**< local ip address */
  struct in_addr a_addr; /**< alias ip address */
  int state; /**< current state of NAT association */
  int TableRegister; /**< stores which look up tables association is registered in */
  int exp; /**< timer expiration in seconds from uptime */
  int exp_loc; /**< current location in timer_Q */
  int num_Gaddr; /**< number of global IP addresses in the list */
  LIST_HEAD(sctpGlobalAddresshead,sctp_GlobalAddress) Gaddr; /**< List of global addresses */
  LIST_ENTRY (sctp_nat_assoc) list_L; /**< Linked list of pointers for Local table*/
  LIST_ENTRY (sctp_nat_assoc) list_G; /**< Linked list of pointers for Global table */
  LIST_ENTRY (sctp_nat_assoc) timer_Q; /**< Linked list of pointers for timer Q */
//Using libalias locking
};
```

*1) **INIT**:*

When the NAT receives an **INIT** chunk (Figure 5), it begins to fill the association state structure (`assoc_state`:

- The initiation tag (*init_tag*) parameter is extracted. SCTP packets flowing in the opposite direction to the **INIT** will use this value as their *vtag*.
- If the **INIT** is from the global side, and the NAT is tracking global IP addresses, the IP address parameters will also be extracted. This presumes that there are appropriate port forwarding rules.
- Source and destination addresses
- Source and destination ports

If adding the `assoc_state` to the relevant lookup table (local or global) will cause a conflict, the NAT may send an **ABORT** chunk back to the sender with the M-bit set to indicate that it was generated by the middle box [7] (referred to as **ABORT-M**). This requests that the initiating host re-initiate the association with a newly generated *init_tag* and remedy the indexing conflict.

Under extreme load or error conditions the NAT may have no resources available to add the new association. In such circumstances it will drop the packet (see Figure 6).

The association will not be considered to be active (UP) until the the corresponding **INIT-ACK** is received, so the association enters the transitory state INi.

*2) **ASCONF-AddIP**:*

An extension to the SCTP protocol defined in [8] allows for dynamic address reconfiguration with **ASCONF** chunks. Of particular interest to the NAT are the **AddIP** and **DelIP** parameters. To cope with a machine that has multiple interfaces connected to multiple NAT devices, a NAT may receive an **ASCONF-AddIP** chunk even though it does not have a current association for the originating IP address [7]. If configured to allow association establishment in this manner (see section VII-B3) an **ASCONF-AddIP** message is treated in a similar way to an **INIT** message to establish an association in the NAT (see Figure 7). Two key differences with the AddIp in this context are:

- The AddIP parameter will not contain additional IP addresses, the source IP address will be used.
- The AddIP parameter will contain both *vtags*, so the LT entry can be completely filled.

The association will not be considered to be active (UP) until a coresponding **ASCONF-ACK** is received, so the association enters the transitory state INa.

B. Confirming association initialisation (INi and INa)

INi and INa are transitory states waiting for association confirmation in the form of the corresponding **INIT-ACK** or **ASCONF-ACK**, before moving to the UP state. Figures 8 and 9 give an overview of this process.

If a relayed **INIT** or **ASCONF-AddIP** message failed to reach its destination, the sender may retransmit it on timeout. The NAT forwards the retransmitted message and resets timer I_T (see Figures 12 and 13). Apart from retransmissions and the confirmation messages, all other SCTP messages are not permitted in this context, and are dropped.

*1) **INIT-ACK**:*

When the NAT receives an **INIT-ACK** it completes `assoc_state` (see Figure 12). The additional *vtag* and IP address information may conflict with entries

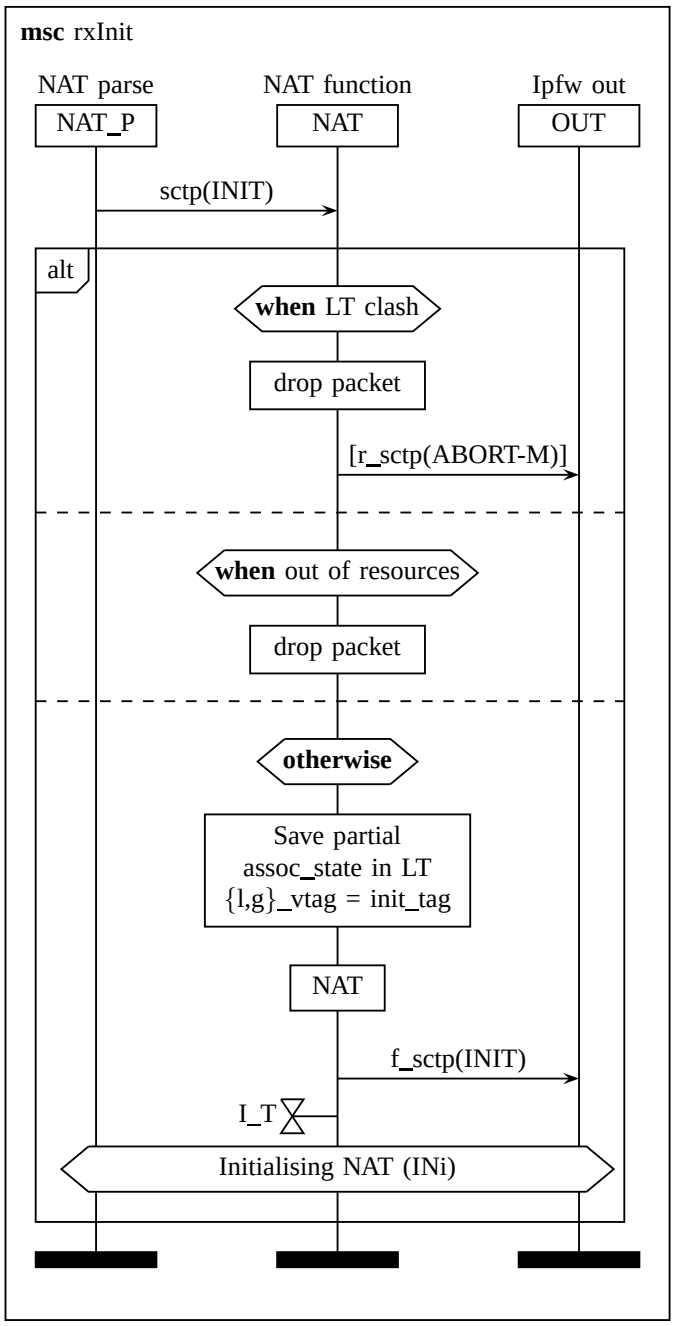


Fig. 5. Initiate dynamic NAT rule

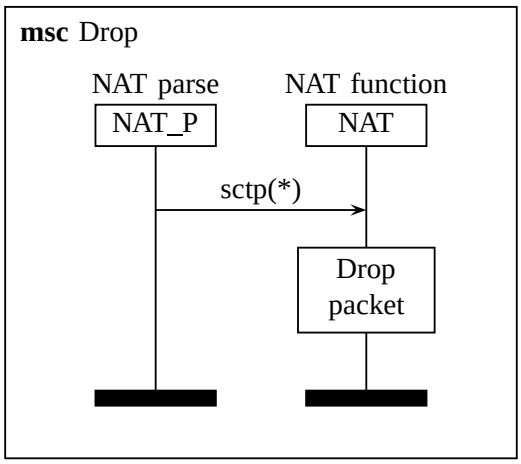


Fig. 6. Action on receipt of an invalid SCTP message

already in the look up table. If this occurs, an **ABORT-M** message may be forwarded to the initiating host to inform it of the clash. Association initiation can then be restarted. Otherwise, the complete `assoc_state` is added to the look up table it has not yet been added to (local or global). The association is now UP.

*2) **ASCONF-AddIP**:*

Receipt of a valid **ASCONF-ACK** confirms association initialisation, changing the state to UP (see Figure 13).

ASCONF chunk parameters have a correlation-ID. This enables the end host to match the acknowledgements with the address configuration requests. Currently the NAT does not check the correlation-ID. For the NAT, a valid confirmation is an **ASCONF-ACK** chunk that matches the `assoc_state`, has an **Asconf-Success** parameter, and is travelling in the correct direction. (Note that the NAT does not check for an **ASCONF-ACK** if the association is in the UP state (see section IV-C4).

