Issues with Network Address Translation for SCTP

David A. Hayes
dahayes@swin.edu.au

Jason But
jbut@swin.edu.au

Grenville Armitage
garmitage@swin.edu.au

Centre for Advanced Internet Architectures (CAIA)
Swinburne University of Technology
PO Box 218
Hawthorn, Victoria 3122, Australia

ABSTRACT

A Stream Control Transmission Protocol (SCTP) capable Network Address Translation (NAT) device is necessary to support the wider deployment of the SCTP protocol. The key issues for an SCTP NAT are SCTP's control chunk multiplexing and multi-homing features. Control chunk multiplexing can expose an SCTP NAT to possible Denial of Service attacks. These can be mitigated through the use of chunk and parameter processing limits.

Multiple and changing IP addresses during an SCTP association, mean that SCTP NATs cannot operate in the way conventional UDP/TCP NATs operate. Tracking these multiple global IP addresses can help in avoiding lookup table conflicts, however, it can also result in circumstances that can lead to NAT state inconsistencies. Our analysis shows that tracking global IP addresses is not necessary in most expected practical installations.

We use our FreeBSD SCTP NAT implementation, `alias_sctp` to examine the performance implications of tracking global IP addresses. We find that typical memory usage doubles and that the processing requirements are significant for installations that experience high association arrival rates.

In conclusion we provide practical recommendations for a secure stable SCTP NAT installation.

Categories and Subject Descriptors

C.2.6 [**Internetworking**]: Routers; C.2.6 [**Internetworking**]: Standards; D.4.4 [**Communications Management**]: Network communication

General Terms

Algorithms, Design, Performance, Measurement

Keywords

Stream Control Transmission Protocol (SCTP), Network Address Translation (NAT), FreeBSD, Libalias

1. INTRODUCTION

Network Address Translation (NAT) devices have been deployed extensively in the Internet [1]. Unfortunately, the Stream Control Transmission Protocol (SCTP) cannot traverse currently deployed NATs. An SCTP NAT is critical in order to facilitate the wider adoption of SCTP outside the telecommunications industry.

Recently, Tüxen and Stewart proposed a set of modifications to the SCTP protocol that provide mechanisms to support the address translation of SCTP packets through NATs [2, 3]. In this paper we explore some of the key issues involved in NAT for SCTP. We draw on Tüxen's and Stewart's work along with the experience we have gained in implementing an SCTP NAT module, `alias_sctp` [4] for FreeBSD's `ipfw2` NAT `libalias` kernel module.

The current proposals do not consider some of the security implications for a NAT. We explore a number of Denial of Service (DoS) attacks for which an SCTP NAT may be vulnerable and recommend modifications to the NATs behaviour to mitigate these.

Another key issue relates to the concept of tracking global IP addresses per association within the NAT. We explore the impact of tracking global addresses by:

1. looking at situations where it is difficult for NATs to gain the necessary association state information
2. modelling NAT lookup table conflicts to determine if the reduction in conflict probabilities is operationally significant
3. and by analysing our FreeBSD implementation of an SCTP NAT to look at the relative performance

The paper is structured as follows: Section 2 provides a background of SCTP and how to NAT SCTP packets, according to Tüxen's and Stewart's proposals. Some security aspects with the SCTP NAT design are discussed in section 3. Section 4 examines scenarios where the tracking of global IP addresses can lead to state inconsistencies within NATs while section 5 investigates the probabilities of lookup table conflicts. Section 6 looks at the performance implications of global IP address tracking using our FreeBSD SCTP NAT implementation (`alias_sctp`). We summarise our conclusions and findings in section 7, providing a list of practical recommendations for SCTP NAT design.

2. BACKGROUND

The SCTP protocol, RFC 4960 [5], was originally designed to transport Public Switched Telephone Network (PSTN) signalling over Internet Protocol (IP) networks, but has a far wider applicability. SCTP is a peer protocol to the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP). However it has the ability to combine and

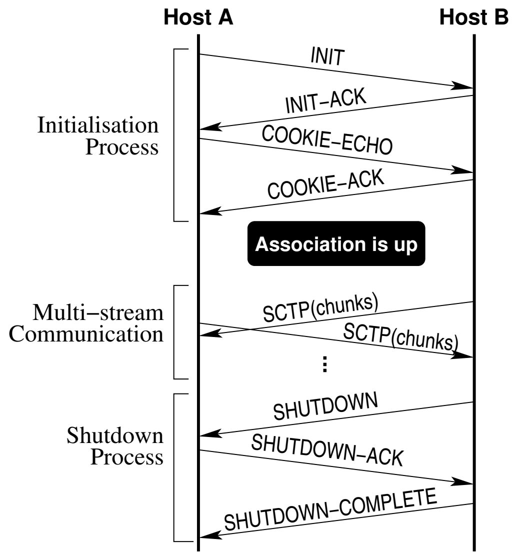


Figure 1: SCTP Initialisation, communication and shutdown

extend the functionality of both TCP and UDP. SCTP's key enhancements include:

- Multi-homing – Ability to use miltiple network interfaces to add network path fault tolerance
- Multiple streams within each connection
- Improved error detection and correction

The SCTP connection mechanism is outlined in the following section (Stewart's and Xie's book [6] provides further description of various scenarios).

2.1 SCTP Streams

Streams are an important part of the SCTP protocol. A single SCTP association can have up to 65 535 concurrent end-to-end sub-flows (or streams) multiplexed together. This is made possible through SCTP's building blocks, *Chunks*. A packet can contain a mixture of control and data chunks, depending on the state of the association. For data transmission, this can prevent head-of-line blocking when packets from a particular stream are lost.

Control chunks can be sent along with data chunks, providing multiple separate control channels for the association. This makes an SCTP NAT's task more difficult, since it needs to search through the different control chunks to find any that affect the current state of this association in the NAT.

