NAME

ng_nat - NAT netgraph node type

SYNOPSIS

#include <netgraph/ng_nat.h>

DESCRIPTION

An **ng_nat** node performs network address translation (NAT) of IPv4 packets passing through it. A **nat** node uses libalias(3) engine for packet aliasing.

HOOKS

This node type has two hooks:

out Packets received on this hook are considered outgoing and will be masqueraded to a configured address.

in Packets coming on this hook are considered incoming and will be dealiased.

CONTROL MESSAGES

This node type supports the generic control messages, plus the following:

NGM_NAT_SET_IPADDR (**setaliasaddr**)

Configure aliasing address for a node. After both hooks have been connected and aliasing address was configured, a node is ready for aliasing operation.

NGM_NAT_SET_MODE (**setmode**)

Set node's operation mode using supplied *struct ng_nat_mode*.

```
struct ng_nat_mode {
        uint32_t   flags;
        uint32_t   mask;
};
/* Supported flags: */
#define NG_NAT_LOG                       0x01
#define NG_NAT_DENY_INCOMING                     0x02
#define NG_NAT_SAME_PORTS                0x04
#define NG_NAT_UNREGISTERED_ONLY         0x10
#define NG_NAT_RESET_ON_ADDR_CHANGE              0x20
#define NG_NAT_PROXY_ONLY                0x40
#define NG_NAT_REVERSE                           0x80
```

```
#define NG_NAT_UNREGISTERED_CGN                    0x100
```

The corresponding libalias flags can be found by replacing the *NG_NAT* prefix with *PKT_ALIAS*.

NGM_NAT_SET_TARGET (**settarget**)
Configure target address for a node. When an incoming packet not associated with any pre-existing aliasing link arrives at the host machine, it will be sent to the specified address.

NGM_NAT_REDIRECT_PORT (**redirectport**)
Redirect incoming connections arriving to given port(s) to another host and port(s). The following *struct ng_nat_redirect_port* must be supplied as argument.

```
#define NG_NAT_DESC_LENGTH     64
struct ng_nat_redirect_port {
            struct in_addr         local_addr;
            struct in_addr         alias_addr;
            struct in_addr         remote_addr;
            uint16_t   local_port;
            uint16_t   alias_port;
            uint16_t   remote_port;
            uint8_t                proto;
            char                   description[NG_NAT_DESC_LENGTH];
};
```

Redirection is assigned an unique ID which is returned as response to this message, and information about redirection added to list of static redirects which later can be retrieved by NGM_NAT_LIST_REDIRECTS message.

NGM_NAT_REDIRECT_ADDR (**redirectaddr**)
Redirect traffic for public IP address to a machine on the local network. This function is known as *static NAT*. The following *struct ng_nat_redirect_addr* must be supplied as argument.

```
struct ng_nat_redirect_addr {
            struct in_addr         local_addr;
            struct in_addr         alias_addr;
            char                   description[NG_NAT_DESC_LENGTH];
};
```

Unique ID for this redirection is returned as response to this message.

NGM_NAT_REDIRECT_PROTO (**redirectproto**)

Redirect incoming IP packets of protocol *proto* (see protocols(5)) to a machine on the local network. The following *struct ng_nat_redirect_proto* must be supplied as argument.

```
struct ng_nat_redirect_proto {
        struct in_addr     local_addr;
        struct in_addr     alias_addr;
        struct in_addr     remote_addr;
        uint8_t            proto;
        char               description[NG_NAT_DESC_LENGTH];
};
```

Unique ID for this redirection is returned as response to this message.

NGM_NAT_REDIRECT_DYNAMIC (**redirectdynamic**)

Mark redirection with specified ID as dynamic, i.e., it will serve for exactly one next connection and then will be automatically deleted from internal links table. Only fully specified links can be made dynamic. The redirection with this ID is also immediately deleted from user-visible list of static redirects (available through NGM_NAT_LIST_REDIRECTS message).

NGM_NAT_REDIRECT_DELETE (**redirectdelete**)

Delete redirection with specified ID (currently active connections are not affected).

NGM_NAT_ADD_SERVER (**addserver**)

Add another server to a pool. This is used to transparently offload network load on a single server and distribute the load across a pool of servers, also known as *LSNAT* (RFC 2391). The following *struct ng_nat_add_server* must be supplied as argument.

```
struct ng_nat_add_server {
        uint32_t  id;
        struct in_addr     addr;
        uint16_t  port;
};
```

First, the redirection is set up by NGM_NAT_REDIRECT_PORT or NGM_NAT_REDIRECT_ADDR. Then, ID of that redirection is used in multiple NGM_NAT_ADD_SERVER messages to add necessary number of servers. For redirections created by NGM_NAT_REDIRECT_ADDR, the *port* is ignored and could have any value. Original redirection’s parameters *local_addr* and *local_port* are also ignored after NGM_NAT_ADD_SERVER was used (they are effectively replaced by server pool).