If the NAT was to check correlation IDs it would need to store the relevant **ASCONF** parameters with their correlation-IDs in a list as part of `assoc_state`. The address reconfiguration would then only become active on receipt of an **ASCONF-ACK** chunk with an **Asconf-Success** parameter with a matching correlation-ID. This list would need to be timed, so that old unacknowledged requests are removed. We consider the cost of additional state space and complexity outweigh the benefits of incorporating this mechanism into the NAT.

*3) Authenticated **ASCONF** chunks:*

ASCONF chunks must be authenticated [8] to guard against association hijacking attacks. Authentication is performed through the use of shared keys and the use of an **AUTHENTICATION** chunk in the SCTP message [9]. In the case of an OOTB **ASCONF-AddIP** initialising association state in the NAT, the NAT will not have the necessary keys to perform the authentication. Further, authenticating SCTP messages is processor intensive and may leave the NAT open to a DoS attack. Instead, the NAT does not authenticate any SCTP messages. Any problems in authentication are left to the end hosts to resolve. If the end host does not authenticate the **ASCONF** chunk, it will not reply with an **ASCONF-**

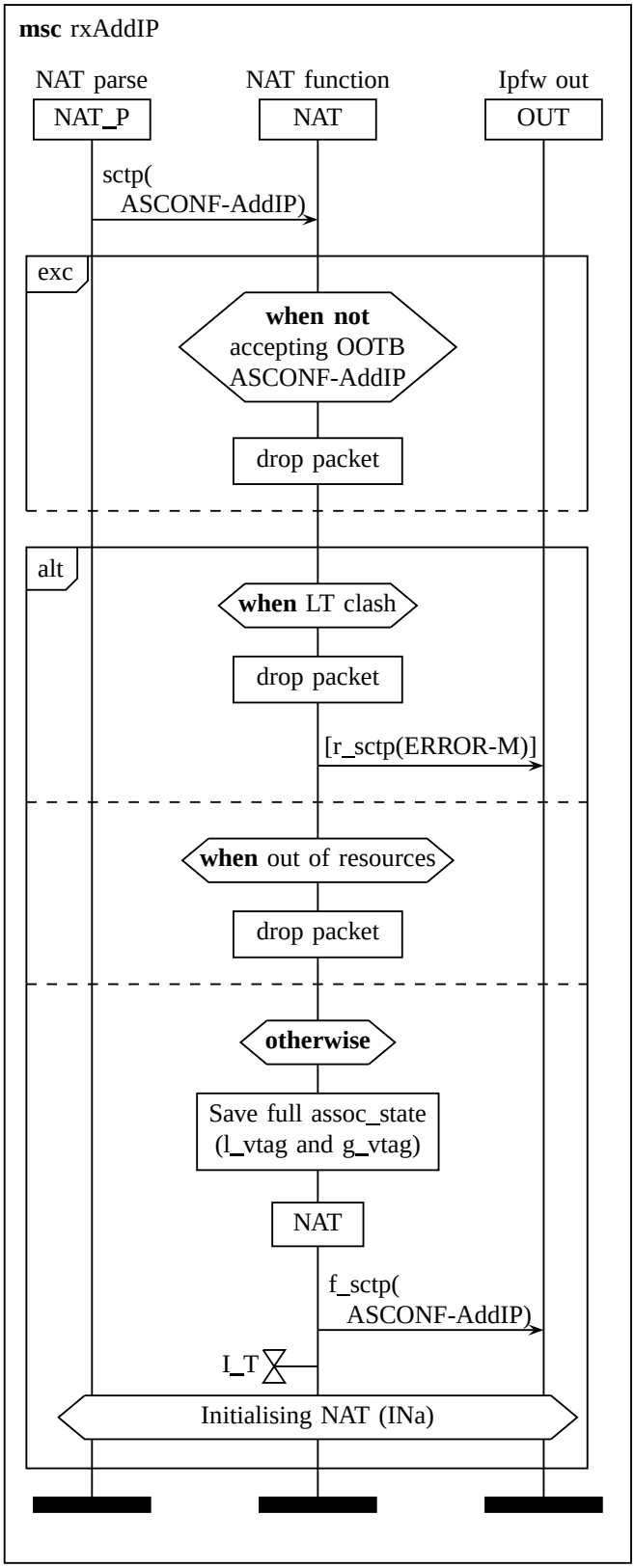


Fig. 7. Initiate dynamic NAT rule from AddIp

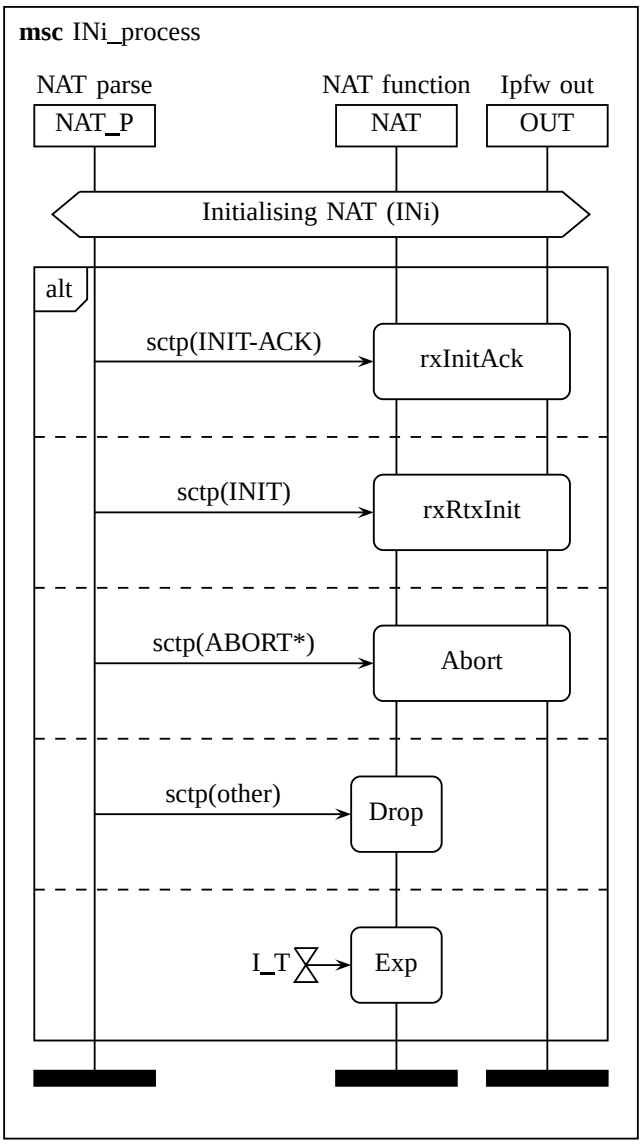


Fig. 8. Initialising NAT (**Init** based)

ACK. After I_T the association state will be removed from the NAT (logically now in ID).

