Framework for interfacing with Network Address Translator
<draft-ietf-nat-interface-framework-03.txt>

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

Abstract

NAT is a stateful network device, providing routing transparency for hosts in disparate address realms to communicate with each other. External agents such as Application Level Gateways (ALGs), Realm Specific IP (RSIP) clients and Management applications may need to interact with NAT to influence its operation. The document identifies resources that may be exposed to external agents and illustrates resource control through an Application Programming Interface (API). While the document is focused on describing how to interface with a middlebox supporting NAT function, the description may be applicable to any middlebox that keeps soft-state for sessions. The intent is not to standardize the API. Rather, use API as the basis for the development of a middlebox communication (MIDCOM) protocol by which MIDCOM agents can interface with NAT. NAT resources identified can also form the basis to generate NAT Management Information Base (MIB).
1. Introduction

NAT provides routing transparency for hosts in disparate address realms to communicate with each other. [Ref 1] details the various flavors of NAT that abound. Many internet applications use IP address as host identifier rather than just as a way to locate a host. For this reason, routing transparency by NAT alone is not sufficient to provide end-to-end transparency for applications operating across realms. Application specific ALGs are required in conjunction with NAT to provide end-to-end transparency for some applications.

In addition to ALGs, there can be other kinds of external agents that choose to influence NAT operation. Section 3 identifies these agents. Section 2 below is devoted to describing the resources controlled by NAT. The requirements of external agents, combined with the nature of NAT resources provide the basis to illustrate the resource control mechanism functionally through an API in section 4. Section 5 illustrates how an external agent could use the API to influence NAT operation.

The intent of the document is two-fold. First, an illustration through an API on how to control NAT resource/operation to form a basis for the development of a protocol by which external agents could communicate with NAT. A portion of the communication interface is generic enough that it can be applied to any stateful middlebox. Further, there are extensions that are NAT specific. The API illustration assumes a trusted environment and does not address authentication, authorization or discovery of middlebox location. It is also important to note that the illustration does not assume or require external agents to reside on the same physical device as NAT. In reality, some agents may be co-located with middlebox on the same device and others reside on external devices. Discussion of a communication protocol for use by external agents to interface with middlebox is outside the purview of this document.

Second, NAT controlled resource description specified in the document may be used as the basis to develop NAT Management Information Base (MIB).
2. Terminology

Definitions for majority of terms used in this document may be found in NAT Terminology and Considerations document [Ref 1]. Below are terms defined specifically for this document.

2.1. MiddleBox and Middlebox services

Middlebox is a network intermediate device implementing a middlebox service. Middlebox service is characterized as a networking function requiring application intelligence for its operation. A middlebox service is also referred as Middlebox function throughout this document. Examples of middlebox functions include NAT, firewall, intrusion detection, load balancing, policy based tunneling and IPsec security Gateway enforcement. A middlebox may implement one or more of these functions.

Clearly, NAT is a specific middlebox function and NAT device is a specific Middlebox device.

2.2. MIDCOM Agents

MIDCOM agents are entities performing ALG function, logically external to a middlebox. MIDCOM agents possess a combination of application awareness and knowledge of the middlebox function. A MIDCOM agent may communicate with one or more middleboxes. A MIDCOM agent is also referred as "external agent" throughout the document.

2.3. Middlebox Communication (MIDCOM) protocol

The protocol used by MIDCOM agents to interface with the middlebox and its resources to influence its operation. This is a protocol yet to be devised. The interface API presented in the document, while focused on NAT function, may be used as the basis for the development of such a protocol, extensible to other middlebox functions.

3. Elements of NAT operation

In order to identify an API for use by external agents, it is important to understand the resources and other elements managed by NAT. This would help identify the extent to which an external agent may influence NAT operation. This section describes objects
within NAT, that could be externalized via Management Information Base (MIB).

3.1. Service Descriptor

Of the fields listed below to describe the service, items 3.1.1 through 3.1.2 are independent of the service the device supports. The remaining items are NAT service specific. All flavors of NAT are designed to provide routing transparency to hosts in disparate address realms. A middlebox may have multiple NAT instances or there may be multiple NAT middleboxes associated with a specific realm. The following attributes identify a specific instance of a middlebox supporting NAT function.

3.1.1. Service IDentifier

A service identifier uniquely identifies service instantiation within a middlebox. The external interface address may be one way to uniquely describe this Identifier.

3.1.2. Service type

The service supported on the intermediate node could be one of Basic-NAT, NAPT, Bi-directional-NAT, Twice-NAT, RSA-IP server, RSAP-IP-server or a combination of the above. Service type is an indication of the direction in which new sessions are permitted and the extent of translation done within the IP and transport headers. [Ref 1] has a discussion on NAT flavors and the extent of their translations.

