Alias_sctp Version 0.1: SCTP NAT implementation in IPFW

David Hayes, Jason But
Centre for Advanced Internet Architectures, Technical Report 080618A
Swinburne University of Technology
Melbourne, Australia
dahayes@swin.edu.au, jbut@swin.edu.au

***Abstract*—Alias_sctp is part 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). Traditional address and port number look ups are inadequate for SCTP's operation. A design of the operational states and packet handling necessary to achieve SCTP functionality are outlined along with the necessary modifications required to integrate it into FreeBSD's ipfw/libalias NAT system. Alias_sctp version 0.1 uses unique verification tag and port numbers for NAT, providing support of global multi-homing, T-flagged packets, logging and statistics, timing, and two association shutdown modes.**

I. Introduction

The Stream Control Transmission Protocol (SCTP)[2] 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.

A standard TCP/UDP type NAT implementation has a number of problems:

- SCTP's checksum is an improvement on TCP's, but requires calculation over the entire packet if any changes are made (such as port number).
  - This increase in work may cause problems in low-end home routers.
- SCTP allows multi-homing. Endpoints may have more than one IP address, providing fault tolerance. Also, there can be a number of associations sharing the same address and port number.
  - Traditional address and port number table look ups are therefore inadequate.

SCTP has a Verification tag (Vtag), which is used to differentiate associations. It is randomly generated by the end-points during the association initialisation process, making it an ideal index for an association look up table. In this work, we use a combination of Vtag and port number to identify associations. This makes the table look up global address independent, allowing multi-homing on the outside of the NAT. We also leave port numbers unchanged, so that there is no need to recalculate SCTP's checksum.

This report outlines the SCTP NAT design. Following a description of alias_sctp's operational states and packet handling mechanisms is short discussion on T-flagged packets and multi-homing. The report then describes the implementation in FreeBSD's libalias module and concludes with a summary of achievements and further work.

II. Alias_sctp operational states packet processing

The operational states and packet processing sequences in alias_sctp are described using ITU-T Message Sequence Charts (MSC) [3]. In these diagrams it is assumed the firewall will only pass packets that are permitted to be address translated.

Figure 1 gives an overview of the SCTP NAT. Packets arriving are first passed and checked. Following this, the resulting SCTP messages are processed according to the recorded state of the particular SCTP association. Table I describes some of the notation used in the MSCs.

A. Packet parser

Arriving packets are first parsed to obtain their details (see figure 2). The association database (DB) is then checked to see if there is a current association for this packet.

The following items will be stored in a C structure:

1) The SCTP control chunk type (ie Init, InitAck, etc or Other)
2) pointer to packet

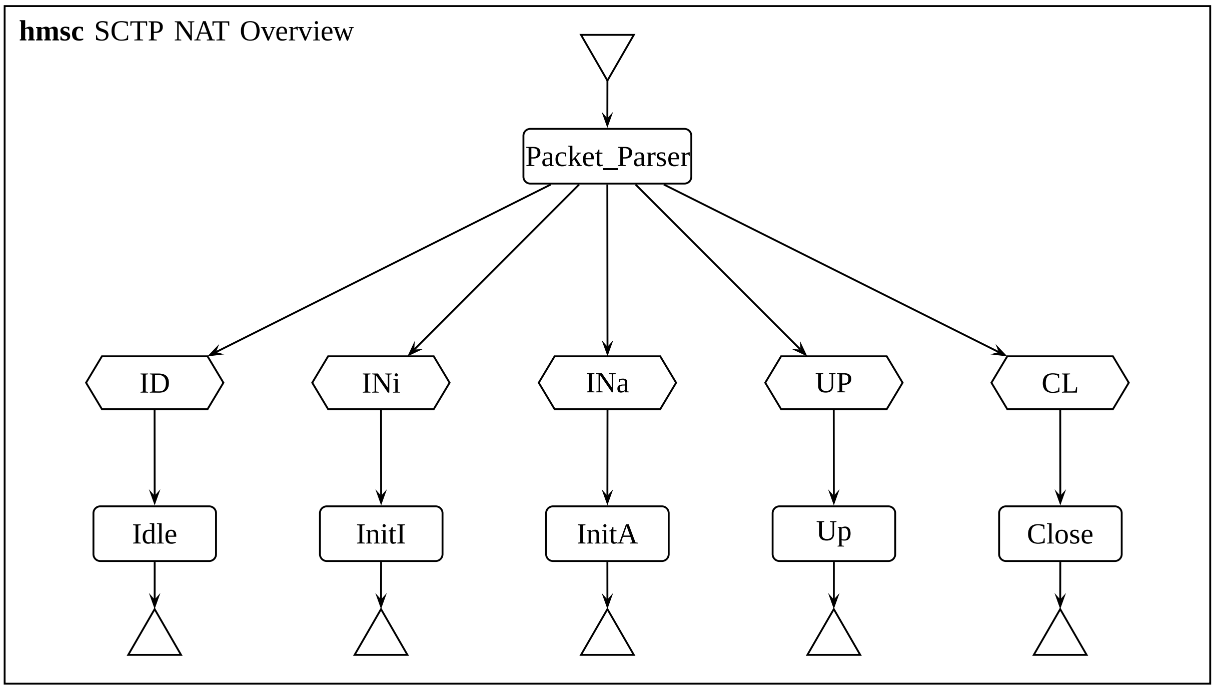

Fig. 1. SCTP NAT overview

sctp(_)	any sctp message
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,gTg	Set appropriate local or global verification tag
ITg	Initiation Tag
__T	Any timer

TABLE I
MSC SCTP MESSAGE SYNTAX

3) pointer to IP header (to access source and destination IP addresses)
4) pointer to SCTP common header (to access Vtag and ports)
5) pointer to chunk being processed

If more than one of the target chunks is present in the packet, only the highest priority chunk type is to be stored and processed. The higher the number the lower the priority:

1) Init (No other chunks are allowed)
2) Abort (Other chunks are meaningless)
3) InitAck (No other chunks are allowed)
4) ShutAck (Other chunks are meaningless)
5) ShutComp (No other chunks are allowed)
6) local AddIp through NAT with no association (Other chunks are meaningless)
7) global AddIpAck through NAT while in INa (Other chunks are meaningless)

When processing the different chunks additional information will need to be extracted. Once the packet is parsed, the action taken depends on the current state of the association in the NAT. These actions are outlined in the following sections.