Since the NAT does not authenticate or check the correlation-ID, it may be possible for an attacker to use **ASCONF** chunks to modify the state information stored in the NAT for an existing association in the UP state. This could lead to valid SCTP messages being dropped by the NAT, or allow an attacker to send packets to an internal host. Since an SCTP end host has full state and will authenticate all **ASCONF** chunks it receives, as well as confirming the correlation-IDs of acknowledgements, any invalid packets admitted by the NAT should not cause significant problems. However, valid packets that were errantly discarded by the NAT will cause temporary

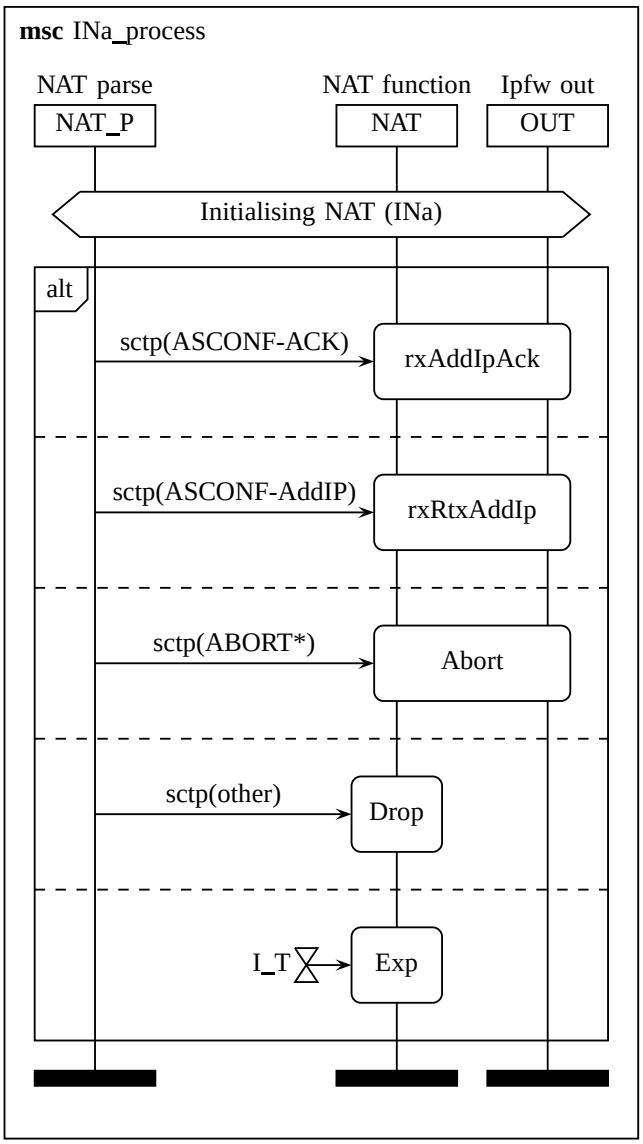


Fig. 9. Initialising NAT (**AddIP** based)

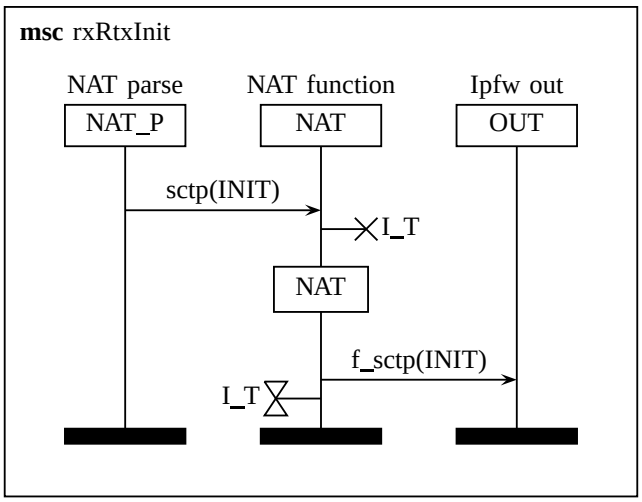


Fig. 10. Receiving a retransmitted **INIT** while in INi

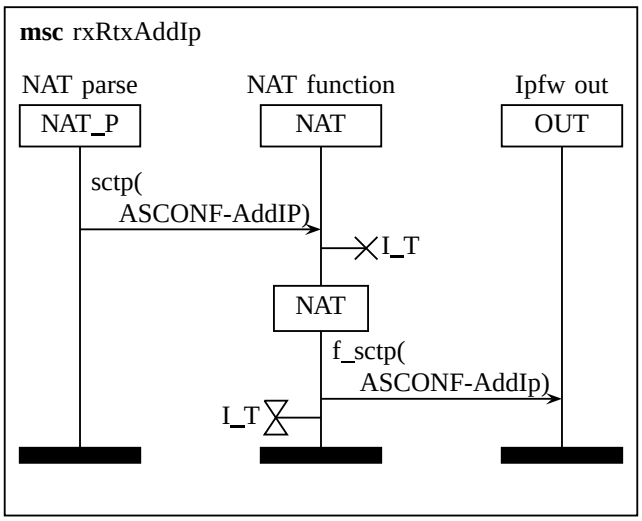


Fig. 11. Receiving a retransmitted **AddIp** while in INa

problems until the SCTP endpoints discover the loss of packets. We note that this is only a problem when global IP addresses are tracked by the NAT (further discussion in [6]).

C. NAT for association is UP

Once the NAT has all the association state information (and in the case of an OOTB **ASCONF-AddIP**, the acknowledgement), the association is considered to be UP for NATing SCTP messages for that particular association. An SCTP end host will not consider the association to be fully initiated until after the successful completion of the cookie exchange, which follows the **INIT** $\leftrightarrow$ **INIT-ACK** exchange. However, the cookie exchange is transparent to the NAT. The NAT, while enhancing security, has a primary purpose of translating IP addresses. It has all the information necessary to do this after the **INIT** $\leftrightarrow$ **INIT-ACK** exchange or **ASCONF-AddIP**.

Not waiting until after the cookie exchange does not significantly diminish the security provided by the NAT. If the cookie exchange fails, one of the end hosts will send an **ABORT**, which will also clear association state in the NAT. The NAT will however, allow any SCTP message to traverse it – not just SCTP messages with **COOKIE** or **COOKIE-ACK** chunks.