3.1.3. Private and External realm types

Every NAT device will have a minimum of two routing interfaces, one connecting to a private realm and one connecting to external realm. An IPv4 NAT device will have both its realm types set to IPv4.

3.1.4. Address(and Transport-ID) maps

Address map on a NAT device is constituted of one or more of static/dynamic Address maps. Transport-ID mapping, on the other hand, constitutes one or more of static/dynamic transport-ID maps. Transport-ID maps a specific TCP/UDP port (or port range) pertaining to an address in external realm to a specific TCP/UDP port (or port range) in private realm or vice versa. Address (and Transport-ID) maps may be defined for both inbound and outbound directions. Outbound address map refers to mapping a selected set of addresses
from private realm to a selected set of addresses in external realm; whereas inbound address map refers to mapping a set of addresses from the external realm to private realm.

3.1.5. Miscellaneous parameters

NAT may optionally provide TCP, UDP and other types of session Idle-times used to terminate sessions. It may also provide the current range (and, the maximum range) of session IDs and Bind IDs (to be covered in the follow on sub-sections); and the actual count of session IDs and BIND IDs. Specifically, this information will be of relevance to another NAT (backup NAT) that intends to emulate this NAT, in case of failure. Lastly, NAT may choose to supply any other vendor specific parameters such as log options, session direction failure actions and so forth.

3.1.6. Realm Specific IP (RSIP) parameters

A NAT device offering RSIP-Server capability may specify the RSIP tunnel types it supports.

3.2. Address (and Transport-ID) BIND Descriptor

Hereafter, the term BIND will be used in place of BINDing, for ease of use. BIND descriptor is specific to NAT service. These BINDs can be static or dynamic. When external agents do not intervene, dynamic address (and transport-ID) binding is determined by NAT based on the first packet of a session, as described in [Ref 1]. Address binding is between an address in private realm and an address from external realm. Transport-ID BIND is an extension of the same concept to the tuple of Address and transport ID (such as TCP/UDP port no.). The following attributes describe the BIND object(s) maintained by a NAT device.

3.2.1. Bind ID

A number (say, in the range of 1 through 0xFFFFFFFF) assigned to BIND to uniquely identify the BIND on a NAT middlebox.

3.2.2. Direction of Bind

A bind can be uni-directional or bi-directional, same as the orientation of address map based on which this BIND is formed. As before, the direction is with reference to private realm.
3.2.3. Bind type

Indicates whether the BIND is Address-BIND (between a pair of addresses) or Transport-ID-Bind (between a pair of Address, transport ID tuples). Further, this also indicates if the Bind is static or dynamically generated.

3.2.4. Private and External addresses (and Transport-IDs)

These parameters specify the BINDing items in private and external realms.

3.2.5. MIDCOM Agent IDentification

This indicates the last external Agent that registered to control (i.e., set) parameters for this BIND. A value of 0 indicates that native NAT is the responsible agent.

3.2.6. Maximum lease time

The validity of a BIND may be limited by the duration of lease time it is allowed. Unless the lease time is renewed, a BIND will not be valid past the lease time. As a special case, a value of 0 may be assumed to indicate no lease time limit. Typically, this attribute is of relevance only in conjunction with Realm-Specific-IP (RSIP) operation.

3.2.7. Available lease time

This parameter is of relevance only when Maximum lease time is a non-zero value. At any given instance of time, this parameter indicates the real-time left for a BIND to remain valid. Typically, this attribute is of relevance only in conjunction with Realm-Specific-IP (RSIP) operation.

3.2.8. Maximum Idle time

This parameter indicates maximum amount of time a dynamic BIND is allowed to remain valid, with no NAT session hanging off this BIND. Typically, a dynamic Bind is established when NAT notices the first session that needs such a binding. Subsequent to this, multiple NAT sessions can be maintained using the same binding. When the last of these sessions is terminated, the bind is also terminated. In other words, Maximum Idle time is 0, by default, for native NAT. External agents could control this parameter differently. Static Binds and lease time limited BINDs are not effected by this parameter.
3.2.9. Current Idle time

This parameter is of relevance only when Maximum Idle time is set to a non-zero value. At any given instance of time, this parameter indicates the real-time the BIND has been idle with no sessions attached to it.

3.3. Session descriptor

NAT device maintains soft-state for the sessions it tracks. These states may be created dynamically during NAT operation and are referenced for translation of packets pertaining to the session. The entries maintained in a session state are specific to the service type supported (e.g., Basic NAT, NAPT, twice-NAT etc.). For instance, twice-NAT service will require two BINDs, unlike other NAT variations which use just one BIND. The following attributes identify various parameters tracked with a session. Items 2.3.1 through 2.3.9 below are service independent. The remaining items are NAT service specific. Not all services need to track all parameters.