B. No current association – Idle (ID)

If there is no current association, the state for the arriving message is Idle (ID) (see figure 3).Verification Tags and ports are used to index (or hash) the database. As such, in the very small probability that there is a collision of these values, the NAT should reply with an **Abort**[1] chunk with the M-bit set to indicate that it was generated by the middle box [4]. Subsequent reissuing of the **Init** will have a different tag and solve the rare collision condition. (Note an **Init** has VT=0).

Under high load or error conditions the NAT may have no resources available to add the new association. In such

[1]Version 0.1 does not actually send the Abort packet, only writing that it should be sent in the log. Future versions will send the AbortM packet

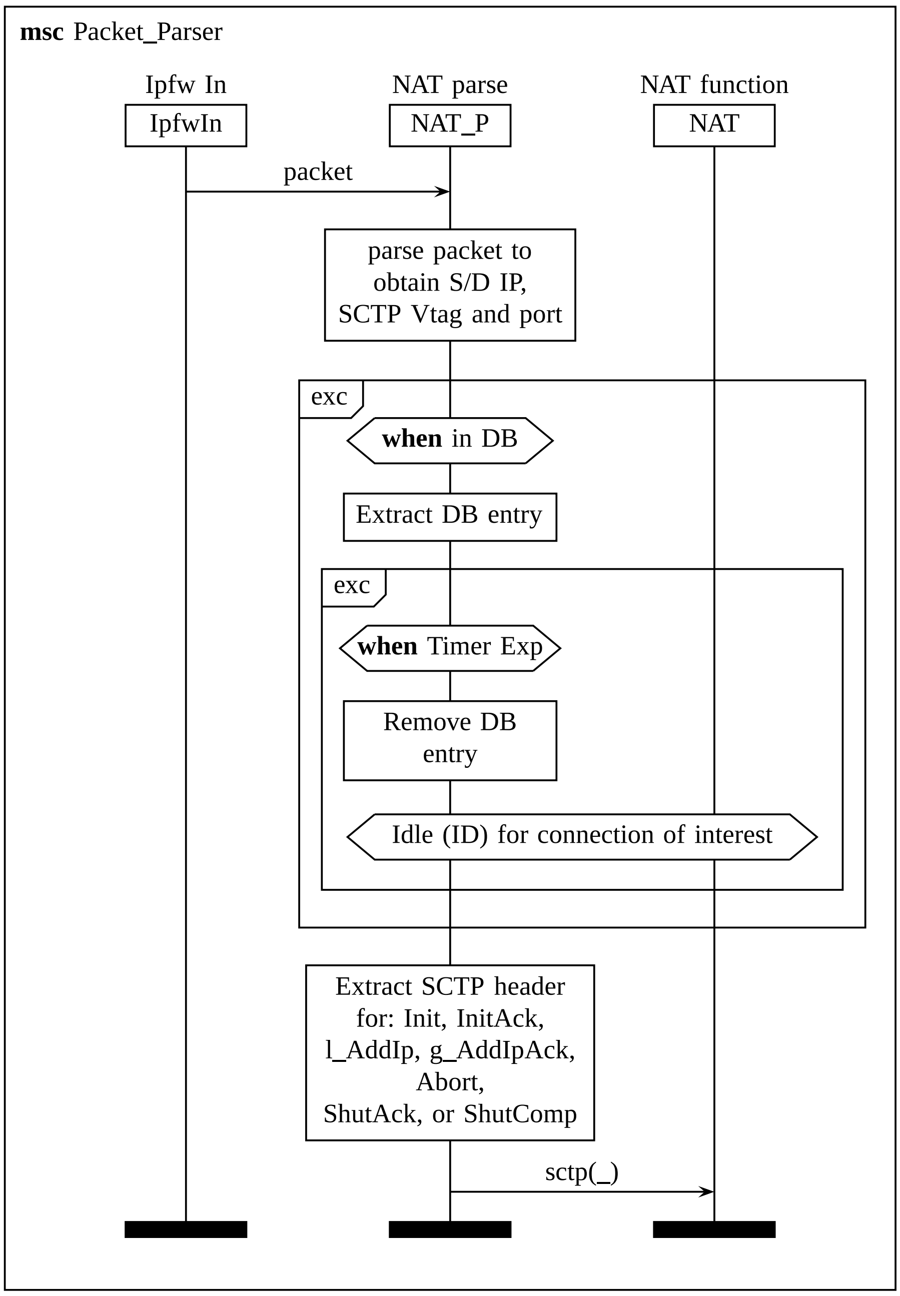
msc Packet_Parser
Ipfw In
IpfwIn
NAT parse
NAT_P
NAT function
NAT
packet
parse packet to obtain S/D IP, SCTP Vtag and port
exc
when in DB
Extract DB entry
exc
when Timer Exp
Remove DB entry
Idle (ID) for connection of interest
Extract SCTP header for: Init, InitAck, l_AddIp, g_AddIpAck, Abort, ShutAck, or ShutComp
sctp(_)

Fig. 2. Parsing and checking of an arriving packet

circumstances it will respond with an **ErrorM**[2] message indicating that it is out of resources.

The DB will need to store the following:

- Local Vtag and port
- Global Vtag and port
- Local IP address
- State [ID—INi—INa—UP—CL]
- Timer expiration time

[2]Alias_sctp version 0.1 does not actually send the Error packet, only writing that it should be sent in the log. Future versions will send the ErrorM packet

Only one local IP address will be stored. No global IP addresses will be stored in the NAT. Look up is based on Vtag/port, if these are correct the NAT will forward the packet to a local IP address regardless of where it came from. The local host may check the source address, or a firewall may block addresses.