While in the UP state the association's state can only be modified by **SHUTDOWN-ACK**, **ABORT** chunks, timeouts, and when tracking global IP addresses,

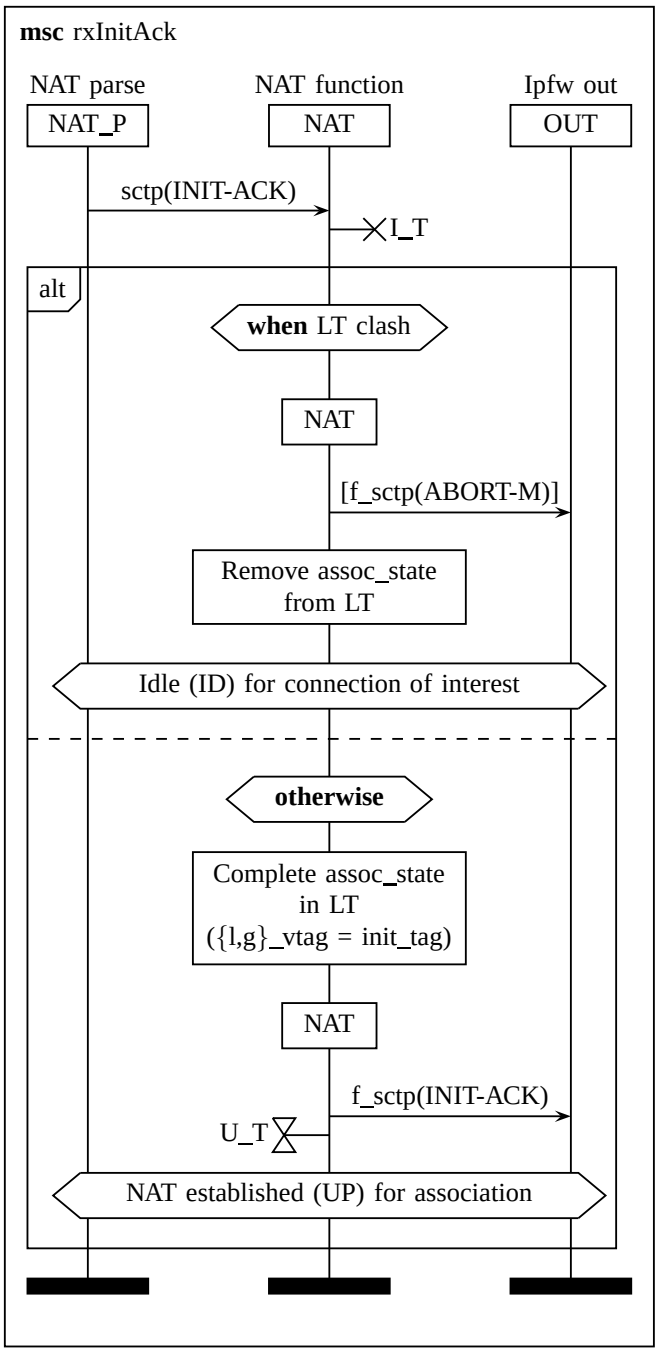


Fig. 12. Receiving an **INIT-ACK** while in INi

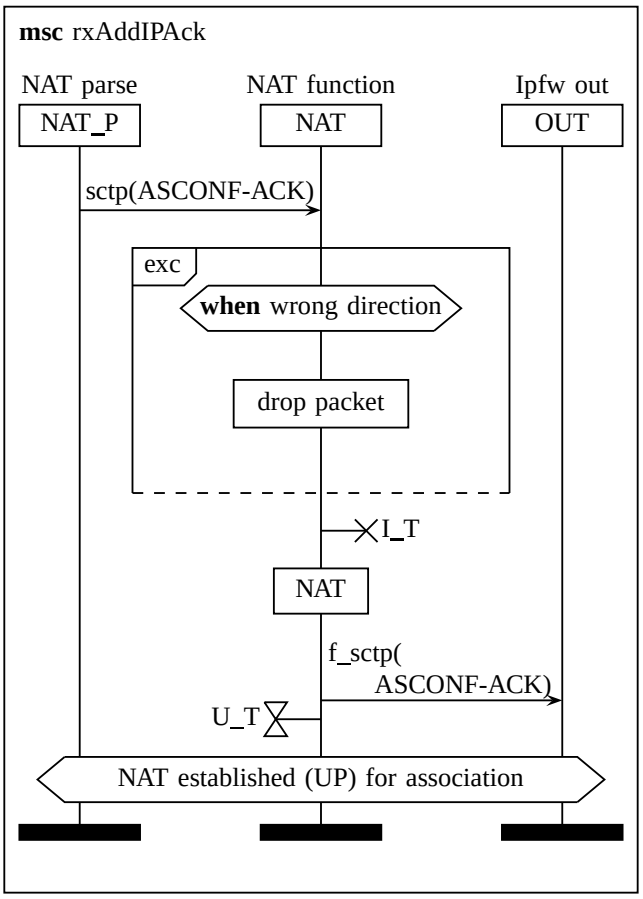


Fig. 13. Receiving an **AddIPAck** while in INa

ASCONF-AddIP and **ASCONF-DelIP** chunks (see Figure 14)

1) SCTP message NAT:
Figure 15 shows the traversal of non state changing SCTP messages, once the association state is UP. The UP timer (U_T) is reset on the receipt of each packet.

*2) **ASCONF-AddIP**:*
If we are tracking global IP addresses, then an incoming **ASCONF-AddIP** may contain additional global IP addresses that should be added to `assoc_state`. In this case, the NAT needs to process the **ASCONF-AddIP** (see Figure 16). Provided there are available resources, the **ASCONF** chunk is parsed for global IP addresses, which are added to the list in `assoc_state` (if they are not already in the list).

If we are not tracking global IP addresses or the **ASCONF-AddIP** is from a local host, the **ASCONF** message is simply NATed, and forwarded to its destination.

*3) **ASCONF-DelIP**:*
Figure 17 shows how the NAT processes an **ASCONF-DelIP** while an association is UP. Its processing mirrors that of an **ASCONF-AddIP**. An additional check must be made to ensure that at least one valid global IP address remains in the list of global IP addresses.

*4) **ASCONF** security:*
Both the **ASCONF-AddIP** and **ASCONF-DelIP** are processed without authentication, and without waiting for an **ASCONF-ACK** with a corresponding correlation-

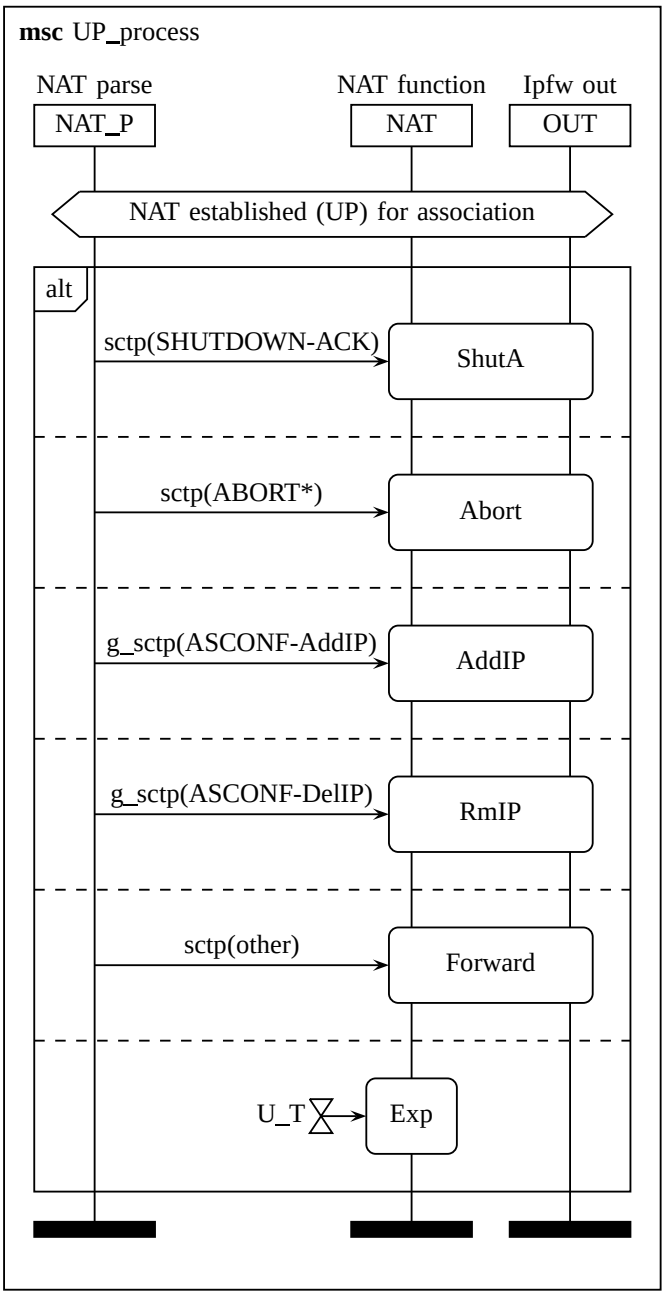


Fig. 14. NAT up for association

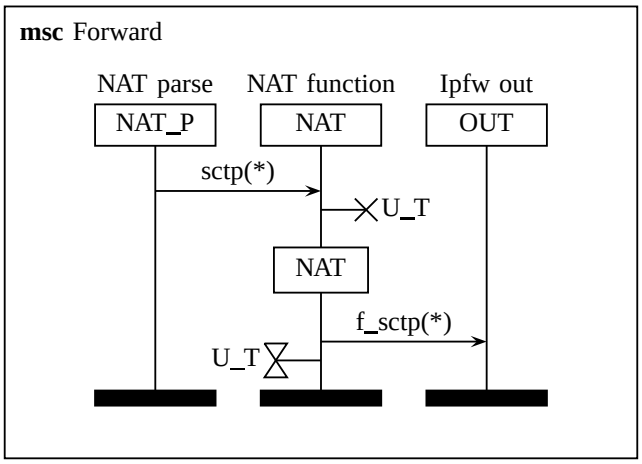


Fig. 15. Normal NAT after association is up

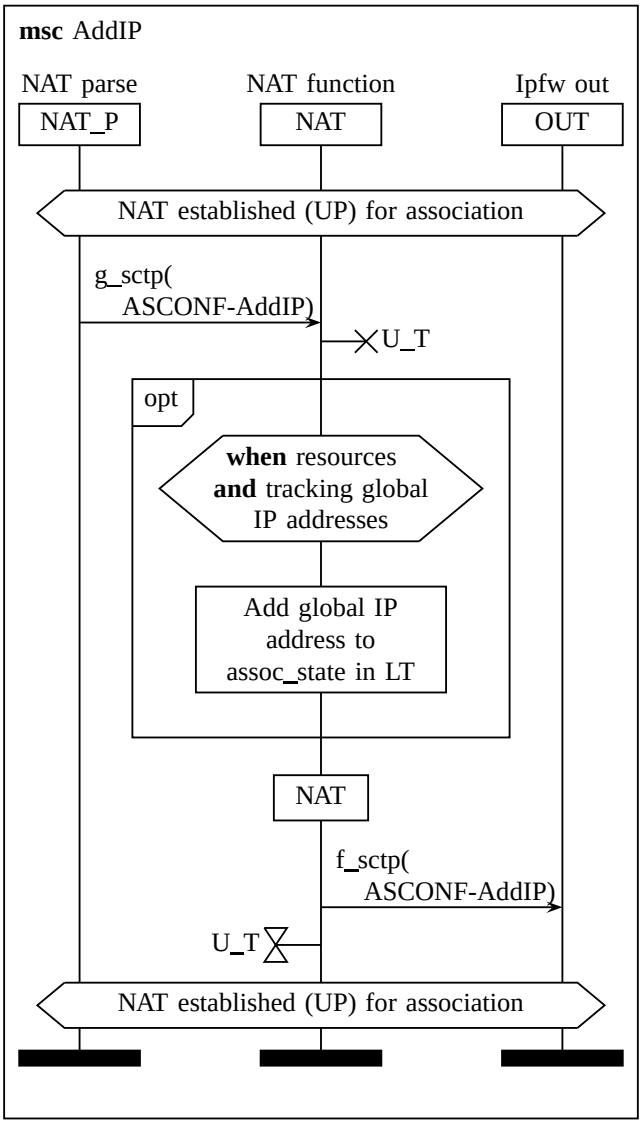


Fig. 16. Action on receipt of AddIpAck

ID. As discussed in section IV-B3, a NAT may not be able to obtain the necessary information to authenticate these chunks, and matching correlation-ID is resource intensive for the NAT.