3.3.1. Session IDentifier

A unique number (say, in the range of 1 through 0xFFFFFFFF) assigned by middlebox to a session to uniquely identify this from other sessions on the same device.

3.3.2. Direction of Session

Direction of first packet of the session. As specified earlier, direction is with reference to private realm.

3.3.3. Original Session parameters

These parameters identify the session level parameters as they appear in the first packet of session. These parameters include src and dest IP addresses, IP protocol and transport IDentifier info (such as TCP/UDP port numbers or ICMP Query Identifier).

3.3.4. Controlling Agent IDentification

This indicates the last external Agent who has tried to control parameters for this session. A value of 0 indicates that no external agent is registered to control the session.

3.3.5. Application tag
Sessions are assigned an application tag, so that sessions bearing the same tag can be handled the same way by an external application agent. Application tag may be specified as a tuple of \((<\text{IP}-\text{protocol}>, <\text{session-Port}>)\). For example, an FTP-ALG may register with NAT to control sessions with an FTP application tag.

### 3.3.6. Session Termination heuristic

Session-Idle-time is typically used as a heuristic to determine if the session has ended. There may be other heuristic approaches. A value of zero indicates that no session termination heuristic is used. The controlling agent may take responsibility for session termination in that case. With or without a termination heuristic, TCP sessions may be terminated based on FIN and RST options in TCP header [Ref1].

### 3.3.7. Maximum Idle time

This parameter indicates maximum amount of time this session is allowed to remain valid, even as there is no activity. This parameter is relevant only when session-Idle-time is used as the heuristic to determine session termination. There may be other heuristic approaches. As a special case, a value of 0 implies that the device should run the default timer configured for all native sessions (that have no controlling agent registered).

### 3.3.8. Current Idle Time

This parameter is of relevance only when session termination heuristic is set to session-idle-time. Typically, the device monitors idle time on the sessions it manages periodically and updates this variable. When the idle time exceeds the maximum allowed idle time, the session is terminated.

### 3.3.9. Bundle ID

Applications that deal with a bundle of sessions may cause multiple sessions to be managed by the middlebox. Even though these sessions constitute a single session from application standpoint, the middlebox is not cognizant of the relation. In such cases, it is not uncommon for controlling agents to store session ID of the initial session in all subsequent sessions spawned in that incarnation of the application. Note, Bundle ID need not be the initial session ID, just a
unique ID for all sessions of the bundle.

3.3.10. Bind IDentifier

Identifies the Bind based on which this session is created. The Direction of BIND must be same as that of the session, if the BIND is uni-directional. Typically, if a Bind supporting the session translation does not already exist, a Bind is created prior to creating new session state. However, this Identifier may be set to 0, when BIND creation is unnecessary for the session. For example, there can be no more than one ICMP Query session using am ICMP Query based transport-ID-bind. In such a case, it suffices to do away with BIND and keep all requisite information within the session state itself.

3.3.11. Translated Session parameters

These parameters identify the session level parameters as the first packet of session is translated. These parameters are derived from the BIND ID(s) off which this session hangs.

3.3.12. Second Bind IDentifier

This is of relevance only to Twice-NAT. For all other flavors of NAT, this parameter may be set to zero. In the case of Twice-NAT, the Primary Bind Identifier refers to the binding of source address and the Second Bind Identifier refers to the binding of the destination address (of the first packet).

3.3.13. Packet modifier functions

Typically, NAT modifies IP header and sometimes the transport header. External agents may choose to assume responsibility for payload modification alone, or the entire packet modification. In the case an external agent assumes responsibility for the entire packet modification, NAT will simply redirect the original packet as is to external translation agent. Otherwise, NAT will perform its share of translation (i.e., IP and transport header translation) and direct the translated packet to external agent.

4. External agents interfacing with NAT

Many network applications assume the IP address of their host to be host Identifier and embed the Identifier information in application specific payload. When packets from such an application traverse NAT, the IP address of private host remains uncorrected in the
payload, as the packet is delivered to hosts in external realm. An Application Level Gateway (ALG) is required to re-interpret such a payload as the payload traverses realms.

In addition, there are applications such as H.323 that use out-of-band signaling to dynamically create newer sessions. While a signaling session itself may be directed to a well-known port, sessions created by it need not be that way. Once again, an ALG may be required to process payload in the signaling sessions and notify NAT to recognize the newly created sessions.

There may be other instances where an ALG may be required to provide application level transparency. In all cases, there is a need for the ALGs to interface with NAT. The ALGs may reside on the NAT device or on an external device. Independent of where an ALG resides, NAT interface requirements remain the same.