2.2 SCTP Association Management

SCTP end-to-end connections are called *associations*. The general association initialisation and disconnection procedure is depicted in Figure 1.

2.2.1 Verification tag (*vtag*)

The *vtag* is a 32 bit random number used by SCTP to help discriminate between different associations. All packets, except the **INIT**, use this identifier during communication. **INIT** packets use $vtag = 0$. *Vtags* are selected during association initialisation. The use of the *vtag* helps prevent blind attacks.

Vtags form part of the flow identification tuple. As such, multiple associations using the same IP address and port number are allowed. We define the local *vtag* as the *vtag* included in all packets sent from the local host, while the global *vtag* is included in all packets sent from the global host.

2.2.2 Initialisation

SCTP uses a four-way handshake to initialise its association. During the **INIT / INIT-ACK** exchange endpoints provide each other with their capabilities, and initialisation for the data transfers. To protect against attacks like the TCP-SYN flood attack [7], SCTP has a *Cookie* exchange before an association is considered to be up. The first cookie is in the **INIT-ACK**, this needs to be echoed with the **COOKIE-ECHO**, and then acknowledged with a **COOKIE-ACK**.

2.2.3 Shutdown

SCTP uses a three way handshake to close its connection. The **SHUTDOWN** chunk informs the end point that the sender wishes to close the association. After receiving a shutdown notification an end point should send any remaining data, and when ready to close the association respond with a **SHUTDOWN-ACK**. To confirm the closure, a **SHUTDOWN-COMPLETE** is sent.

Associations may also be terminated by sending an **ABORT** chunk. This is an unreliable best effort means of immediately shutting down an association. Unlike TCP, there can be no half open connections in the SCTP protocol.

2.2.4 Multi-homing

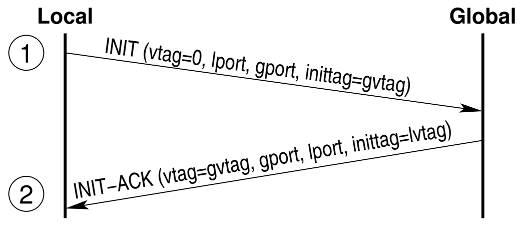
SCTP hosts with more than one IP address (see Figure 3 as an example) include these in the association initialisation to add network path fault tolerance to the association. An end-point uses separate round trip time measures and congestion windows for each address. During normal communication only one address should be used.

Extensions to the base protocol allow support for dynamic multi-homing through the **ASCONF** addition in RFC 5061 [7]. **ASCONF** chunks can add and remove IP addresses connected with an association through use of their **AddIP** and **DelIP** parameters.

Multi-homing presents a number of difficulties for a NAT. No longer is it a simple task of matching address and port numbers (as conventional TCP/UDP NATs do [8, 9]). The IP addresses, may change during an association's life. This paper looks in some detail at these issues in sections 3–6.

2.3 Basic SCTP NAT

In order to translate the addresses of SCTP packets, the NAT needs to uniquely identify all concurrent SCTP associations. Multi-homing presents a significant problem. Single associations now potentially have multiple end-points while individual packets handled by the NAT still have a single source and destination address. Suggestions for how an SCTP NAT should be implemented can be found in [3] and the Internet Draft [2]. This design will hereafter be refered to as Tüxen's NAT.


Local
Global
1
INIT (vtag=0, lport, gport, inittag=gvtag)
2
INIT-ACK (vtag=gvtag, gport, lport, inittag=lvtag)

(a) Initialisation packet exchange

1	idle	*gvtag*	*gport*	*gaddr*		*lport*	*laddr*

(b) NAT association state after **INIT**

2	UP	*gvtag*	*gport*	*gaddr*	*lvtag*	*lport*	*laddr*

(c) NAT association state after **INIT-ACK**

Figure 2: SCTP association initialisation in the Tüxen NAT Architecture

In a TCP/UDP NAT port numbers are modified to resolve flow identification conflicts (two flows with the same <*address,port*> tuple). An SCTP NAT should not do this because:

- Due to multi-homing, an association may traverse multiple NATs via a variety of network paths. Each NAT would need to select the same port number, requiring some form of synchronisation.
- SCTP's checksum requires a full CRC-32C calculation over the entire SCTP packet. Calculating this for every packet traversing the NAT is computationally intensive, and may degrade the performance of the NAT.

Instead, *vtags* are coupled with *port* numbers to provide a unique identity for SCTP associations through the NAT.

Figure 2 shows how Tüxen's NAT would discover the information necessary to identify SCTP associations from the first two session initiation packets.

1. The NAT will store the local and global IP address, source and destination ports, and the global side *vtag* (*gvtag*) as it forwards the SCTP **INIT**.
2. Upon receiving the corresponding **INIT-ACK** and matching the <*gvtag,ports*, *address*> tuple, it will obtain the local side *vtag* (*lvtag*).

Tüxen's NAT recommends tracking global addresses in order to minimise the probability of lookup table conflicts [3]. In addition to matching the <*vtag,port*> tuple, NATs will need to store and match against a list of global IP addresses for each association (see discussion in sections 4 and 5). As this may overload small NATs, Stewart and Tüxen [2] recommend that tracking global IP addresses be optional for them.

The NAT now has enough information to translate the addresses of all subsequent packets belonging to this association. The SCTP cookie exchange will be transparent to the NAT.

Modifications to SCTP are proposed that restrict SCTP end-points from including private IP addresses within the **INIT** and **INIT-ACK** packets [2, 3]. Alternate addresses are added after initialisation using an **ASCONF-AddIP**. NATs parse these **ASCONF** chunks for the **ASCONF-VTag** parameter, which contains both global and local *vtags*. An **ASCONF-ACK** is used for confirmation.