A retransmitted **ASCONF-AddIP** or **ASCONF-DelIP**, will not present a problem to the NAT's state, since only addresses not already stored in `assoc_state` can be added, and addresses cannot be deleted more than once. There is potential for extraneous SCTP messages to be injected through the NAT by an at-

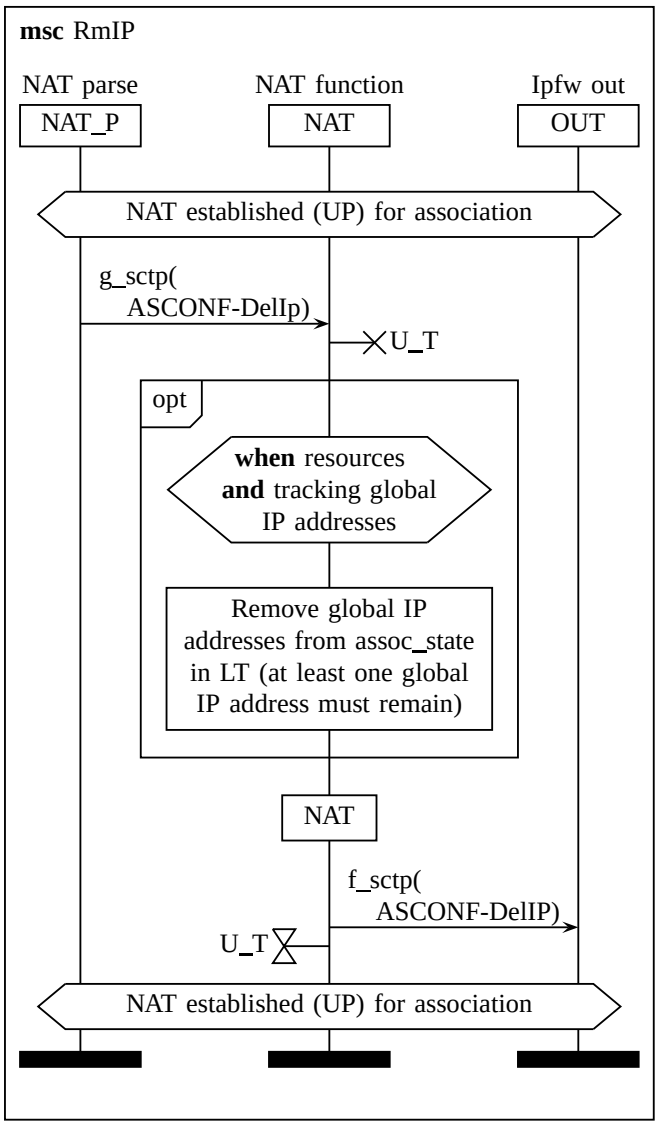


Fig. 17. Action on receipt of RmIp

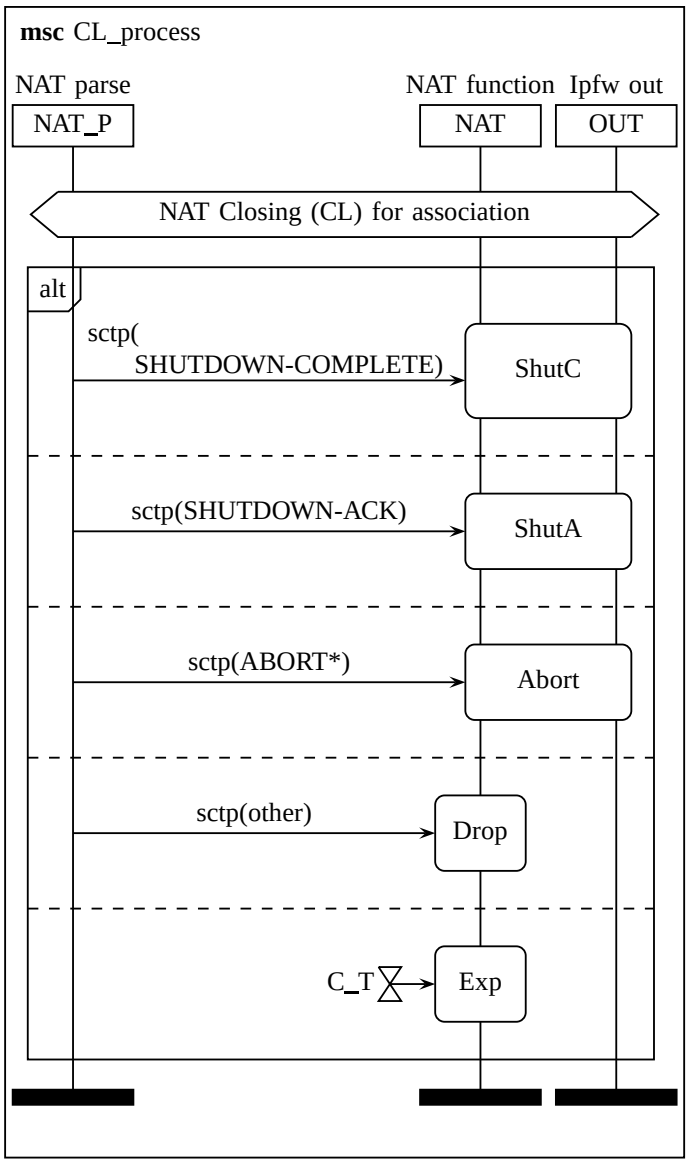


Fig. 18. NAT association closure

tacker, however the SCTP end hosts will recognise these as invalid as they maintain complete association state. An attacker removing IP addresses is more dangerous and could prevent the NAT from providing service to valid SCTP associations. To remove a valid IP address an attacker would need to guess or sniff the correct vtag and ports, and send the **ASCONF-DelIP** using an IP source address that matches a current global IP address in the `assoc_state` list. This problem will not occur if global IP addresses are not tracked.

D. Process of Closing an established NAT association (CL)

An SCTP association is normally released via a 3-way handshake of **SHUTDOWN**, **SHUTDOWN-ACK**, and **SHUTDOWN-COMPLETE** chunks. A **SHUTDOWN** chunk announces one endpoint's wish to terminate an SCTP association. The other endpoint may have more data to send. However, it is only when the NAT sees a **SHUTDOWN-ACK** chunk, that it starts to close support for that association.