In a multi-homed NAT configuration, there is a need for a backup NAT to communicate with the primary and keep in sync, so that when the primary goes away, the backup NAT could instantly assume support for the sessions that primary NAT was responsible for. This is yet another case where an external agent (i.e., backup NAT) has a need to interface with NAT.

A NAT device is uniquely qualified to serve as Realm-Specific-IP Server (i.e., RSA-IP-Server or RSAP-IP-Server) for Realm-Specific-IP clients (i.e., RSA-IP clients or RSAP-IP clients). [Ref 1] has a description of RSIP terminology. RSA-IP clients and RSAP-IP clients need to interface with the server node to obtain an external address (or a tuple of address and TCP/UDP port) while communicating with hosts in external realms. In addition, if NAT were to act as tunnel end-point, RSIP clients will need to interface with NAT to setup tunnel state for the lifetime of RSIP-client address assignment. So, once again, there is a need for an API for use by an external agent (i.e., RSIP client) to communicate with NAT, acting as RSIP-server.

Lastly, a management utility would be useful to interface with NAT for configuration and monitor purposes and to enforce NAT policies. For example, reconfigure a NAT device to switch over from NAPT to Basic-NAT configuration or vice versa. Or, add, terminate and monitor ALGs and other external agents on a NAT box. Such a program would also be useful to notify NAT about the status and setup information concerning ALGs, backup NATs and RSIP clients.

Clearly, agents such as RSIP clients and Backup-NATs are likely to reside on a different physical device than the NAT device. Some of the ALG agents may also reside on an external device. The API
presented in the follow-on section will form a basis for the development of a MIDCOM protocol.

The following diagram identifies a selected list of external agents that might interact with NAT via an API or an yet to be devised protocol.

```
+--------------+  +------+  +-------------+  +------------------+
| RSIP Clients |  | ALGs |  | Pri/Sec NAT |  | Management Appl. |
+--------------+  +------+  +-------------+  +------------------+
                 ^              ^           ^
                 |              |           |
                 v              v           v
+----------------------------------------+
|  Middlebox Communication (MIDCOM) API  |
+----------------------------------------+
+----------------------------------------+
|  NAT Service                           |
|  (Resource management, Packet translation|
|  and External agent interface)          |
+----------------------------------------+
|  NAT managed resources                 |
|  (sessions, BINDs, Address maps etc.)  |
+----------------------------------------+
```

Figure 1. External agents interfacing with NAT Middlebox

The following attributes uniquely identify an external agent to a middlebox. Items 4.1 through 4.4 are service independent, while the remaining items are NAT specific.

4.1. Agent IDentifier

A unique number (say, in the range of 1 through 0xFFFFFFFF) assigned to the agent by middlebox so as to distinguish from other agents. Typically, this ID is issued during the agent registration with middlebox.

4.2. Agent type

The Agent type can be one of ALG or RSIP-Client or Backup-NAT or Management-Application or something else. The agent type may be a criterion used by middlebox to accept or deny registration.

4.3. Agent accessibility information
When an agent is resident on a different physical device than middlebox, this parameter specifies means by which the middlebox can access the agent. This can be a combination of Agent’s IP address, IP protocol (e.g., TCP or UDP), well-known port etc. A value of 0 for agent’s IP address indicates that the agent is co-resident with middlebox.

4.4. Periodic Notification interval

This parameter would be required only when the agent calls for periodic notification from the middlebox. This may be specified in units of seconds.

4.5. Agent call-back requirements

An agent will typically require the middlebox to invoke a call-back function supplied by the agent upon the occurrence of specific events. Events for which an agent needs notification depends on agent type.

An ALG will require NAT to call back when a data packet is received on a session with a certain application-tag (say, FTP). Management applications and Backup-NAT might require NAT to periodically invoke a call-back function. Events all agents might require to be notified of (through a call-back function) would be - session termination, service termination and BIND termination.

4.6. Agent call-back functions

Depending upon call-back requirements, the agent will be required to register one or more call-back function entry points with the middlebox. Below are three different call-back function prototypes.

Event notification – void agent_callback_event(service_id, agent_id, event, event_info)

Periodic notification – void agent_callback_periodic(service_id, agent_id, info_type, info_length, information)

Packet notification – void agent_callback_packet(service_id, agent_id, sess_id, pkt_direction, packet)

4.7. RSIP Server tunnel type requirement

An RSIP client may have a requirement for NAT, acting as
RSIP server to support a certain type of tunneling. In such a case, the agent will specify the tunneling requirement through this parameter.

5. External Agent Programming Interface

A resource control mechanism by which external agents could interface with a NAT device is illustrated in this section through an Application Programming Interface (API) in pseudo C language. The API is by no means exhaustive in coverage.