An association will remain UP in the NAT until either:

1. A normal SCTP shutdown process is observed
2. An SCTP **ABORT** chunk is observed
3. An activity timeout occurs – after a predetermined period of no traffic the NAT will consider the association dead and release any resources it was using to track that association

3. SOME SECURITY CONSIDERATIONS

SCTP's features of control chunk multiplexing and multihoming present a number of potential security risks for Tüxen's NAT design. This section examines these and proposes ways to mitigate them. The means through which these risks are mitigated in our implementation [10] are outlined in Appendix A.

3.1 Chunk and parameter processing

SCTP packets may contain any number of chunks, each of which may contain any number of parameters (within the limits of the underlying IP packet size). The NAT requires information found within the chunks and parameters to determine whether it should update its internal state for the association. Due to error (the NAT does not verify the CRC-32C) or design (malicious or otherwise) a packet with a very large number of chunks or a chunk with a very large number of parameters may arrive at the NAT. The NAT will need to parse this information.

3.1.1 Example: UP state processing

If a packet is part of an association that is currently in the UP state, the NAT will need to scan the control chunks for an **ASCONF-AddIP**, **ASCONF-DelIP**, **ABORT**, or **SHUTDOWN-ACK** chunk. The smallest possible chunk size is 4 bytes, so in a typical IPv4 packet (1500 bytes) the NAT could potentially have to parse up to 375 possible chunks.

3.1.2 Example: Initialisation

If an Initialisation chunk is received from a multi-homed host, it may contain any number of IPv4 and IPv6 parameters. If the SCTP NAT is to find and process these parameters, it could potentially scan the entire packet. This problem is only applicable when tracking global IP addresses.

3.1.3 Protecting the SCTP NAT

The previous two examples illustrate how Tüxen's NAT could be exposed to a DoS type attack that consumes its processing resources, preventing it from servicing valid packets.

Possible solutions to limit the vulnerability of the NAT to SCTP packet processing DoS attacks are to:

1. Limit chunk processing – *NAT relevant control chunks should always be as near as practical to first in the SCTP message.*

   INIT, **INIT-ACK**, and **SHUTDOWN-COMPLETE** are specified as single chunk packets. It makes good sense for **SHUTDOWN-ACK** and **ABORT** chunks to be single chunk packets as well, since it is of no practical use for them to contain other chunks. Since **ASCONF** chunks are authenticated, they need to be preceded by an **AUTH** chunk. Therefore, we recommend limiting the the processing of control chunks within an SCTP message to a maximum of two chunks.

2. Limit parameter processing – *NAT relevant parameters should always be within the first N parameters in a chunk.*

   There is a practical limit to the number of addresses a multi-homed host can have. We do not expect this to exceed 25. Given this, we recommend that a NAT should not parse more than $N = 25$ parameters, customisable for a particular installation.

Having chunk and parameter processing limits could result in the NAT missing information important to an association's state. We consider the trade-off reasonable to protect the resource consumption of the NAT. We also recommend that the limits be configurable to cater for any given installation's characteristics.

3.2 Address storage

An attacker could consume the memory resources within Tüxen's NAT by sending malicious **ASCONF** chunks that request IP addresses to be added to an association's state. To do this an attacker needs to establish its own association through the NAT, or correctly guess or sniff the global <*vtag*,*ports*,*address*> tuple of an active association.

To protect against such an attack, an SCTP NAT should limit the number of addresses that can be added to an association (this problem does not exist if global IP addresses are not tracked).

3.3 Packets sent in response to Errors

SCTP uses **ABORT** and **ERROR** chunks to notify the communicating host that there is a problem with communication. Proposed additions to the protocol to better facilitate middle boxes [3] allow the NAT to respond to certain errors. The middle box bit (M-bit) set to indicate that the response came from the middle box and not the end host. If the *vtag* for replying is not known the received packets *vtag* may be reflected, with the T-bit set, to communicate the error back to the sender.

ERROR-M chunks may be sent when the NAT receives a packet for which it does not have the necessary information to NAT. This mechanism helps a multi-homing host that starts sending along a different route through the network to provide a NAT with the information that it needs. On receipt of the **ERROR-M**, the host will reply with an **ASCONF-AddIP**.

Two possible scenarios exist for the NAT with missing state information:

1. **No matching association**

   If a NAT were to respond to any packet from the global side in this way, an attacker could probe the NAT to discover currently active associations. Responding to a globally received packet with an **ERROR-M** implies that the NAT will also accept an **ASCONF-AddIP** from a global address for an association that it knows nothing about. This allows an attacker to create multiple fake associations by sending **ASCONF** chunks requesting the addition of IP addresses.

   In this case we recommend that **ERROR-M** chunks are not sent in response to packets on the global side of the NAT.

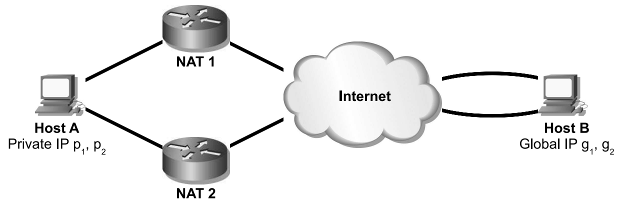


Figure 3: Multi-homed Private and Public Host

2. **<*vtag*,*ports*> match, but no global IP address match**

   This situation will only occur if the NAT is tracking global IP addresses. In this case the NAT recognises the association, but not the global address in the packet. The **ERROR-M** will allow this multi-homed host to add the new IP address via the use of an **ASCONF-AddIP**.

   In this case we recommend that the NAT should respond with an **ERROR-M** since the host has identified all other aspects of an existing association.

4. NAT STATE INCONSISTENCIES WHEN TRACKING GLOBAL IP ADDRESSES

Tüxen's NAT recommends tracking global addresses. However, tracking global IP addresses within an association can lead to situations where Tüxen's NAT may not be aware of all IP addresses involved within that association. Consider the network as portrayed in Figure 3 (the same scenario illustrated in Section 4-2 of [2]). We wish to explore the situation where **Host A** wishes to establish an association with **Host B**.