NGM_NAT_LIST_REDIRECTS (**listredirects**)

Return list of configured static redirects as *struct ng_nat_list_redirects*.

```
struct ng_nat_listrdrs_entry {
        uint32_t  id;                  /* Anything except zero */
        struct in_addr    local_addr;
        struct in_addr    alias_addr;
        struct in_addr    remote_addr;
        uint16_t  local_port;
        uint16_t  alias_port;
        uint16_t  remote_port;
        uint16_t  proto;               /* Valid proto or NG_NAT_REDIRPROTO_ADDR */
        uint16_t  lsnat;               /* LSNAT servers count */
        char              description[NG_NAT_DESC_LENGTH];
};
struct ng_nat_list_redirects {
        uint32_t          total_count;
        struct ng_nat_listrdrs_entry redirects[];
};
#define NG_NAT_REDIRPROTO_ADDR      (IPPROTO_MAX + 3)
```

Entries of the *redirects* array returned in the unified format for all redirect types. Ports are meaningful only if protocol is either TCP or UDP and *static NAT* redirection (created by NGM_NAT_REDIRECT_ADDR) is indicated by *proto* set to NG_NAT_REDIRPROTO_ADDR. If *lsnat* servers counter is greater than zero, then *local_addr* and *local_port* are also meaningless.

NGM_NAT_PROXY_RULE (**proxyrule**)

Specify a transparent proxying rule (string must be supplied as argument). See libalias(3) for details.

NGM_NAT_LIBALIAS_INFO (**libaliasinfo**)

Return internal statistics of libalias(3) instance as *struct ng_nat_libalias_info*.

```
struct ng_nat_libalias_info {
        uint32_t  icmpLinkCount;
        uint32_t  udpLinkCount;
        uint32_t  tcpLinkCount;
        uint32_t  sctpLinkCount;
        uint32_t  pptpLinkCount;
        uint32_t  protoLinkCount;
```

```
              uint32_t   fragmentIdLinkCount;
              uint32_t   fragmentPtrLinkCount;
              uint32_t   sockCount;
      };
```

In case of **ng_nat** failed to retrieve a certain counter from its libalias instance, the corresponding field is returned as *UINT32_MAX*.

NGM_NAT_SET_DLT (**setdlt**)

Sets the data link type on the *in* and *out* hooks. Currently, supported types are **DLT_RAW** (raw IP datagrams , no offset applied, the default) and **DLT_EN10MB** (Ethernet). DLT_ definitions can be found in <*net/bpf.h*>. If you want to work on the ipfw(8) level you must use no additional offset by specifying **DLT_RAW**. If, however, you attach **ng_nat** to a network interface directly and **EN10MB** is specified, then the extra offset will be applied to take into account link-level header. In this mode the **ng_nat** would also inspect appropriate type field in the Ethernet header and pass-through any datagrams that are not IP packets.

NGM_NAT_GET_DLT (**getdlt**)

This control message returns the current data link type of the *in* and *out* hooks.

In all redirection messages *local_addr* and *local_port* mean address and port of target machine in the internal network, respectively. If *alias_addr* is zero, then default aliasing address (set by NGM_NAT_SET_IPADDR) is used. Connections can also be restricted to be accepted only from specific external machines by using non-zero *remote_addr* and/or *remote_port*. Each redirection assigned an ID which can be later used for redirection manipulation on individual basis (e.g., removal). This ID guaranteed to be unique until the node shuts down (it will not be reused after deletion), and is returned to user after making each new redirection or can be found in the stored list of all redirections. The *description* passed to and from node unchanged, together with ID providing a way for several entities to concurrently manipulate redirections in automated way.

SHUTDOWN

This node shuts down upon receipt of a NGM_SHUTDOWN control message, or when both hooks are disconnected.

EXAMPLES

In the following example, the packets are injected into a **nat** node using the ng_ipfw(4) node.

```
# Create NAT node
ngctl mkpeer ipfw: nat 60 out
ngctl name ipfw:60 nat
ngctl connect ipfw: nat: 61 in
```

```
      ngctl msg nat: setaliasaddr x.y.35.8

      # Divert traffic into NAT node
      ipfw add 300 netgraph 61 all from any to any in via fxp0
      ipfw add 400 netgraph 60 all from any to any out via fxp0

      # Let packets continue with after being (de)aliased
      sysctl net.inet.ip.fw.one_pass=0
```

The **ng_nat** node can be inserted right after the ng_iface(4) node in the graph. In the following example, we perform masquerading on a serial line with HDLC encapsulation.

```
      /usr/sbin/ngctl -f- <<-SEQ
              mkpeer cp0: cisco rawdata downstream
              name cp0:rawdata hdlc
              mkpeer hdlc: nat inet in
              name hdlc:inet nat
              mkpeer nat: iface out inet
              msg nat: setaliasaddr x.y.8.35
      SEQ
      ifconfig ng0 x.y.8.35 x.y.8.1
```

The **ng_nat** node can also be attached directly to the physical interface via ng_ether(4) node in the graph. In the following example, we perform masquerading on a Ethernet interface connected to a public network.

```
      ifconfig igb0 inet x.y.8.35 netmask 0xfffff000
      route add default x.y.0.1
      /usr/sbin/ngctl -f- <<-SEQ
          mkpeer igb0: nat lower in
          name igb0:lower igb0_NAT
          connect igb0: igb0_NAT: upper out
          msg igb0_NAT: setdlt 1
          msg igb0_NAT: setaliasaddr x.y.8.35
      SEQ
```

SEE ALSO

libalias(3), ng_ipfw(4), natd(8), ngctl(8), ng_ether(8)

HISTORY

The **ng_nat** node type was implemented in FreeBSD 6.0.

AUTHORS

Gleb Smirnoff <*glebius@FreeBSD.org*>