This section is divided into three sub-sections. The first two sub-sections list function calls that may be made available to external agents. These calls are synchronous and require the middlebox to return back a value. Functions in the first sub-section are service independent and are prefixed with "middlebox_", whereas functions in the second subsection are NAT service specific and are prefixed with "nat_". The third sub-section lists functions required to be provided by external agents in order for the middlebox to call-back upon some events.

5.1. Service-independent middlebox API functions

The following functions in a middlebox may be made available to external agents needing to interface with NAT. However, these functions are not NAT specific and may be equally applicable to middleboxes supporting other types of services such as firewall. As such, all functions in this subsection are prefixed with "middlebox_" to highlight the service-neutral nature of these APIs.

The functions listed may be broadly divided into three categories - Service-Query functions, Agent-Registration functions and Session management functions. Session Identifiers are used as the basis for middlebox resource interface. While Session Identifier is service-neutral, some of the resources associated with a session-id can be service specific.

5.1.1. int middlebox_query_IDentity(service_type, **service_info)

Purpose:

This function is used to obtain the services and the individual characteristics of the services supported on a middlebox. Characteristics of a NAT service are described
Input parameters:

- service_type - This parameter is filled to verify if the middlebox supports a certain service type (Refer section 3.1.2). A value of 0 may be used to request the middlebox to report all service types supported.

Output Parameters:

- service_info - Middlebox will fill up a descriptor block with all the characteristics for the matching service_type (refer section 3.1 for NAT characteristics) and return pointer to the descriptor block. The descriptor block would specifically include a service identifier (refer section 3.1.1) that uniquely identifies service instance. This is referred as service_id throughout the document.

Multiple pieces of this information may be returned, if the middlebox supports multiple instances of the same service type. Multiple instances of descriptor blocks may also be returned when service_type is set to 0 and the middlebox supports multiple service types.

Return Value:

- No-Error(0) - Function processing successful. Service_info may be examined for service description.

- SERVICE-TYPE-NOT-SUPPORTED - Middlebox does not support the requested service type.

5.1.2. int middlebox_register_agent (service_id, &agent_info)

Purpose:

This function is used by external agents to register with the middlebox.

Input parameters:

- service_id - The identifier that uniquely identifies a specific
Service instance on a middlebox.

agent_info - The agent is required to provide all the requisite information (with the exception of agent_id) as described in section 4.0. This ID may be used by the caller to influence middlebox operation.

Output Parameters:

agent_info - NAT will set agent_id in agent_info structure when registration is successful.

Return Value:

No-Error(0) - Registration successful.
AGENT-TYPE-NOT-SUPPORTED - Notify the caller that middlebox does not support API requirements of the agent.
TUNNEL-TYPE-NOT-SUPPORTED - Notify caller that middlebox does not support the RSIP tunnel type requested.
INVALID-SERVICE-ID - The specified service_id is not operational or is incorrect.
AGENT-NOT-AUTHORIZED - The specified agent is not authorized to register with the middlebox.

5.1.3. int middlebox_unregister_agent (service_id, agent_id)

Purpose:

This function is used by external agents to unregister with the middlebox.

Input parameters:

service_id - The identifier that uniquely identifies a specific service instance on a middlebox.
agent_id - The agent Identifier that uniquely identifies the agent to middlebox.

Output Parameters:

None.
Return Value:

- No-Error(0) - Successful unregistration.
- AGENT-ID-NOT-SUPPORTED - Agent ID unknown to middlebox.
- INVALID-SERVICE-ID - The specified service_id is not operational or is incorrect.

5.1.4. int middlebox_query_sess_range(service_id, agent_id, sessid_min, sessid_max, &sess_count, &sess_info)

Purpose:

This function is used to request a middlebox to send valid session information for all sessions with an ID in the range of sessid_min through sessid_max. As a special case, this will return session descriptor block for a single session when sessid_min and sessid_max are the same.

Input parameters:

- service_id - The identifier that uniquely identifies a specific service instance on a middlebox.
- agent_id - The agent Identifier that uniquely identifies the agent to middlebox.
- sessid_min, sessid_max - The range of session IDs that the agent is interested in knowing about.

Output Parameters:

- sess_count - Number of sessions being returned through sess_info pointer.
- sess_info - Return one or more sessions with an ID in the given range.

Return Value:

- No-Error(0) - Session termination successful.
- INVALID-SERVICE-ID- The specified service_id is not operational or is incorrect.
- INVALID-AGENT-ID - The specified Agent_ID is not currently
registered with the middlebox.

5.1.5. int middlebox_set_sess(service_id, agent_id, &sess_info)

Purpose:
This function is used to create a new session state on a middlebox or to set parameters of an existing session.