The NAT enters the transitory Closing (CL) state upon receipt of a **SHUTDOWN-ACK** (see Figure 18). Within this state it will only will only allow retransmissions of the **SHUTDOWN-ACK** chunk (see Figure 19) or SCTP messages that contain chunks which will completely close the association: **ABORT** or **SHUTDOWN-COMPLETE** to pass through.

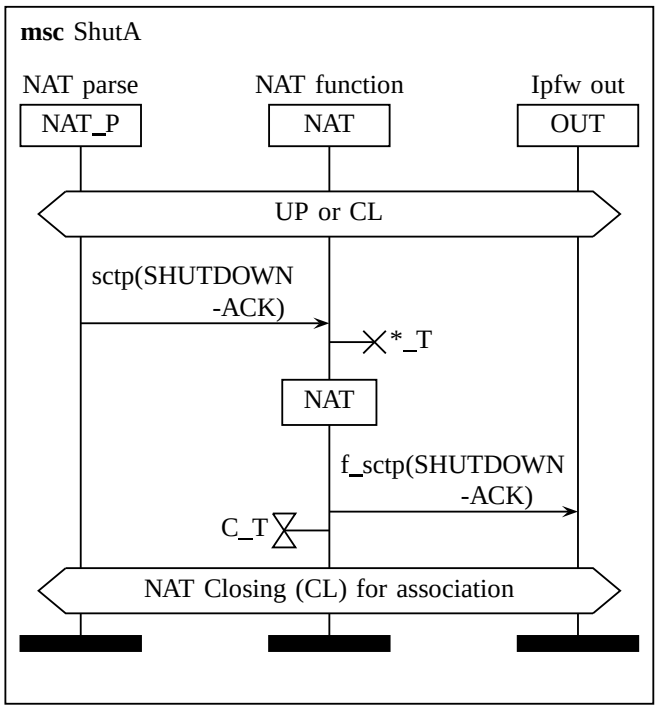


Fig. 19. Initiation of association closure under normal conditions

1) Graceful close:

The process of closing an association is completed when a **SHUTDOWN-COMPLETE** chunk is received (see figure 20).

As an alternative, the association may be allowed to continue for X_T time, to allow for end points to recover a lost **SHUTDOWN-COMPLETE** chunk. This is not enabled as a default, but is configurable (see section VII-B11).

2) Abortive close:

Figure 21 shows the processing of an **ABORT** chunk. An **Abort** has no response, so the association is immediately removed from the LT. An **ABORT** is processed by the NAT regardless of the state of the association.

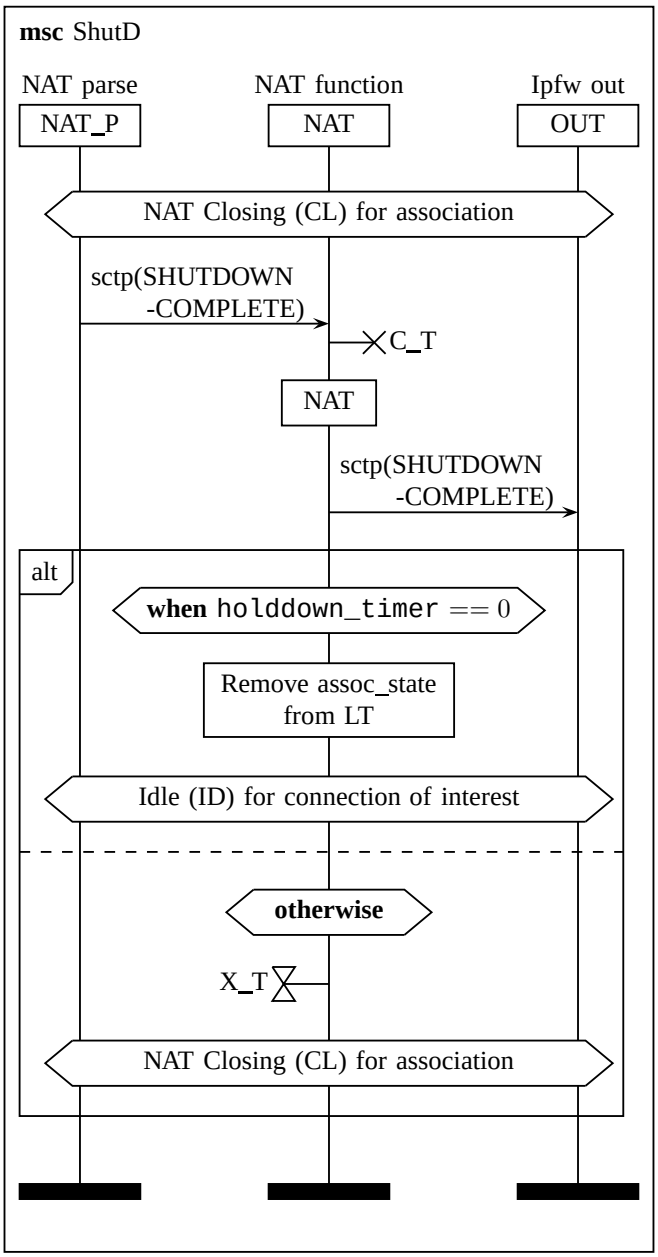


Fig. 20. Association closure complete normal conditions

E. Handling T-flag packets

If an SCTP entity receives an Out Of The Blue packet (OOTB), that is a packet for which it has no active association, it may under certain circumstances choose to NAT this packet.

If an SCTP message containing a **SHUTDOWN-COMPLETE** chunk has been lost in transit, the sender of the **SHUTDOWN-ACK** will time-out and resend the **SHUTDOWN-ACK**, however the receiver will have closed its association when it sent the lost **SHUTDOWN-COMPLETE**. On receipt of the retransmitted **SHUTDOWN-ACK**, the end host may respond with another **SHUTDOWN-COMPLETE**, but since it now does not know the correct *vtag* to use, it responds with its *vtag* equal to the *vtag* of the SCTP message with the **SHUTDOWN-ACK** chunk. The T flag is set to indicate that the *vtag* has been reflected.

In the unlikely event that there is ambiguity for the NAT as to how it should relay an incoming T-flag packet, the packet will be dropped. This is improbable, but possible if two or more associations using the same <*vtag, ports*> access different local IP addresses (if tracking global IP addresses, the global IP address is

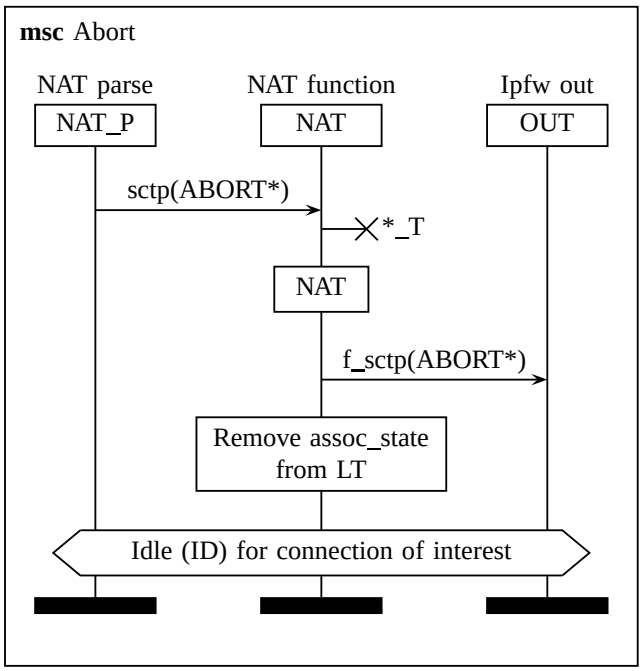


Fig. 21. Action on receipt of Abort

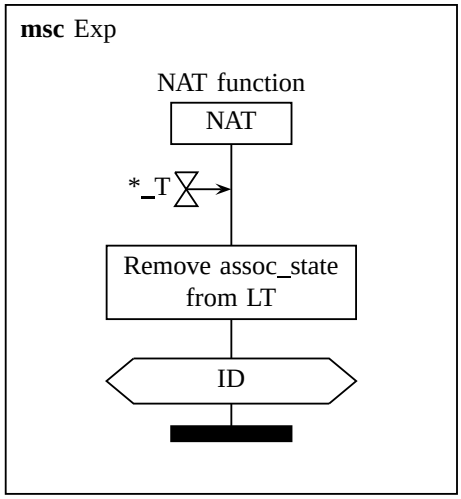


Fig. 22. Action on receipt of Abort

also matched).