The exchange of packets, illustrated in Figure 4, will be processed as follows:

1. **A** will begin by sending an **INIT** from IP Address p_1 via **NAT1** to one of the global addresses on **B** – say g_1. Since an **INIT** can only include public (or global) IP addresses, **A** will not include p_2 in the **INIT**.

   As the **INIT** traverses **NAT1**, the NAT will store the destination IP address g_1 in its lookup tables for the new association.

2. **B** will respond with an **INIT-ACK** from g_1 via **NAT1**. Included in the **INIT-ACK** will be **B**'s alternate public IP address g_2. When this packet traverses **NAT1**, it will add the IP address g_2 to the lookup table for this association

3. Once the association is up, **A** may seek to add its alternate path by sending an **ASCONF-AddIP** packet from address p_2 to **B**(g_1) via **NAT2**. **NAT2** will:

   - Extract both *vtags* from the **ASCONF-Vtag** parameter.

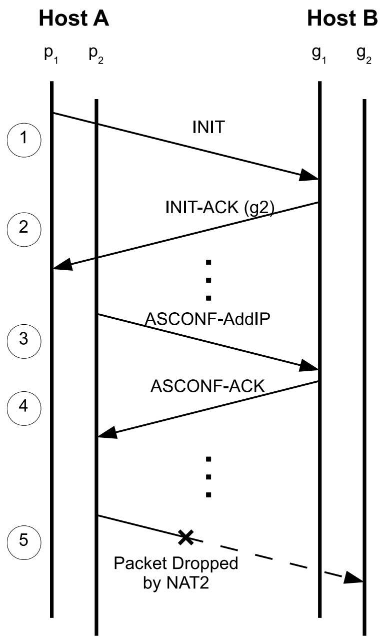


Figure 4: Establishing a multi-homed association through multiple NATs

NAT	state	*gvtag*	*gport*	*gaddr*	*lvtag*	*lport*	*laddr*
1	UP	gv	gp	g1 g2	lv	lp	p1
2	UP	gv	gp	g1	lv	lp	p2

Table 1: Association state in NAT1 and NAT2

- Store the destination IP address of the **ASCONF-AddIP**(g_1)

4. **B** will respond with an **ASCONF-ACK** via **NAT2**. Since there are no IP addresses within the **ASCONF-ACK**, **NAT2** will not learn of **B**'s alternate IP address g_2 as **NAT1** did from the **INIT-ACK**.

5. Suppose that **A**, using address p_2, now wishes to send a packet to **B**, destination address g_2. Since **NAT2** does not know g_2 is valid for this association (see Table 1) it will not forward the packet to **B**.

4.1 Inconsistency in NAT states

The normal response to send an **ERROR-M** packet back to **A** will not result in anything useful occurring as **A** cannot inform **NAT2** of the information it needs to successfully NAT the packet.

In order for **NAT2** to learn the alternate address g_2, **B** has to send an **ASCONF-AddIP** to **A** via **NAT2**. At present, there is no mechanism for **B** to realise that this needs to be done, leaving **NAT2** with incomplete information of the association's state.

NAT2 does not have the necessary state to NAT the packet. Hence, this valid packet, and any future valid packets for this association, will be dropped.

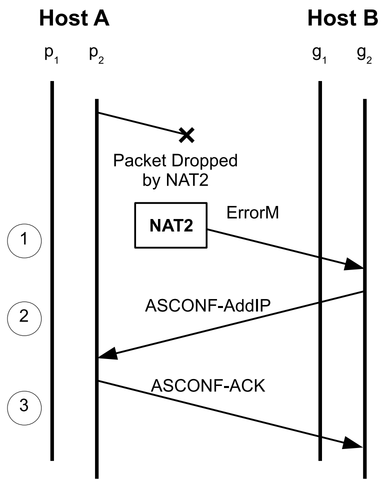


Figure 5: Extension of ERROR-M to handle NAT state inconsistencies

4.2 Possible Solutions

There are a number of possible ways to ensure that both **NAT1** and **NAT2** have the complete association's state.

4.2.1 Mediation entity

Stewart and Tüxen suggest that an inter-NAT communication or an external management scheme may be a solution [2]. In this simple example, **NAT1** and **NAT2** need to know of each others existence and then communicate directly with each other. However in more complex scenarios a mediation entity (or individual NATs) will need to:

1. be known by and know all relevant NATs
2. be trusted by all relevant NATs
3. gather association state information from NATs
4. propagate association state information (possibly on demand) to NATs

This will make practical implementations more difficult.

4.2.2 Extend the use of ERROR-M

Another possible solution to force **B** to generate the required **ASCONF-AddIP** is shown in Figure 5.

1. Upon receipt of the problem packet from **A**, **NAT2** could send an **ERROR-M** to **B**(g_2).

2. **B** should then respond with an **ASCONF-AddIP** so that **NAT2** can add **B**'s alternate IP address, g_2, to its lookup table.

This extends the proposed use of the **ERROR-M** chunk, which replies to the destination address of the packet it was unable to forward. Extending the use of the **ERROR-M** chunk could prove a security issue for the end-host SCTP stack. An attacker could use unsolicited **ERROR-M** chunks to generate a DoS attack, especially since the resultant **ASCONF-AddIP** will be larger than the original **ERROR-M**.

Further, the original packet from **A** to **B** remains dropped and will have to be resent at a later time.

4.2.3 Do not track global addresses

If global IP addresses are **not** tracked this problem *will not occur*. Referring again to the packet sent from Host A via NAT 2 in Figure 4:

5. The packet received at **NAT2** will be correctly forwarded since its <*vtag,ports*> tuple match the lookup table's values.

More complicated case scenarios are also resolved by removing the global address from the tuple.