An extension to the SCTP protocol defined in [5] allows for dynamic address reconfiguration with **ASCONF** chunks. Of particular interest to the NAT are the

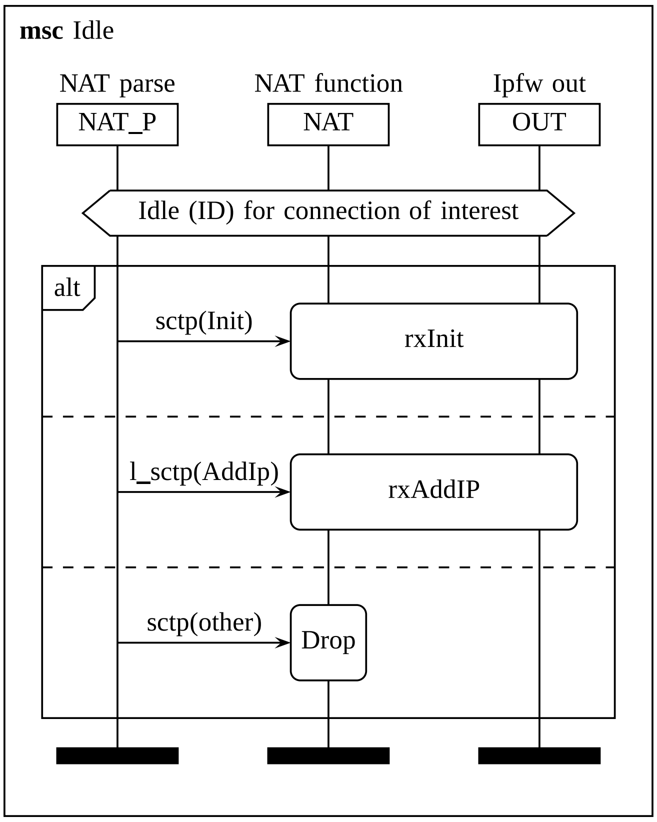

Fig. 3. Initiate dynamic NAT rule

AddIp and **AddIpAck**. To cope with a machine that has multiple interfaces connected to multiple NAT devices, a NAT may receive an **AddIp** message even though it does not have a current association for the originating IP address [4]. To enable this to work, an **AddIp** message coming from a local machine is treated in a similar way to an **Init** message to establish an association in the NAT.

Two key differences with the AddIp in this context are:

- The AddIp SCTP chunk will not contain an IP address, the source IP address will be used by the destination instead.
- The AddIp chunk will contain both Vtags, so the DB entry can be completely filled. (It is not consider active until the AddIpAck is received)

Figures 4 and 5 give the details of how an **Init** and **AddIp** messages are handled respectively.

C. Initialising the association (INi and INa)

After an **Init** or local **AddIp** message has been received an **InitAck** (or **AddIpAck**) message is required for the association to be fully established in the NAT. Figure 6 gives an overview of this process, with details given in figures 8 and 9.