V. SCTP NAT Timers

Four time-out values are used in the SCTP NAT mechanism:

- I_T for timing out the response to the **INIT** chunk,
- U_T for ensuring the association hasn't dropped out while up,
- C_T for timing out the response to the **SHUTDOWN-ACK** chunk,

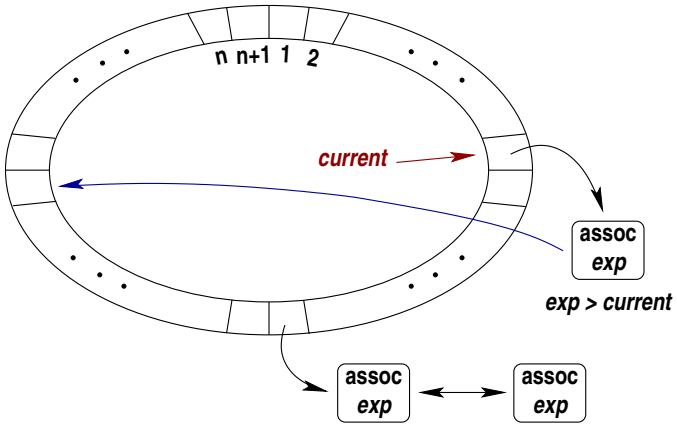


Fig. 23. Timer Q mechanism

- and X_T for holding the association in the CL state after the receipt of a **SHUTDOWN-COMPLETE** chunk to allow for retransmissions.

Upon receipt of any SCTP packet, the timer queue is checked. If any association's timer expires, the corresponding `assoc_state` is removed from the NAT, leaving the association in the idle state (ID) (see figure 22). Since only one timer is active at any one time, a single timer is used in `alias_sctp.c` with four possible values: I_T, U_T, C_T, and X_T.

A. Timer Queue

The timer Q is implemented as a circular Q of size greater than the maximum timer value in seconds. The timer queue is initialised so that the first element in the queue is the current time. As time progresses (in seconds) the table is stepped through (see Figure 23).

Each table entry is a linked list of associations that, at the time they were inserted, were due to expire at that particular second. Once a timeout has been set in the queue it will not be altered in the queue unless it has to be changed to a shorter time (typically only for aborts and closing). On a queue timeout, the real expiration time is checked, and if not less than or equal to the current time it is re-queued at its later time. This is especially important for normal packets sent during an association, since it means that while in the UP state, the timer queue is only altered every U_T (every few minutes) for a particular association, instead of for each packet received.

VI. SCTP NAT Lookup tables

The association state global and local look up tables are implemented as depicted in Figure 24. An incoming

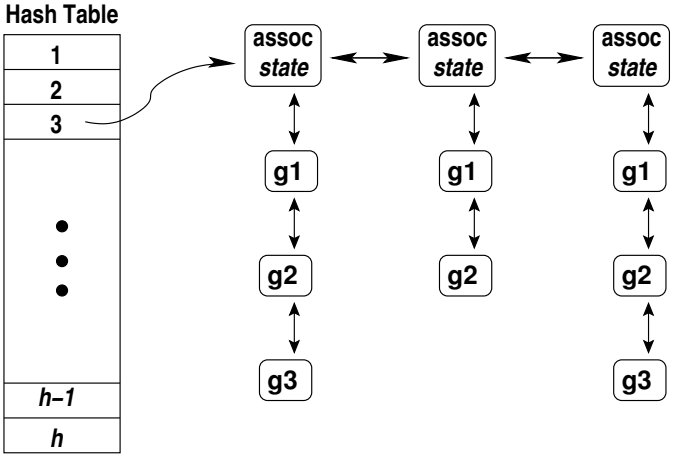


Fig. 24. Look up table structure

packet's $< vtag, ports, [addresses] >$ tuple is hashed, and the resulting linked list in the hash table linearly searched for a match to the association. `Assoc_state` contains links to the global and local tables as well as the TimerQ (see code segment 1). Global addresses, if tracked, are stored as a linked list within each association state and are searched linearly.

VII. FreeBSD implementation specific notes

Alias_sctp has been implemented in FreeBSD through the libalias kernel module. `Alias_sctp` uses its own look up tables, state space, and timers. It relies on `libalias` to pass a pointer to the IP packets that contain SCTP messages, and `ipfw_nat` for processing the associated SCTP NAT rules that have been configured using the `ipfw` command line interface.

A. Libalias interface

The SCTP NAT code is called from `LibAliasInLocked()` and `LibAliasOutLocked()` found in `alias.c`. The following outlines changes or additions to the libalias and ipfw source.

In `sys/netinet/libalias` :

1) `alias_sctp.h` – Additional file, header file for `alias_sctp.c`.
2) `alias_sctp.c` – Additional file, primary source code for the SCTP NAT
3) `alias.h` – define `_ALIAS_SCTP`
4) `alias.c` –
   - include `alias_sctp.h`
   - calls to functions in alias_sctp from `LibAliasIn/OutLocked()`
5) `alias_db.c` –
   - include `alias_sctp.h`
   - declaration and implementation of `SctpShowAliasStats()`
   - sctp additions to `ShowAliasStats()`
   - calls to `AliasSctpInit()` and `AliasSctpTerm()` from `LibAliasInit()` and `LibAliasUninit()`.
6) `alias_local.h` –
   - include `alias_sctp.h`
   - additions to `struct libalias`:

```
int sctpLinkCount;
struct sctp_nat_timer
                    sctpNatTimer;
u_int sctpNatTableSize;
LIST_HEAD(sctpNatTableL,
                  sctp_nat_assoc)
*sctpTableLocal;
LIST_HEAD(sctpNatTableG,
                  sctp_nat_assoc)
*sctpTableGlobal;
```

   - function prototypes:

```
void AliasSctpInit(
           struct libalias *la);
void AliasSctpTerm(
           struct libalias *la);
int SctpAliasIn(
            struct libalias *la,
                 struct ip *ip);
int SctpAliasOut(
            struct libalias *la,
                 struct ip *ip);
```

In `sys/netinet` :

1) `ip_fw_nat.c` – Modifications to allow **ERROR-M** and **ABORT-M** replies by modifying the passed IP packet.

In `sys/modules/libalias/libalias` :

1) `Makefile` – Addition of `alias_sctp.c` to the list of source files.

In `sbin/ipfw` :

1) `ipfw.c` – Modifications to the ipfw command line utility so that it recognises SCTP as a valid protocol for port based NAT rules. Also modifications so that ports cannot be mapped to different port numbers with SCTP.

In `share/man/man8` :

1) `alias_sctp.8` – new man page
2) `Makefile` – modifications to include new man page in build/install

B. Configurable parameters

Alias_sctp has a number of parameters that can be configured through the `sysctl` interface. They all have a prefix `net.inet.ip.alias.sctp.` Descriptions and current defaults for the parameters follow.

1) `hashtable_size` $= 2003$*:*
Size of hash tables used for NAT lookups $(100 <$ prime_number $> 1000001)$ This value sets the hash table size for any *future* created NAT instance and therefore must be set prior to creating a NAT instance (ie ipfw NAT 100 config ...).

The table sizes may be changed to suit specific needs. If there will be few concurrent associations, and memory is scarce, you may make these smaller. If there will be many thousands (or millions) of concurrent associations, you should make these larger. A prime number is best for the table size. The sysctl update function will adjust your input value to the next highest prime number.