Input parameters:

service_id - The identifier that uniquely identifies a specific service instance on a middlebox.

agent_id - The agent Identifier that uniquely identifies the agent to middlebox.

sess_info - The caller supplies the specifics of a new session parameters or sets a selected number of parameters of an existing session to influence NAT operation. A new session request is made by setting the session-ID within sess_info structure to 0. A non-Zero session-ID would be interpreted by NAT to mean that the agent is attempting to set some session specific parameters.

Output Parameters:

sess_info - If the caller requested for a session creation and NAT was successful in creating a new session, NAT will fill the structure with the assigned session-ID and any other NAT assigned parameter values. If the caller requested to set some session parameters and NAT succeeded in doing so, the sess_info would be filled with the values that NAT holds.

Return Value:

No-Error(0) - Successful session creation or parameter setting.

SESS-MAKE-FAILED - When middlebox was unable to create session state or was unable to set the requested parameter(s).

INVALID-SESS-INFO - When NAT finds that one or all of the parameters specified is not valid.
5.1.6. int middlebox_free_sess(service_id, agent_id, sess_id)

Purpose:

This function is used to terminate a session on middlebox.

Input parameters:

service_id - The identifier that uniquely identifies a specific service instance on a middlebox.

agent_id - The agent Identifier that uniquely identifies the agent to NAT.

sess_id - The ID of the session that needs to be terminated.

Output Parameters:

none.

Return Value:

No-Error(0) - A return value of 0 implies successful session termination.

INVALID-SESS-ID - The specified session ID does not exist.

INVALID-SERVICE-ID- The specified service_id is not operational or is incorrect.

INVALID-AGENT-ID - The specified Agent_id is not currently registered with the middlebox.

5.1.7. int middlebox_free_sess_bundle(service_id, agent_id, bundle_id)

Purpose:

This function is used to terminate a bundle of sessions, identified by the same bundle ID.

Input parameters:
service_id - The identifier that uniquely identifies a specific service instance on a middlebox.

agent_id - The agent Identifier that uniquely identifies the agent to middlebox.

bundle_id - The ID of the session bundle (group of sessions) that needs to be terminated.

Output Parameters:

none.

Return Value:

No-Error(0) - Successful session termination.

INVALID-BUNDLE-ID - The specified bundle ID does not exist.

INVALID-SERVICE-ID - The specified service_id is not operational or is incorrect.

INVALID-AGENT-ID - The specified Agent_id is not currently registered with the middlebox.

5.2. NAT specific API functions

The following functions in a middlebox may be made available to external agents needing to interface with NAT. These functions are necessary to manipulate NAT specific resources such as NAT address Binds, Address Maps and session parameters. NAT specific session parameters include translated session tuples, Bind ID and Packet modification functions. As such, all functions in this subsection are prefixed with "nat_" to highlight NAT specific nature of these APIs.

5.2.1. int nat_query_address_bind (service_id, pvt_address, ext_address, &bind_info)

Purpose:

This function is used by external agents to obtain Address BIND information.

Input parameters:

service_id - This uniquely identifies the NAT instance.
pvt_address, ext_address - The caller might specify both or just
one of either private address or external address and
set the other to zero.

Output Parameters:

bind_info - NAT will fill up the bind_info data structure
with info as described in section 2.2, if NAT were
to find a match for the addresses specified.

Return Value:

No-Error(0)        - Successful in finding a match.
NO-MATCHING_BIND  - Notify the client that there isn’t a BIND
matching the specified addresses.
INVALID-SERVICE-ID - The specified service_id is not operational
or is incorrect.

5.2.2. int nat_query_transport_bind(service_id, pvt_address, pvt_port,
transport_protocol, ext_address, ext_port, &bind_info)

Purpose:

This function is used by external agents to obtain
Transport ID BIND information.

Input parameters:

service_id - The identifier that uniquely identifies a specific
service instance on a middlebox.

pvt_address, pvt_port,
ext_address, ext_port - The caller might specify both or just
one of either (private address and the port no.) or
external address and the port number.

transport_protocol - This must be one of TCP, UDP or ICMP Query

Output Parameters:

bind_info - NAT will fill up the bind_info data structure
with info as described in section 2.2, if NAT were
to find a match for the addresses specified.

Return Value:
No-Error(0) - A return value of 0 implies success in finding a match.

NO-MATCHING_BIND - Notify the client that there isn’t a BIND matching the specified addresses.

INVALID-SERVICE-ID - The specified service_id is not operational or is incorrect.

5.2.3. int nat_set_bind (service_id, agent_id, &bind_info)

Purpose:

This function is used to create a new Address Bind or set parameters of an existing Bind.