Not tracking global IP addresses makes it easier to randomly guess the <*vtag,ports*> tuple of an active association. This could allow an attacker to inject packets through the NAT to the internal end host. Since an SCTP host does track IP addresses, it will not accept these packets. Despite this, the authors' believe that the state and security problems that occur when the NAT tracks global IP addresses are a greater problem than the vulnerability of a local host to the aforementioned attack in most circumstances.

5. NAT LOOKUP TABLE CONFLICTS

We have identified in the previous sections some problems that can arise when an SCTP NAT tracks global IP addresses. A lookup table conflict occurs when two associations have the same identifying tuple. Tracking global IP addresses extends the state space of a <*vtag,ports*> tuple by adding a list of global addresses to the tuple <*vtag,ports,address*>.

Section 4 demonstrates that tracking global IP addresses can lead to situations where valid packets are dropped by a NAT. The full association state has not been communicated to all NATs. If global addresses are not tracked this will not occur (see 4.2.3). We examine the extent of the conflict problem to determine if tracking global IP addresses is necessary, even in large NATs.

5.1 Model of lookup table process

Since current use of SCTP is in the telecommunications industry, we use Internet telephony type traffic as a base for our model. We assume that calls traversing the NAT can be modelled as having independent Poisson arrivals with exponentially distributed holding times. Looking at calls in one direction, the lookup table system can be modeled as shown in Figure 6. Telephony calls, in this case SCTP associations, arrive at rate Λ. The lookup table is modelled as m servers, which remain busy for the life of the call $1/\mu$. SCTP sources randomly choose a *vtag* and source *port*. This is modelled by association arrivals being randomly directed to a particular server (lookup table element). If the server is busy (a conflict), the association initialisation is retried with a newly generated *vtag*. This results in an $R\Lambda$ increase to the arrival rate. We approximate this retry rate as a Poisson rate of $R\Lambda$, which can be added to the new association arrival rate to give a total arrival rate of $\Lambda + R\Lambda$.

Since the choice of server is random and independent, each server can be modelled using the Markov process depicted in figure 7 where: $\lambda = \Lambda/m$, $r = R/m$, and $1/\mu$ is the average time an association remains in the table (holding time). The retry rate, $r\lambda$ is the proportion of incoming traffic that finds the system busy ($r = P[1]$).

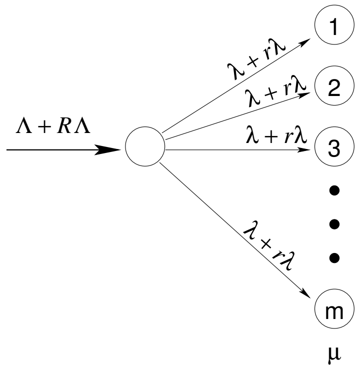


Figure 6: Model of the lookup table process for Λ INIT arrivals per second

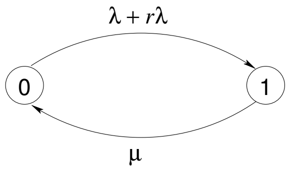


Figure 7: A single lookup table element, receiving the fraction $\lambda = \Lambda/m$ INIT arrivals per second

Solving the balance equations gives (see appendix B):

$$P[\text{conflict}] = \text{P}[1] = \left(1 + \left(\frac{\mu}{2\lambda}\right)^2\right)^{\frac{1}{2}} - \frac{\mu}{2\lambda} \qquad (1)$$

$P[1]$ is the probability that an association wishing to use this element in the lookup table will find it already in use. Since the choice of element is uniformly random, $P[conflict]$ is also the probability that an arriving association finds a lookup table conflict. We would like the NAT to operate with a low $P[conflict]$. If $P[\text{conflict}] \ll 1$, then $\lambda \ll \mu$ and $r\lambda \approx 0$, the probability of a conflict may be approximated as simply:

$$P[\text{conflict}] \approx \frac{\lambda}{\mu} = \frac{\Lambda}{m\mu} \qquad (2)$$

5.2 Two extreme scenarios

In this section we calculate $P[conflict]$ for two extreme scenarios. In the first scenario the lookup table state space is defined only by the *vtag*. In the second, more realistic, scenario the lookup table state space is defined by the *vtag* and *port* combination.

5.2.1 Vtag sized state space

In the theoretical *worst case* all local hosts attempt to initiate associations to the same destination port from the same source port numbers. A conflict occurs when two local hosts coincidentally generate the same *vtag*. Tracking global IP addresses would resolve this conflict if the destination addresses were different. However, when global IP addresses are not tracked, the lookup table state space is limited to $m = 2^{32} - 1$ (32 bit *vtag*, with 0 reserved).

Figure 8 shows a plot of Equation (1) where the average association holding time ($1/\mu$) is 180 s. Table 2 lists the $P[conflict]$ for various NAT capacities.

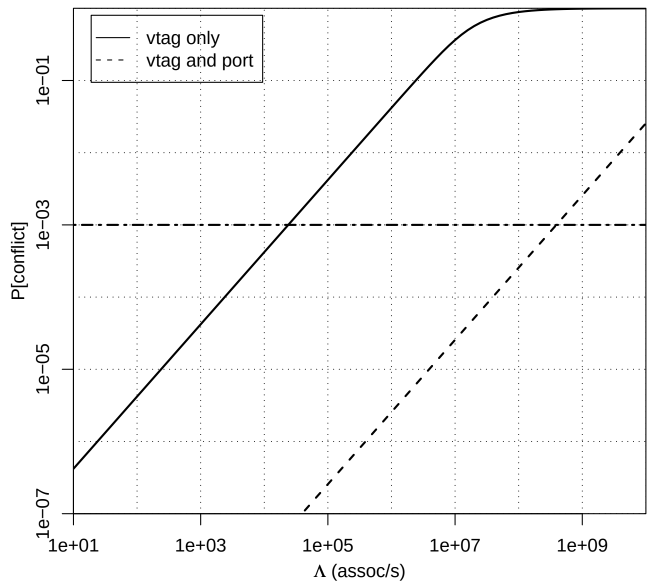


Figure 8: Probabilities of a NAT lookup table conflict versus association arrival rate for the two extreme scenarios (average association holding time of 180 s)

arrivals/s	10	10^3	10^5	10^7
concurrent	1.8×10^3	1.8×10^5	1.8×10^7	1.8×10^9 (see note)
$P[conflict]$ ($vtag$)	4.2×10^{-7}	4.2×10^{-5}	4.2×10^{-3}	0.36
$P[conflict]$ ($vtag/port$)	2.6×10^{-11}	2.6×10^{-9}	2.6×10^{-7}	2.6×10^{-5}

Note: At 10^7 arrivals/s, the vtag only state space has a high $P[conflict]$, resulting in a lower number of concurrent associations, 1.5×10^9.