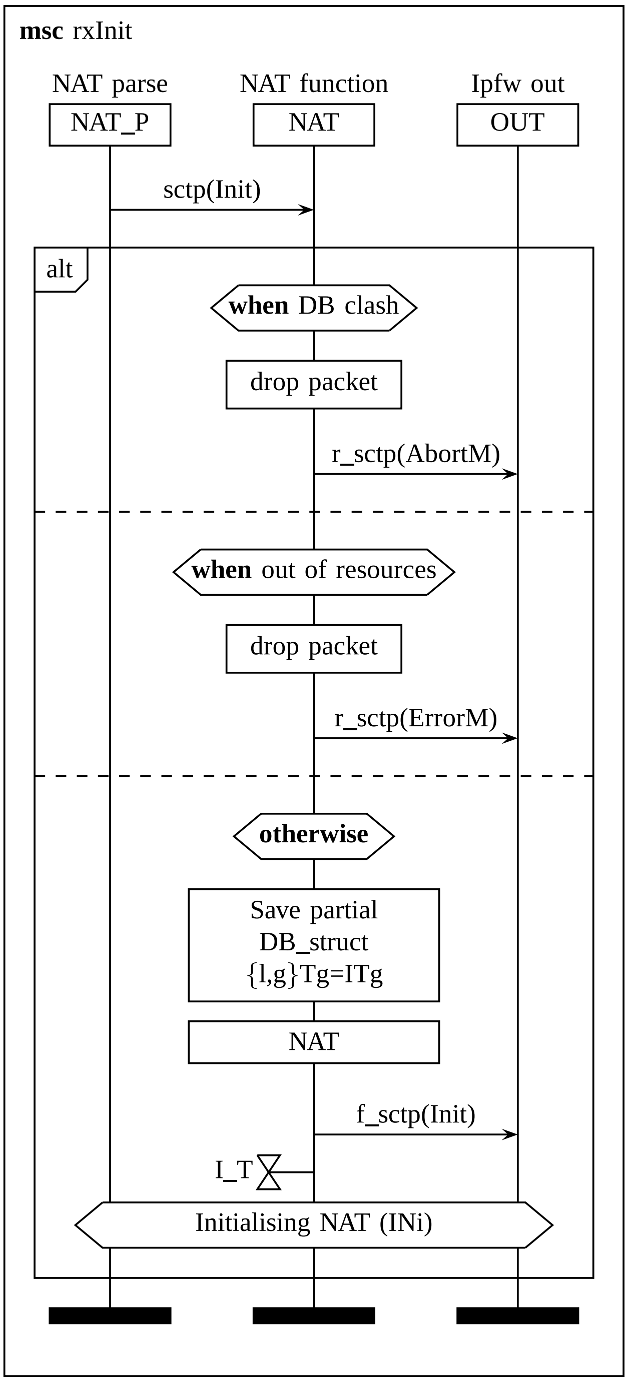

Fig. 4. Initiate dynamic NAT rule

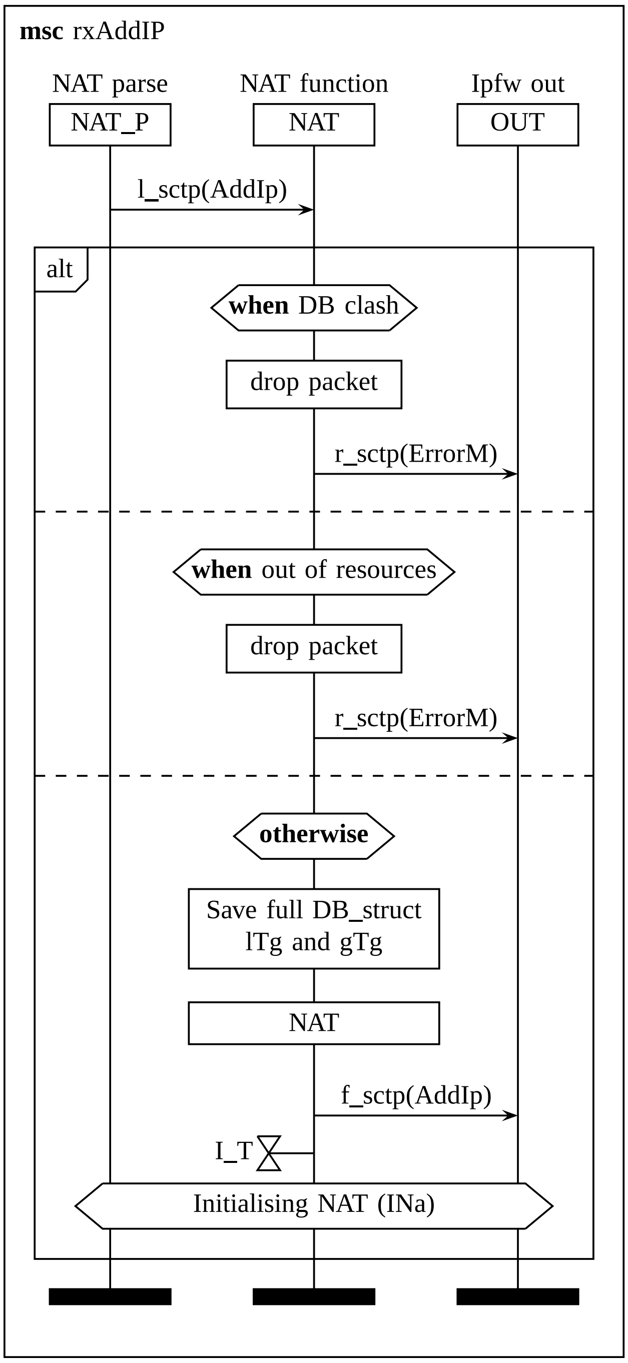

Fig. 5. Initiate dynamic NAT rule from AddIp

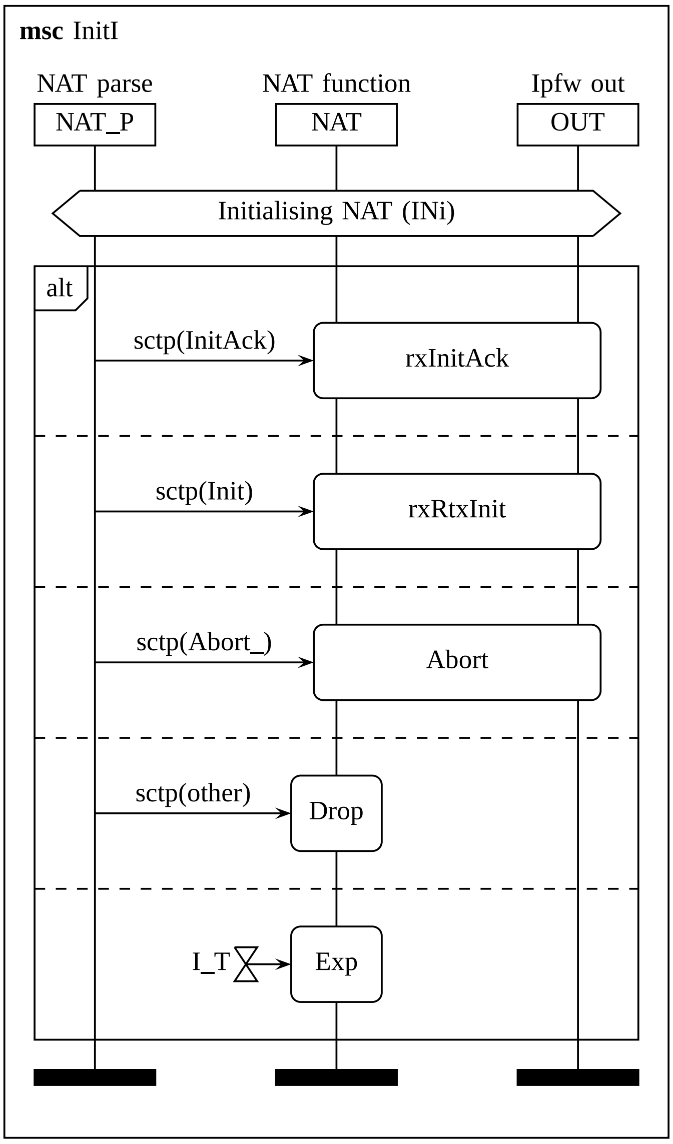

Fig. 6. Initialising NAT (**Init** based)

If a relayed **Init** or **AddIp** message failed to reach its destination, the sender will retransmit it on timeout. The NAT could simply relay this message, however, if there was some change in the messages it might corrupt the representation of the association in DB. To be safe the DB entry is rebuilt (see figures 4 and 5).

An **Abort** message will cause the NAT state to go back to **ID**. Any other messages received are not valid in this context, and will be dropped. SCTP does define an error message that could be sent, however it is possible that all other SCTP messages received at this point could be part of some sort of malicious attack, so simply dropping invalid packets prevents the NAT adding to the problem.

D. NAT for association is UP

After the preceding initialisation procedure, the association is considered to be up as far as NAT is concerned. The cookie exchange and subsequent data can be passed normally. The association may be modified by