Input parameters:

service_id - The identifier that uniquely identifies the NAT instance.

agent_id - The agent Identifier that uniquely identifies the agent to NAT.

bind_info - The caller supplies the specifics of a new BIND or sets a selected number of parameters of an existing BIND to influence NAT operation. The BIND can be an address BIND or transport BIND. A new BIND request is made by setting the BIND ID within bind_info structure to 0. A non-Zero Bind-ID would be interpreted by NAT to mean that the agent is attempting to set some BIND parameters.

Output Parameters:

bind_info - If the caller requested for a BIND creation and NAT was successful in creating a new BIND, NAT will fill the structure with the assigned BIND ID and any other NAT assigned parameter values. If the caller requested to set some BIND parameters and NAT succeeded in doing so, the bind_info would be filled with the values that NAT holds.

Return Value:

No-Error(0) - Successful BIND creation or parameter setting.
BIND-MAKE-FAILED - NAT was unable to create BIND or was unable to set the requested parameter(s).

INVALID-BIND-INFO - One or all of the parameters specified is not valid.

INVALID-SERVICE-ID - The specified service_id is not operational or is incorrect.

INVALID-AGENT-ID - The specified Agent_id is not currently registered with the middlebox.

5.2.4. int nat_free_bind(service_id, agent_id, bind_id)

Purpose:

This function is used to terminate the specified BIND and any sessions that are based on this BIND.

Input parameters:

service_id - The identifier that uniquely identifies the NAT instance.

agent_id  - The agent Identifier that uniquely identifies the agent to NAT.

bind_id    - The ID of the BIND that needs to be terminated.

Output Parameters:

none.

Return Value:

No-Error(0)       - Successful BIND termination.

INVALID-BIND-ID   - The specified bind_id does not exist.

INVALID-SERVICE-ID- The specified service_id is not operational or is incorrect.

INVALID-AGENT-ID  - The specified Agent_id is not currently registered with the middlebox.

5.3. Call-back functions into an external agent
The callback functions listed can be the same independent of the
middlebox function or the agent that interfaces with the middlebox.
Parameters supplied to these functions may however be specific to
the tuple of (service_type, agent_type). The asynchronous callback
functions may be used to redirect packets to MIDCOM agents (or) to
notify a MIDCOM agent of significant events such as BIND creations
or to notify MIDCOM agent of periodic statistical data.

5.3.1. void agent_callback_event(service_id, agent_id, event_type,
    &event_info)

Purpose:

This function is invoked by the middlebox to notify an agent
of an event status.

Input parameters:

    service_id - The identifier that uniquely identifies a specific
                 service instance on a middlebox.
    agent_id   - The agent Identifier that uniquely identifies the
                 agent to the middlebox.
    event_type - The event can be one of session Creation, session
                 termination, BIND creation or BIND termination
                 for a NAT middlebox.
    event_info - This will return the BIND or session description
                 structure that contains the specific instance
                 identifier and other pertinent information.

5.3.2. void agent_callback_periodic(service_id, agent_id, info_type,
                                         info_length, &periodic_info)

Purpose:

This function is used by the middlebox to notify an agent of a
certain piece of information periodically.

Input parameters:

    service_id - The identifier that uniquely identifies a specific
                 service instance on a middlebox.
    agent_id   - The agent Identifier that uniquely identifies the
                 agent to the middlebox.
info_type - Middlebox may have been requested to periodically notify the agent many types of information. Possible values for this parameter would be statistics update, Incremental BIND update Incremental session update, Incremental BIND termination, Incremental session termination etc..

info_length - Number of bytes included in periodic info block.

periodic_info - This point to the actual periodic information being sent to the agent.

5.3.3. void agent_callback_packet(service_id, agent_id, sess_id, pkt_direction, packet)

Purpose:

This function is used by the middlebox to notify an agent of a data packet for processing. The agent is expected to process the packet and forward to the actual destination in the first-in-first-out (FIFO) order. The processing performed by the agent may be limited to just the payload or the entire packet, as set by the agent at session setup time.

Input parameters:

service_id - The identifier that uniquely identifies a specific service instance on a middlebox.

agent_id - The agent Identifier that uniquely identifies the agent to the middlebox.

sess_id - The Identifier of session to which the packet belongs.

pkt_direction - This can be inbound or outbound.

packet - IP packet that needs to be processed by the agent. If NAT was required to perform header translation, this packet is post-NAT-translated version of the packet. In the case the agent selected to perform the entire translation, the original packet is sent as is to the agent, without any NAT transformation.
6. An illustration of the use of NAT Resource Control Mechanism

The following is an illustration of how an ALG for FTP application ([Ref 7]) could use the API specified to interface with NAT to provide application level transparency. This is not meant to be a detailed description of how an FTP-ALG would work. But, rather an illustration of how an ALG could use the interface with NAT to accomplish the desired application transparency. The section is divided into three sub-sections to illustrate (a) ALG registration with NAT, (b) NAT interface with ALG while an FTP session is active, and (c) Notification to ALG when the FTP session terminates.