Table 2: Association arrival rates and number of concurrent associations for the two scenarios (average association holding time of 180 s)

A NAT that keeps $P[conflict] \leq 10^{-3}$ has a working capacity of up to 2.3×10^4 association arrivals per second. On average there can be up to 4.2×10^6 concurrent associations. This will result in an average of 23 conflicts per second, each which will cause an **ABORT-M**[1] reply. The local hosts will then retry with a new *vtag*. The probability of a host having to initiate the same association more than twice is $(P[conflict])^2 = 10^{-6}$.

5.2.2 *Vtag* and *port* sized state space

In a real system, local hosts will randomly generate their 16 bit *port* number, using ephemeral ports (IANA recommended range 49152 through 65535)[11]. An extreme case is where all local hosts are initiating associations to the same service (destination port)[2]. This extends the state space to $m = (2^{32} - 1)2^{14}$. A conflict occurs when two local hosts coincidentally generate the same *vtag* and *source port*.

A NAT that keeps $P[conflict] \leq 10^{-3}$ then has a working capacity of up to 3.8×10^8 association arrivals per second. On average there can be up to 6.8×10^{10} concurrent associations (see Figure 8). Table 2 lists the $P[conflict]$ for various NAT capacities.

5.3 Other traffic types

A NAT on a general purpose network may see a large variety of different traffic types with arrival processes and holding time distributions which are not completely independent and have heavier tails. This will cause wider fluctuations in the number of concurrent associations, increasing the probability of conflict. Even so, we suggest that the probability of a lookup table conflict when not tracking global IP addresses will be very low in all small or large SCTP NAT installations.

6. PERFORMANCE ANALYSIS

We have discussed some of the issues that arise tracking global IP addresses in a NAT. Another aspect to tracking global IP addresses is the processing requirements in doing so. Here we evaluate this using our FreeBSD SCTP NAT implementation (`alias_sctp`) under two scenarios:

Experiment 1 High association arrival/departure rate with a low data rate. This provides an idea of the increased work involved in parsing **INIT** packets to extract and add global IP addresses to the NAT lookup tables.

Experiment 2 A static number of active associations with a high data traffic rate. This provides an idea of the increased work involved in searching the NAT lookup tables when translating the addresses of each SCTP packet.

In each experiment, we run trials under three different scenarios:

1. **Tracking** – Tracking global addresses
2. **No tracking** – Not tracking global address;
3. **Baseline** – no SCTP packet processing (see discussion in section 6.3)

6.1 Alias_sctp

`Alias_sctp` is a FreeBSD kernel based NAT implementation that extends ipfw2's `libalias` kernel module [10]. Version 0.2 of `alias_sctp` supports:

- Configurable NAT management parameters
- Optional tracking of global IP addresses
- Limitations on levels of SCTP chunk/parameter processing

6.2 Experimental testbed

An experimental testbed is configured as shown in Figure 9. Each machine, including the NAT, consists of a Pentium Core 2 Duo running at 2.33 GHz with 1 GB of RAM running the FreeBSD 8.x (kernel revision 180784) operating system.

The existing SCTP stack is not used to generate the test traffic because:

[1][2] recommends that NATs resolve conflicts by sending an **ABORT** chunk with the middle box bit set.

[2]When the possibility of different services is considered, the state space is extended even further.

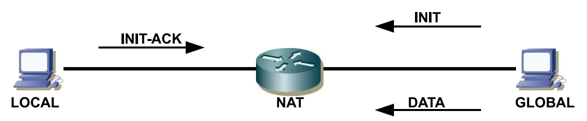


Figure 9: Experimental setup for performance tests

- The stack chooses randomly available port numbers and *vtags*.
- The end host processing load is too high. There is a lot of work in generating, managing, and serving thousands of concurrent SCTP associations and high data rates.

Instead a number of custom test scripts are used:

- SCTP packets are manually generated using raw IP sockets. We select *ports* and *vtags* to ensure there are no lookup table conflicts. While the probability of a conflict is low, the guarantee ensures that the number of associations is deterministic.
- Only the packets necessary for the NAT to establish and terminate associations are used in the initialisation and shutdown phases.
- The CRC-32C is not generated since the NAT does not check this.
- Only **INIT** and **SHUTDOWN-ACK** packets are parsed and responded to with appropriate SCTP response packets returned to the sender (A firewall rule stops incoming SCTP packets, otherwise the local SCTP stack will respond with **ABORT**s).

6.3 Measuring NAT load

Our objective is to measure the relative CPU use of the SCTP NAT when tracking and not tracking global IP addresses. We use the unix `ps` command to obtain the CPU time of the *kernel* process. It gives the cumulative execution time, including interrupt handling, rounded to 0.01 s.

Apart from running the `alias_sctp` module, the *kernel* process includes:

- Bidirectional work of the firewall processing incoming and outgoing packets.
- All other kernel tasks

A baseline experiment is run to measure the work the *kernel* process performs for everything except `alias_sctp`. To do this a modified `alias_sctp` is loaded which writes a hard coded source or destination IP address on each packet it receives. It returns without any parsing or table lookups. Subtracting the **baseline** measurements from the **tracking** and **no tracking** experiments gives an estimate of the CPU load of `alias_sctp`.