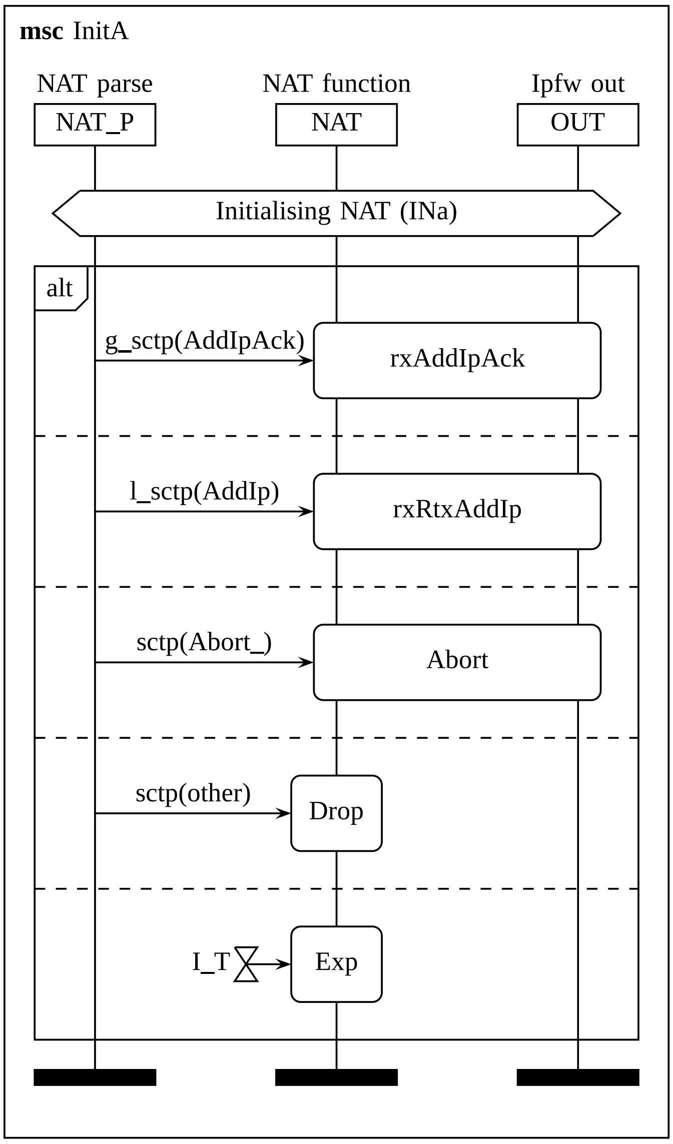

Fig. 7. Initialising NAT (**AddIp** based)

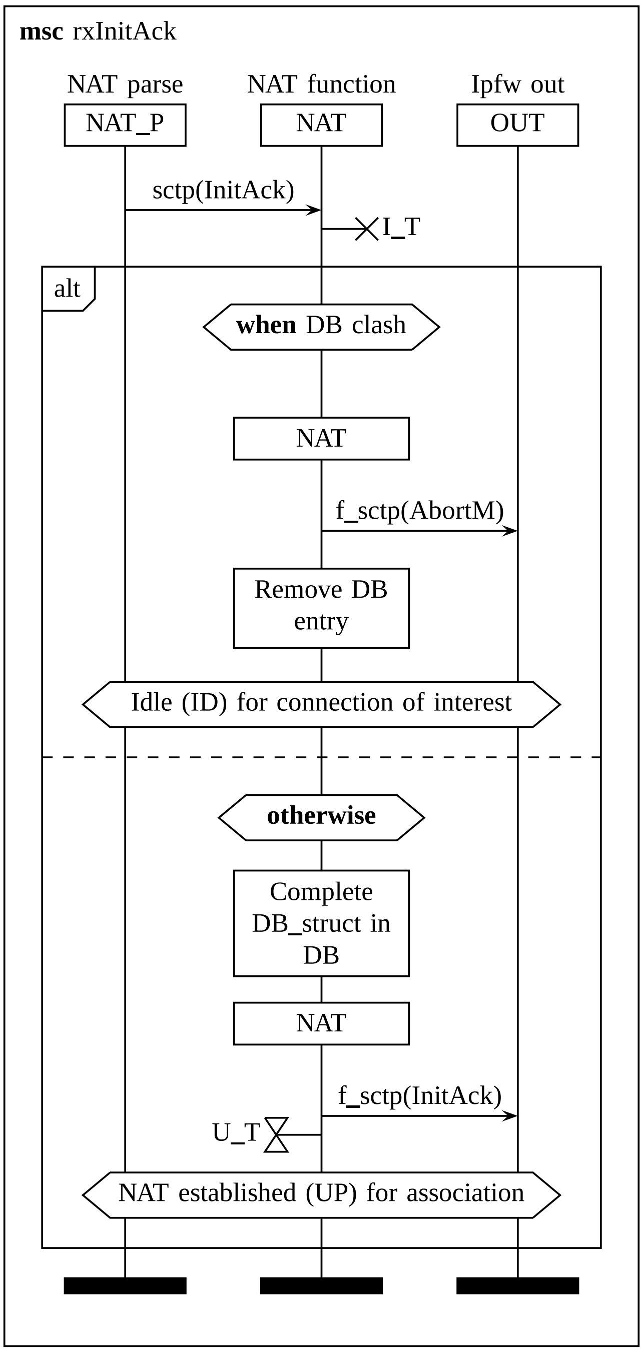

Fig. 8. Receiving an **InitAck** while in INi

the **AddIpAck** or **RmIp** messages[3]. The association may be disconnected through the receipt of a **ShutAck**, or an **Abort** or timeout (see figure 12).

Figure 13 shows the normal NAT operation for forwarding SCTP packets. The UP timer (U_T) is set on the receipt of each packet.

E. Closing and established NAT association

An SCTP association is normally released via a 3-way handshake of shutdown, shutdown-ack, and shutdown-complete messages. A shutdown message announces one endpoint's wish to terminate a SCTP association. The other endpoint may have more data to send, so it is only when the NAT sees a shutdown-ack message, that it starts to close support for that association.

The process of closing an association is completed when a shutdown complete message (**ShutComp**) message is received (see figure 14.

As an alternative, the association may be allowed to

[3]Alias_sctp version 0.1 does not support this feature in the NAT. The AddIpAck and RmIp messages will be passed for the endpoints to deal with