6.1. FTP-ALG registration with NAT

FTP-ALG will first probe NAT middlebox to understand the type of service provided by NAT and obtain NAT-ID. Once the service type is agreeable, the ALG will register itself as a client with the NAT middlebox with callback functions (as described below) and obtain an agent-ID from the NAT. The tuple of (service_id, agent_id) uniquely identifies the interface between NAT and ALG.

ftp_alg_pkt_notify() will be registered to process FTP session (TCP port 21) traffic. ftp_alg_event_notify() will be registered to process session or NAT termination.

```
FTP-ALG               NAT
-------               ---

1. Obtain NAT descriptor Info

    middlebox_query_IDentity( 
    0, **nat_descriptor) 
    ------------------------>

    OK
    <--------------------------

2. Register with NAT as ALG for FTP (TCP port 21) and obtain
```
agent_ID from the NAT.

middlebox_register_agent(nat_id, &ftp_alg_info)
------------------------>

NAT will assign an agent_ID.

OK
<-----------------------------

6.2. NAT interface with the ALG during FTP session operation

When NAT sees the first packet of an FTP session, it sets up a BIND descriptor and a session descriptor and tags the session descriptor as FTP-Type (i.e., TCP port 21). NAT will then redirect the packet to FTP-ALG by invoking the ALG supplied callback function - ftp_alg_pkt_notify(). The ALG will obtain session descriptor and BIND descriptor info from the NAT.

Subsequent to this, when NAT redirects FTP packets, the ALG would parse the payload for PORT command or response to "PASV" to determine ensuing data sessions and interact with NAT, as necessary, to obtain the requisite translation parameters. The ALG may modify the FTP packet with translation parameters prior to resending to NAT for forwarding.

FTP-ALG
------

NAT
---

1. NAT sees the first packet of an FTP session. NAT will set up a session state and notify the agent as follows.

ftp_alg_pkt_notify(nat_id, agent_ID, sess_ID, packet_direction, pkt)
<-----------------------------

The ALG may optionally make calls to the NAT to find out about the session and BIND characteristics of the FTP. Further, additional calls may
be made to change the control parameters in these blocks.

\[
\text{middlebox\_query\_sess\_range(}
    \text{nat\_id, agent\_id,}
    \text{sess\_id, sess\_id,}
    \text{&sess\_count, **sess\_info)}
\]

-----------------------------
Find the session descriptor block matching sess_id and return Pointer to this. Bind_id is one of the items in the block.

OK
<-----------------------------

n. NAT will forward all subsequent FTP packets to the agent as follows.

\[
\text{ftp\_alg\_pkt\_notify(nat\_id,}
    \text{agent\_ID, sess\_ID,}
    \text{packet\_direction, pkt)}
\]
<-----------------------------

The ALG will parse for PORT command and PASV response in the payload and track any deltas to TCP sequence and acknowledge numbers. The ALG will interact with NAT, as necessary, to obtain BIND parameters for the data session, setup data session state ahead of time and modify the FTP
packet (as necessary) prior to resending to NAT for forwarding.

Request BIND parameters for the new data session such that there is no leased-time set for it.

\[
nat\_set\_bind(nat\_id, agent\_id, \\
&bind\_info)\]

\[\]----------------------------->

....

Setup a new state for the data session such that the Bundle-ID is set to be the session ID of the controlling FTP session.

\[
middlebox\_set\_sess(nat\_id, agent\_id, \\
&sess\_info)\]

\[\]----------------------------->

6.3. Session termination notification

When the FTP control session is ready to be terminated by the NAT, NAT will notify the event to FTP-ALG as follows.

\[
\begin{align*}
\text{FTP-ALG} & \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \ quad
7. Acknowledgement

The author would like to thank Yakov Rekhter for his valuable advice and contribution in the organization of this document.


The security considerations described in [Ref 1] for all variations of NATs are applicable here. There can be security vulnerabilities to a middlebox, if external agents are allowed to interface with a NAT middlebox without proper authentication and authorization. It is possible that a NAT middlebox may be abruptly disrupted due to malicious manipulation of the resource by an agent purporting to offer ALG service.

REFERENCES

[8] "TRANSMISSION CONTROL PROTOCOL (TCP) SPECIFICATION", RFC 793
RFC 792


Author’s Address:

Pyda Srisuresh
jasmine Networks
3061 Zanker Road, Suite B.
San Jose, CA 95134
U.S.A.

Voice: (408) 895-5032
EMail: srisuresh@yahoo.com