6.4 Experiment 1 – High association arrival rate

In this experiment a single client on the global side of the NAT establishes many short associations with a single server on the local side of the NAT. Details are as follows:

kernel cpu time	**Total** Mean (s)	**Total** Variance (s^2)	`alias_sctp` (−base) (s)
tracking	177.46	0.062	33.56
not tracking	165.52	0.067	21.62
baseline	143.90	0.058	—
increase due to tracking	7.2 %	—	55.2 %

Table 3: Experiment 1 (High association arrival rate – 1 500 assocs/s) results

- Each second, 20 blocks of 75 associations are started (association arrival rate = 1 500 assoc/s). This rate ensures that the source, sink and NAT are highly loaded, but not overloaded.
- Collision free environment. *Vtags* are artificially chosen to be unique
- Association lifetime approximately 10 s. This results in about 15 000 concurrent associations.
- Each association, once up, sends at a rate of 1 packet per second (about 15 000 packets/s overall).
- Four IP address parameters are contained within each **INIT**. This simulates a global host with five IP addresses (the packet's source address is not included in the **INIT**).

The experiment is run for 150 minutes under each of the three scenarios. The kernel CPU time is sampled at 15 minute intervals.

6.4.1 Results

Table 3 shows the mean and variance of the 15 minute sampled kernel CPU time for experiment 1, along with an estimate of the time spent in the `alias_sctp` module. These numbers indicate that the NAT is not overloaded. About 3 minutes[3] of each 15 minute sample period is being used by the kernel. The SCTP NAT module uses less than one fifth of the overall processing requirements of the kernel.

Setting up associations is the most CPU intensive task that the SCTP NAT performs. Operations specific to an SCTP NAT are:

1. Parse the packet for the <*vtag*,*ports*,[*address*]> tuple
2. Parse **INIT** chunk for *inittag* and, if tracking, extract all global IP addresses
3. Check for conflicts and add association details to the lookup table

These results show an increase of 55 % in the `alias_sctp` workload when tracking global addresses, with a 7 % increase in the overall kernel workload. In real installations the increased work will depend on the proportion of associations that connect to multi-homed global hosts and the association arrival rate. The proportion of multi-homed associations is potentially quite large, since much through NAT communication is to large multi-homed WWW servers.

[3] 3 minutes of cpu time of one of the 2 cores in the system

This result is important for both small and large NAT devices. Small home routers, although dealing with quite low association arrival rates, have limited processing power. At the other end of the spectrum, corporate or ISP NATs will need to deal with very high association rates making the increase in work load especially significant.

In this experiment there were no <*vtag,ports*> conflicts. When tracking is employed, determining if a conflict exists requires the comparison of all the global addresses in each matching association entry with each of the global addresses contained in the **INIT/INIT-ACK**. Each matching association requires an $O(NM)$ search, where N is the number of addresses in the incoming **INIT/INIT-ACK**, and M is the number of addresses in the otherwise matching association in the table.

6.5 Experiment 2 – High packet rate

The same testbed configuration shown in Figure 9 is used. In this experiment a single client on the global side of the NAT establishes multiple long-term associations with a single server on the local side of the NAT. Details are as follows:

- 2000 concurrent permanent associations
- Collision free environment
- Each association tries to send at a rate of 100 packets/s. The total traffic rate is 2×10^5 packets/s, flooding the 100 Mbps link.
- Each association has 5 global IP addresses.

The experiment is run for 150 minutes under each of the three scenarios. The CPU kernel load is sampled at 15 minute intervals.

6.5.1 Results

When tracking global addresses, a <*vtag,ports*> lookup table match must be searched to ensure that the packet's global address is valid for this association. For this scenario we need to search through 5 address, of which the 5^{th} is a match. Table 4 summarises the results. As per the previous experiment, the NAT is not overloaded. About 4 minutes of each 15 minute sample period is being used by the kernel. The SCTP NAT module is using less than 5 % of the overall kernel processing requirements.

The 31 % increase in `alias_sctp`'s work when tracking global addresses in this scenario is not significant. Even though the NAT is handling over five times the amount of traffic as in Experiment 1, **alias_sctp** is only using about one quarter of the CPU time. Unless there are very high numbers of global IP addresses in each association, the extra workload of tracking global IP addresses when NATing regular packets is minimal.

6.6 Memory usage

A NAT needs to store state for each current association traversing it. In our FreeBSD SCTP NAT implementation each association's state uses 60 bytes of memory, with each stored global IP address adding 12 bytes (4 bytes for the IP address, and 8 bytes for the two pointers in the doubly linked list).

In general, the total memory usage can be described as:

$$M = 8h + 60n + 12\sum_{i=1}^{n} g_i \text{ bytes} \tag{3}$$

kernel cpu time	Total Mean (s)	Total Variance (s^2)	`alias_sctp` (−base) (s)
tracking	231.63	0.068	8.48
not tracking	229.61	0.063	6.46
baseline	223.15	0.130	—
increase due to tracking	0.9 %	—	31.3 %

Table 4: Experiment 2 (High packet rate – 200 000 pkt/s) results

Memory usage	Tracking	Not tracking
Experiment 1	1.8×10^6 bytes	9.0×10^5 bytes
Experiment 2	2.4×10^5 bytes	1.2×10^5 bytes

Table 5: Estimated association state memory consumption

where h is the hash table size (4 byte pointers for local and global tables), n is the number of associations, and g_i is the number of global IP addresses for association i.

Table 5 lists the memory requirements for storing association state, ignoring the hash table, in these experiments[4].

In our implementation, tracking global IP addresses increases the association state memory requirements by 100 % where the SCTP servers are multi-homed with five IP addresses. When multi-homed SCTP servers become common in the internet, NAT-enabled routers can expect a large increase in their memory requirements when they track global IP addresses.