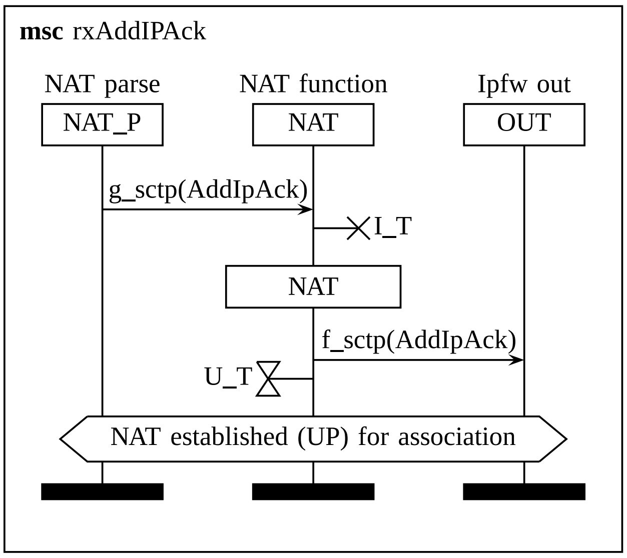

Fig. 9. Receiving an **AddIPAck** while in INa

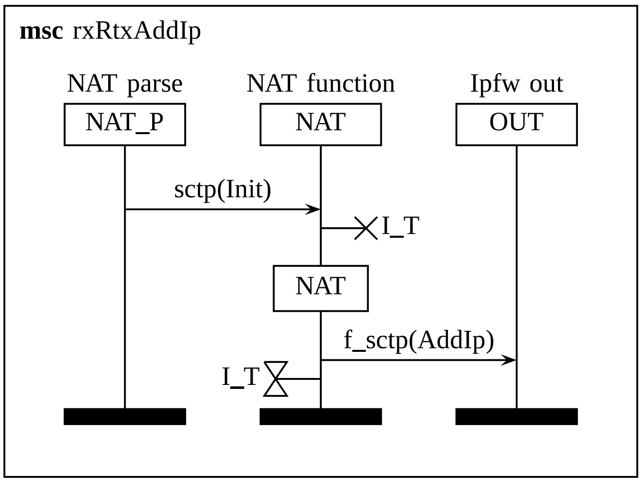

Fig. 11. Receiving a retransmitted **AddIp** while in INa

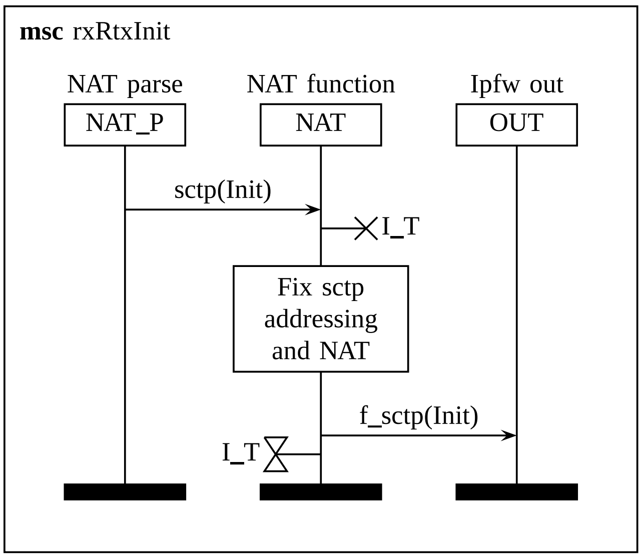

Fig. 10. Receiving a retransmitted **Init** while in INi

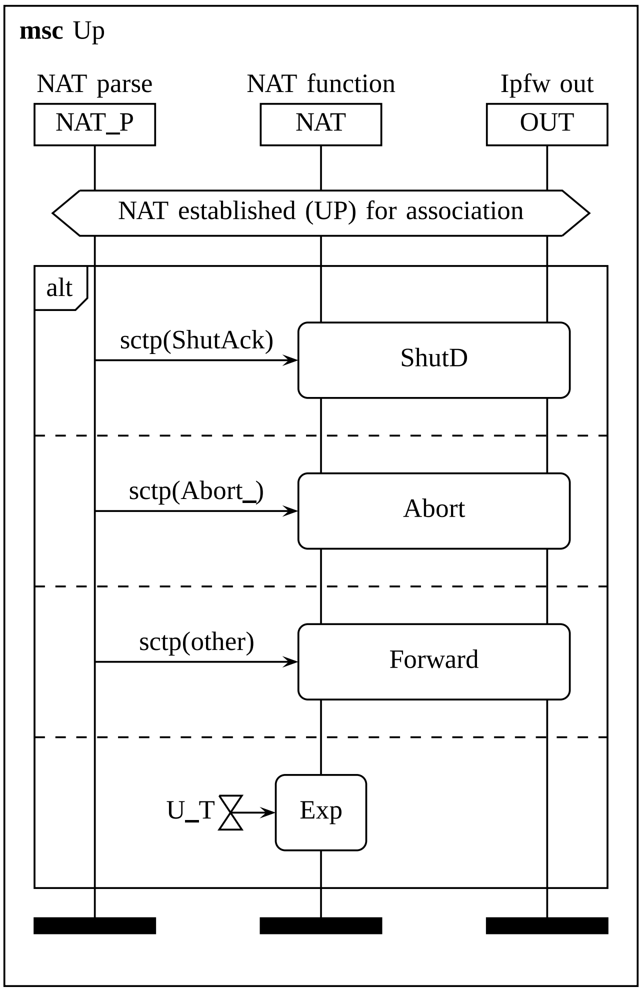

Fig. 12. NAT up for association

continue for X_T time, to allow for end points to recover a lost **ShutComp** message. This is not enabled as a default, but is configurable currently through `#define SN_SHUTDOWN > 0`.

Figure 16 shows the response to an **Abort** message. An **Abort** has no response, so the association is immediately removed from the DB.

F. Timers

Three timer values are used in the SCTP NAT mechanism: I_T for timing the response to the **Init** message, U_T for ensuring the association hasn't dropped out while up, and C_T for timing the response to the