2) `error_on_ootb` $= 1$*:*
Defines how the NAT responds to any Out-of-the-Blue (OOTB) packets. An OOTB packet is a packet that arrives with no existing association registered in the NAT AND does not contain an **INIT** or **ASCONF-AddIP**:

- $0-$ **ERROR-M** is never sent in response to OOTB packets
- $1-$ **ERROR-M** is only sent to OOTB packets received on the local side
- $2-$ **ERROR-M** is sent to the local side and on the global side ONLY if there is a partial match (ports and vtags match but the source global IP does not). This value is only useful if the NAT is tracking global IP addresses
- $3-$ **ERROR-M** is sent in response to all OOTB packets on both the local and global side (DoS risk)

We recommend setting this value to 1 to allow multi-homed local hosts to function with the NAT. If the SCTP stacks on the local side hosts do not support this feature, this value should be set to 0. If tracking global addresses we recommend setting this value to 2 to allow global hosts to be informed when they need to (re)send an **ASCONF-AddIP**. Value 3 should never be chosen (except for debugging) as the NAT will respond to all OOTB global packets, a DoS risk (see [6] for discussion).

3) `accept_global_ootb_addip` $= 0$*:*
Defines how the NAT responds to the receipt of a global OOTB **ASCONF-AddIP**:

- $0-$ No response (unless a partially matching association exists - ports and vtags match but global address does not)
- $1-$ NAT will accept and process all OOTB global **ASCONF-AddIP** chunks.

Option 1 should never be selected as this forms a security risk. An attacker can establish multiple fake associations by sending **ASCONF-AddIP**s.

4) `initialising_chunk_proc_limit` $= 2$*:*
Defines the maximum number of chunks in an SCTP packet that will be parsed when no existing association exists that matches that packet. Ideally this packet will only contain an **INIT** chunk or **ASCONF** with AUTH chunk. A higher value may become a DoS risk as malformed packets can consume processing resources.

5) `chunk_proc_limit` $= 5$*:*
Defines the maximum number of chunks in an SCTP packet that will be parsed for a packet that matches an existing association. This value is enforced to be $>=$ (initialising_chunk_proc_limit). As for the previous parameter, a high value is a DoS risk yet setting too low a value may result in important control chunks in the packet not being located and parsed.

6) `param_proc_limit` $= 25$*:*
Defines the maximum number of parameters within a chunk that will be parsed in a packet. As for other similar sysctl variables, larger values pose a DoS risk.

7) `track_global_addresses` $= 0$*:*
Enables/disables global IP address tracking within the NAT and places an upper limit on the number of addresses tracked for each association:

- $0-$ Global tracking is disabled
- $> 1-$ Enables tracking, the maximum number of addresses tracked for each association is limited to this value

This variable is fully dynamic, the new value will be adopted for all newly arriving associations, existing associations are treated as they were previously. Global tracking will decrease the number of collisions within the NAT at a cost of increased processing load, memory usage, complexity, and possible NAT state problems in complex networks with multiple NATs (see [6] for a full discussion). We recommend not tracking global IP addresses, this will still result in a fully functional NAT.

8) `init_timer` $= 15$*:*
Timeout value (s) while waiting for (**INIT-ACK — AddIP-ACK**). This value cannot be 0.

9) `up_timer` $= 300$*:*
Timeout value (s) to keep an association up with no traffic. This value cannot be 0.

10) `shutdown_time` = 15*:*
Timeout value (s) while waiting for **SHUTDOWN-COMPLETE**. This value cannot be 0.

11) `holddown_time` = 0*:* Hold association in table for this many seconds after receiving a **SHUTDOWN-COMPLETE**. This allows endpoints to correct shutdown gracefully if a shutdown_complete is lost and retransmissions are required. This may be a good option in high loss environments.

12) `log_level` = 0*:*
Level of detail in the system log messages (0 - minimal, 1 - event, 2 - info, 3 - detail, 4 - debug, 5 - max debug)

VIII. IPv6

Currently the `libalias` kernel module does not support IPv6, and hence neither does `alias_sctp`. When IPv6 support does become available, the following changes will need to be made to `alias_sctp`.

A. Address storage

We suggest that the `l_addr` and `a_addr` (see code segment 1) be stored as IPv6 structure with IPv4 addresses embedded when necessary (See [10]).

For the global IP address lists, there are two reasonable approaches:

1) Separate lists for IPv4 and IPv6 addresses. IPv4 addresses will only need 4 bytes of storage if the lists are separate. Also, only the relevant list will need to be searched to check for a matching address when tracking global IP addresses.
2) A single IPv6 list with embedded IPv4 addresses.

Two lists will result in slightly more complex code, but will save memory over a single list.

Note: `alias_sctp` does not store multiple local IP addresses.

B. Parsing

IP packets will need to be checked to determine whether they are IPv4 or IPv6, and the source and destination addresses extracted and stored accordingly. Also, currently `alias_sctp` only searches **INIT**, **INIT-ACK**, and **ASCONF** chunks for IPv4 addresses. An additional test for IPv6 addresses will need to be added.

C. NAT

Addresses will need to be translated to the correct IPv4 or IPv6 address. Incoming IPv4 packets to local IPv6 destinations or IPv6 packets to local IPv4 destinations should be dropped. The SCTP protocol should ensure that this does not happen.

IX. Conclusion

Alias_sctp version 0.2 provides a fully functional SCTP NAT within the Freebsd ipfw/libalias framework. In particular:

- Modifications to the libalias kernel module to allow network address translation of SCTP packets
- NAT based on unique verification tag and port numbers.
- Support for global multi-homing
- Support for T-flagged packets
- Logging and statistics
- Global IP address tracking and management
- Port forwarding
- Supports NAT specific SCTP enhancements recommended in [5] and [7]
- Dynamic parameter configuration through the **sysctl** interface

Patches to Freebsd 8.X are available from [11].

Acknowledgements

The development of the SCTP NAT enhancements to *libalias* is part of the SONATA [1] project and was made possible in part by a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley.

The project benefits from the people and facilities of CAIA. Of special note are Lawrence Stewart for his help with svn and freebsd, and CAIA director Grenville Armitage.

References

[1] CAIA, "SONATA SCTP over NAT adaptation," viewed 12 June 2008. [Online]. Available: http://caia.swin.edu.au/urp/sonata

[2] D. A. Hayes and J. But, "Alias_sctp version 0.1: SCTP NAT implementation in IPFW," CAIA, Swinburne University, Tech. Rep. 080618A, Jun. 2008. [Online]. Available: http://caia.swin.edu.au/reports/080618A/CAIA-TR-080618A.pdf

[3] R. Stewart, "Stream control transmission protocol," IETF, RFC 4960, Sep. 2007.

[4] Z.120, *Message Sequence Chart (MSC)*, ITU-T, May 2004.

[5] R. Stewart and M. Tüxen, "Stream control transmission protocol (SCTP) network address translation," Internet-Draft, Jul. 2008.

[6] D. A. Hayes, J. But, and G. Armitage, "Issues with network address translation for SCTP," *CM SIGCOMM Computer Communication Review*, vol. 39, no. 1, Jan. 2009, (accepted Oct.08, to be published).

[7] M. Tüxen, I. Rüngeler, R. Stewart, and E. P. Rathgeb, "Network address translation (NAT) for the stream control transmission protocol (SCTP)," *IEEE Network*, vol. 22, no. 5, pp. 26–32, September/October 2008.

[8] R. Stewart, Q. Xie, M. Tüxen, and S. Maruyama, "Stream control transmission protocol (SCTP) dynamic address reconfiguration," IETF, RFC 5061, Sep. 2007.

[9] M. Tuexen, R. Stewart, P. Lei, and E. Rescorla, "Authenticated chunks for the stream control transmission protocol (SCTP)," IETF, RFC 4895, Aug. 2007.

[10] R. Hinden and S. Deering, "IP version 6 addressing architecture," IETF, RFC 2373, Jul. 1998.

[11] D. A. Hayes and J. But, "Alias_sctp NAT module," viewed 24 November 2008. [Online]. Available: http://caia.swin.edu.au/urp/sonata/downloads.html