7. CONCLUSIONS

Enabling SCTP to work seamlessly through NAT devices will be an important enabler of SCTP's broader adoption in the public Internet. Experience we have gained through our development of a FreeBSD SCTP in kernel NAT module, `alias_sctp`, has helped us gain insight into many of the issues involved.

SCTP's multi-homing leads to some complex issues regarding global IP address management. We have demonstrated that tracking global IP addresses within the NAT can lead to scenarios of inconsistent state. Some NATs involved in the association may not be aware of all the global IP addresses involved. This can result in a situation where packets are dropped unnecessarily by the NAT. When global IP addresses are not tracked the problem goes away.

Stewart and Tüxen suggest that tracking global IP addresses is necessary in practical implementations to avoid lookup table conflicts. Our analysis of the probability of lookup table conflicts suggest that in all practical implementations (up to many millions of concurrent associations) tracking global addresses is not necessary.

Analysis of the performance of `alias_sctp` 0.2 show a significant increase in a NAT's memory requirements when tracking global IP addresses. The work involved in setting up and pulling down associations in the NAT also increases

[4]Memory consumption for Experiment 1 can fluctuate due to the nature of the experiment.

when tracking global IP addresses, but the overall increase in kernel load is less significant.

Other considerations include the need to secure both the NAT and the internal LAN against different attacks. SCTP's multiplexing of control chunks can make an SCTP NAT particularly vulnerable to certain types of DoS attacks. The current NAT proposal provides most of mechanisms necessary for an SCTP NAT. However, it does not specifically consider a number of these cases. To provide a secure and robust SCTP NAT installation we recommend:

- Limiting the number of chunks (2) and parameters (25) processed by the NAT when analysing packets
- Global IP address tracking should be disabled by default
- If global IP address tracking is enabled, to limit the numbers of IP addresses to be tracked for each association to 25
- Not to respond to an unexpected packet arriving from outside the NAT (unless the NAT is tracking global IP addresses and <*vtags*,*ports*> matches an existing association)

8. ACKNOWLEDGMENTS

The development of the SCTP NAT enhancements to FreeBSD's *libalias* is part of the SONATA [10] project and was made possible in part by a grant from the Cisco University Research Program Fund at Community Foundation Silicon Valley.

The authors would also like to thank Randall Stewart and Michael Tüxen for their vigorous email discussions and help with clarifying some of the issues, CAIA colleague Hai Vu for his helpful discussions on some of the analysis, and the reviewers for their suggestions that have improved the paper.

9. REFERENCES

[1] L. Zhang, "A retrospective view of NAT," *IETF Journal*, vol. 3, no. 2, Oct. 2007. [Online]. Available: http://www.isoc.org/tools/blogs/ietfjournal/?p=157

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

[3] 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.

[4] D. A. Hayes and J. But, "Alias_sctp NAT module," viewed 30 October 2008. [Online]. Available: http://caia.swin.edu.au/urp/sonata/downloads.html

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

[6] R. R. Stewart and Q. Xie, *Stream Control Transmission Protocol (SCTP)*. Addison-Wesley, 2002.

[7] P. Ferguson and D. Senie, "Network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing," IETF, RFC 2827, May 2000.

[8] P. Srisuresh and M. Holdrege, "IP network address translator (NAT) terminology and considerations," IETF, RFC 2663, Aug. 1999.

[9] P. Srisuresh and K. Egevang, "Traditional IP network address translator (traditional NAT)," IETF, RFC 3022, Jan. 2001.

[10] CAIA, "SONATA – SCTP over NAT adaptation," viewed 28 August 2008. [Online]. Available: http://caia.swin.edu.au/urp/sonata

[11] I. A. N. A. (IANA), "Port numbers," viewed 23 October 2008, Oct. 2008. [Online]. Available: http://www.iana.org/assignments/port-numbers

APPENDIX

A. `ALIAS_SCTP`

Our FreeBSD SCTP NAT implementation [10] mitigates the attacks outlined in Section 3 through the use of system parameters configured through FreeBSD's `sysctl` interface:

- `error_on_ootb` – Configures how the NAT responds to packets belonging to an association which it does not have the necessary information to NAT (0 – do not respond, 1 – local response only, 2 – local and global)
- `accept_global_ootb_addip` – If > 0 the NAT will accept global side **ASCONF-AddIP** requests for associations it knows nothing about
- `initialising_chunk_proc_limit` – Limits the number of chunks that will be processed if an association is being initialised
- `chunk_proc_limit` – Limit on the number of chunks that will be processed for an existing association
- `param_proc_limit` – Limit on the number of parameters within a chunk that will be processed
- `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)

B. LOOKUP TABLE PROCESS MODEL

Solving the balance equations for the Markov model depicted in Figure 7:

$$p_0(\lambda + p_1\lambda) = p_1\mu \ , \quad \text{where } p_0 = P[0] \text{ and } p_1 = P[1] \quad (4)$$

and

$$p_1 + p_1 = 1 \quad (5)$$

substituting (5) into (4)

$$p_1^2 + p_1\frac{\lambda}{\mu} - 1 = 0 \quad (6)$$

factorising

$$\left[p_1 + \frac{\mu}{2\lambda} + \left(1 + \left(\frac{\mu}{2\lambda}\right)^2\right)^{\frac{1}{2}}\right] \left[p_1 + \frac{\mu}{2\lambda} - \left(1 + \left(\frac{\mu}{2\lambda}\right)^2\right)^{\frac{1}{2}}\right] = 0 \quad (7)$$

the non-negative root, since $0 \le p_1 \le 1$

$$P[\text{conflict}] = \text{P}[1] = \left(1 + \left(\frac{\mu}{2\lambda}\right)^2\right)^{\frac{1}{2}} - \frac{\mu}{2\lambda} \quad (8)$$