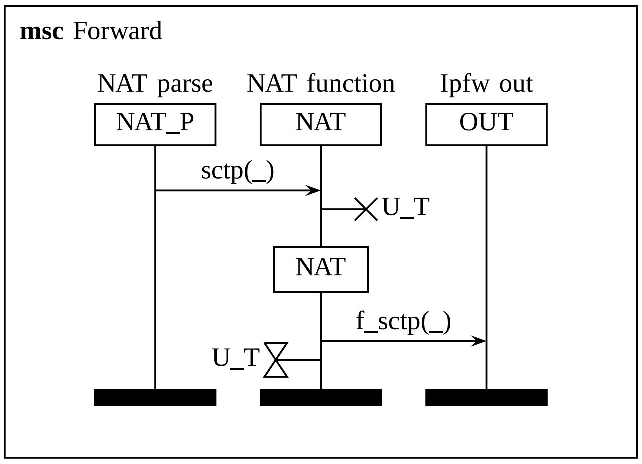

Fig. 13. Normal NAT after association is up

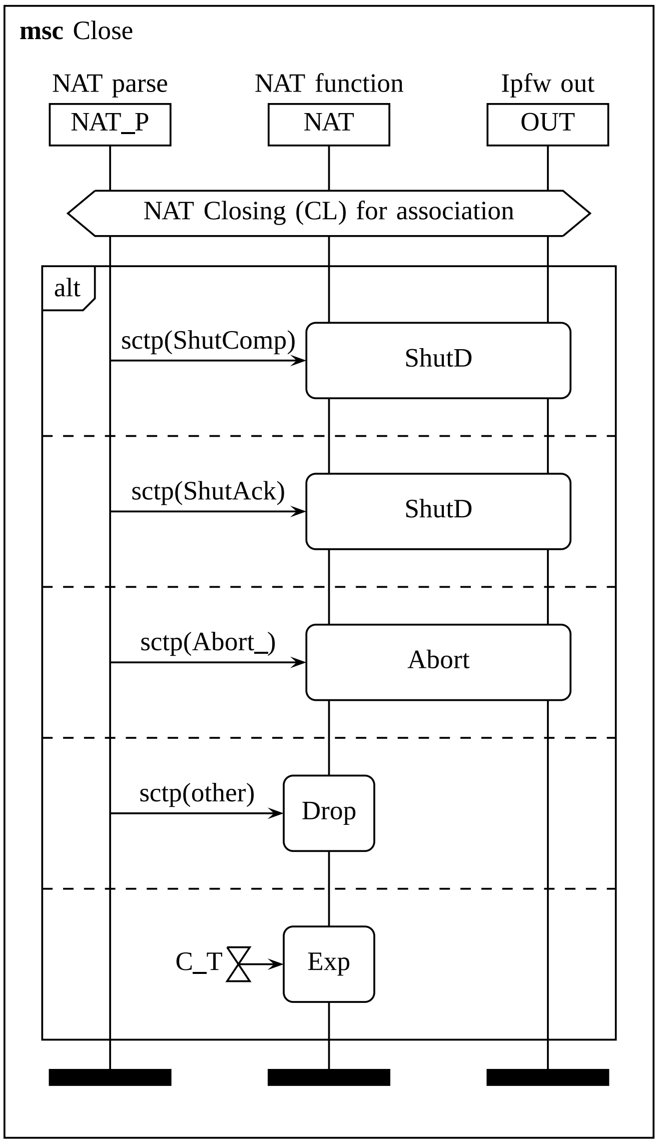

Fig. 14. NAT association closure

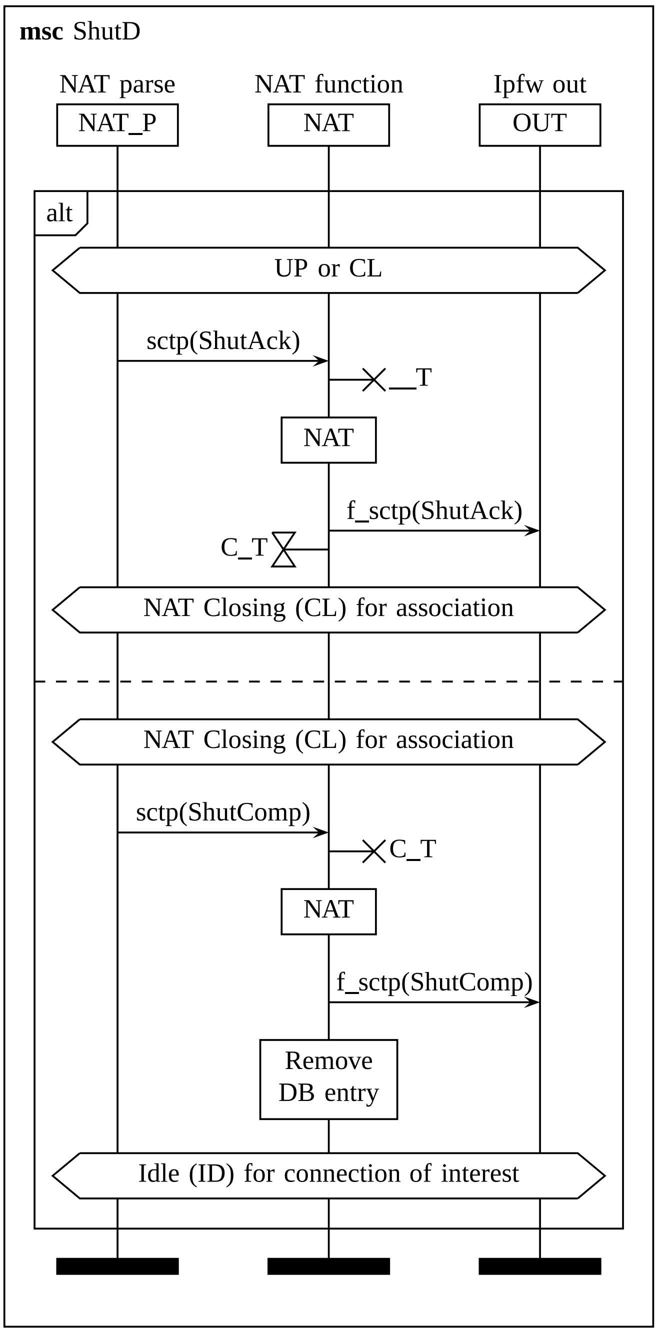

Fig. 15. NAT release under normal conditions

ShutAck message. The timer is checked on the receipt of any SCTP packet directed to the particular NAT instance. Since only one timer is active at any one time, a single timer is used in `alias_sctp.c` with three possible values: I_T, U_T, and C_T. If any timer expires, the association is removed and the NAT moves to the idle state (ID) for that association (see figure 18).

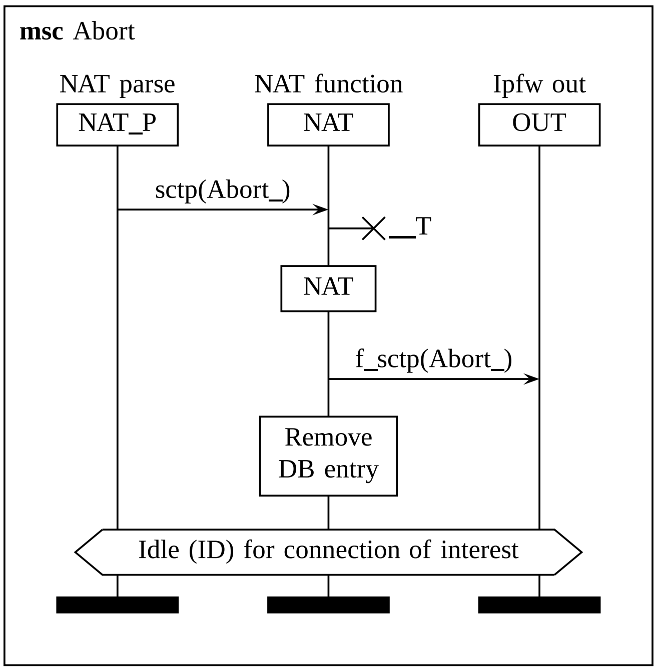

Fig. 16. Action on receipt of Abort

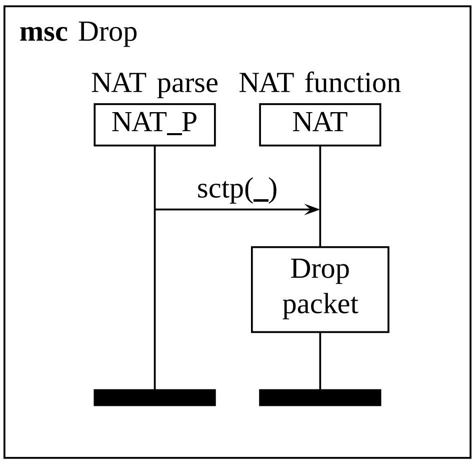

Fig. 17. Action on receipt of an invalid SCTP message

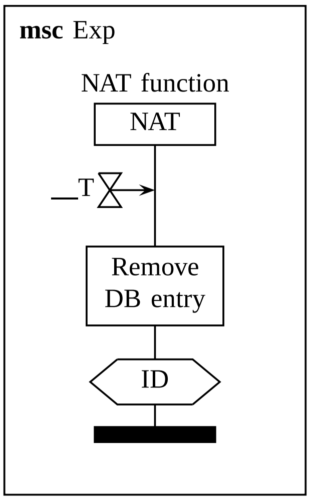

Fig. 18. Action on receipt of Abort

G. 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 respond to this packet.

If a ShutComp packet has been lost in transit. The sender of the ShutAck will time-out and resend the ShutAck, however the receiver will have closed its association when it sent the lost ShutComp. The SCTP RFC states that the receiver of the ShutAck may respond with a ShutComp with its Vtag equal to the Vtag of the ShutAck (The receiver has closed its association and therefore does not know the correct Vtag to use).

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 use the same Vtag and port to access different local IP addresses.

H. Multi-homing

Multi-homing is a SCTP feature that allows end points in an association to register all IP addresses that they may have. In the event of the path to one of the IP addresses failing, the association may then use one of the other registered IP addresses to reach the endpoint.

Mulit-homing can present a problem when one or both of the machines in the association are behind a NAT. [4] suggest the following mechanism for handling these situations:

1) SCTP entities must only advertise their global addresses inside the **Init** or **InitAck** messages during the initialisation process.
2) SCTP entities may use an AddIp ASCONF chunk to add additional local interfaces which may be connected via NATs. An **AddIp** may be sent out each of these interfaces after the association has been established to enable these possible paths.

This revision to the SCTP protocol simplifies the NAT mechanism. Since no local addresses are stored in association initialisation packets there is no need to rewrite these with global address, necessitating the recalculation of the checksum.

The current design does not store the global IP addresses. This means that multi-homing on a host outside the NAT will work without the need to examine and store addresses in **Init**, **InitAck** and ASCONF messages.

III. Implementation Notes

The implementation is in general as described in the message sequence charts. The SCTP NAT code is called from `LibAliasInLocked()` and `LibAliasOutLocked()` found in alias.c.

A. Libalias interface

Libalias has a facility for allowing the simple addition of support for new protocols through *libalias* modules. Handlers for new protocols are found through the `find_handler()` function. Unfortunately, `find_handler()` is only called for UDP, TCP, and GRE packets. This facility is mainly for additional application layer protocol, and SCTP is a transport layer protocol. Even with modifications, searching the handler list is slow compared to directly calling the SCTP NAT functions.

For this work we will keep the SCTP functionality as separate as possible, but it will need to be part of the main libalias module. Changes are enclosed in `#ifdef _ALIAS_SCTP` statements. Changes are as follows:

1) *`alias_sctp.h`:* Additional file.

2) *`alias_sctp.c`:* Additional file.

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;
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);
```

B. Timer Q

The timer Q is implemented as a table of size greater than the maximum timer value in seconds. The timer queue is initialised so that the first element in the table is the current time. As time progresses (in seconds) the table is stepped through. When the end is reached, the first element will be next, making it a circular queue.

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 (usually 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 UP state, the timing queue is only altered every U_T (every few minutes) for a particular association.

IV. Conclusion and further work

Alias_sctp version 0.1 provides a functional SCTP NAT in 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 (single NAT router);
- Support for T-flagged packets;
- Logging and statistics;
- Association timer queue;
- Association removal on shutdown, or on timeout after shutdown.

Subsequent versions of alias_sctp will include:

- Support for the new ASCONF AddIP address configuration packets. This will add support for complex multi-homing configurations. Currently the nat should work for a multi-homed server on the outside (public/global) side of the nat, though this has yet to be fully tested.

- Support for sending AbortM and ErrorM packets in response to vtag/port collisions. (At the moment these are just writen to the log file)
- Port forwarding
- There are three main SCTP specific options:
  - When to close an association (see section II-E).
  - Hash table size.
  - Debug level.

  Currently these are configured at compile time through `#define` statements. This will be enhanced so that they are accessible through the sysctl interface
- Speed Optimisation
- IPv6 support
- Global IP address tracking and management.

Acknowledgements

The development of the SCTP NAT enhancements to *libalias* is part of the SONATA [1] project and was made possible through the funding of Cisco.

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] R. Stewart, “Stream control transmission protocol,” IETF, RFC 4960, Sep. 2007.

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

[4] M. Tüxen, I. Rüngeler, R. Stewart, and E. Rathgeb, “Network address translation (NAT) for the stream control transmission protocol (SCTP),” in *Submitted to the IEEE Network Special Issue on Implications and Control of Middleboxes in the Internet*, 2008.

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