NAME

ng_netflow - Cisco’s NetFlow implementation

SYNOPSIS

#include <sys/types.h>
#include <netinet/in.h>
#include <netgraph/netflow/ng_netflow.h>

DESCRIPTION

The **ng_netflow** node implements Cisco’s NetFlow export protocol on a router running FreeBSD. The **ng_netflow** node listens for incoming traffic and identifies unique flows in it. Flows are distinguished by endpoint IP addresses, TCP/UDP port numbers, ToS and input interface. Expired flows are exported out of the node in NetFlow version 5/9 UDP datagrams. Expiration reason can be one of the following:

- RST or FIN TCP segment.

- Active timeout. Flows cannot live more than the specified period of time. The default is 1800 seconds (30 minutes).

- Inactive timeout. A flow was inactive for the specified period of time. The default is 15 seconds.

Node supports IPv6 accounting (NetFlow v9 only) and is aware of multiple fibs. Different fibs are mapped to different domain_id in NetFlow V9 and different engine_id in NetFlow V5.

HOOKS

This node type supports up to NG_NETFLOW_MAXIFACES (default 65536) hooks named *iface0*, *iface1*, etc., and the same number of hooks named *out0*, *out1*, etc., plus two export hooks: *export* (for NetFlow version 5) and *export9* (for NetFlow version 9). Export can be done simultaneously for all supported export hooks. By default (ingress NetFlow enabled) node does NetFlow accounting of data received on *iface** hooks. If corresponding *out* hook is connected, unmodified data is bypassed to it, otherwise data is freed. If data is received on *out* hook, it is bypassed to corresponding *iface* hook without any processing (egress NetFlow disabled by default). When full export datagram for an export protocol is built it is sent to the *export* or *export9* hook. In normal operation, one (or more) export hook is connected to the *inet/dgram/udp* hook of the ng_ksocket(4) node.

CONTROL MESSAGES

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

NGM_NETFLOW_INFO (**info**)

Returns some node statistics and the current timeout values in a *struct ng_netflow_info*.

NGM_NETFLOW_IFINFO (**ifinfo**)

Returns information about the *ifaceN* hook. The hook number is passed as an argument.

NGM_NETFLOW_SETDLT (**setdlt**)

Sets data link type on the *ifaceN* hook. Currently, supported types are **DLT_RAW** (raw IP datagrams) and **DLT_EN10MB** (Ethernet). DLT_ definitions can be found in <*net/bpf.h*> header. Currently used values are 1 for **DLT_EN10MB** and 12 for **DLT_RAW**. This message type uses *struct ng_netflow_setdlt* as an argument:

```
struct ng_netflow_setdlt {
        uint16_t iface;         /* which iface dlt change */
        uint8_t  dlt;           /* DLT_XXX from bpf.h */
};
```

The requested *ifaceN* hook must already be connected, otherwise message send operation will return an error.

NGM_NETFLOW_SETIFINDEX (**setifindex**)

In some cases, **ng_netflow** may be unable to determine the input interface index of a packet. This can happen if traffic enters the **ng_netflow** node before it comes to the system interface's input queue. An example of such a setup is capturing a traffic *between* synchronous data line and ng_iface(4). In this case, the input index should be associated with a given hook. The interface's index can be determined via if_nametoindex(3) from userland. This message requires *struct ng_netflow_setifindex* as an argument:

```
struct ng_netflow_setifindex {
        uint16_t iface;         /* which iface index change */
        uint16_t index;         /* new index */
};
```

The requested *ifaceN* hook must already be connected, otherwise the message send operation will return an error.

NGM_NETFLOW_SETTIMEOUTS (**settimeouts**)

Sets values in seconds for NetFlow active/inactive timeouts. This message requires *struct ng_netflow_settimeouts* as an argument:

```
struct ng_netflow_settimeouts {
        uint32_t inactive_timeout;      /* flow inactive timeout */
        uint32_t active_timeout;        /* flow active timeout */
        };
```

NGM_NETFLOW_SETCONFIG (**setconfig**)

Sets configuration for the specified interface. This message requires *struct ng_netflow_setconfig* as an argument:

```
        struct ng_netflow_setconfig {
                uint16_t iface;          /* which iface config change */
                uint32_t conf;           /* new config */
        #define NG_NETFLOW_CONF_INGRESS          1
        #define NG_NETFLOW_CONF_EGRESS           2
        #define NG_NETFLOW_CONF_ONCE             4
        #define NG_NETFLOW_CONF_THISONCE     8
        #define NG_NETFLOW_CONF_NOSRCLOOKUP      16
        #define NG_NETFLOW_CONF_NODSTLOOKUP      32
        };
```

Configuration is a bitmask of several options. Option NG_NETFLOW_CONF_INGRESS enabled by default enables ingress NetFlow generation (for data coming from ifaceX hook). Option *NG_NETFLOW_CONF_EGRESS* enables egress NetFlow (for data coming from outX hook). Option *NG_NETFLOW_CONF_ONCE* defines that packet should be accounted only once if it several times passes via netflow node. Option *NG_NETFLOW_CONF_THISONCE* defines that packet should be accounted only once if it several times passes via exactly this netflow node. These two options are important to avoid duplicate accounting when both ingress and egress NetFlow are enabled. Option *NG_NETFLOW_CONF_NOSRCLOOKUP* skips radix lookup on flow source address used to fill in network mask. Option *NG_NETFLOW_CONF_NODSTLOOKUP* skips radix lookup on destination (which fills egress interface id, destination mask and gateway). If one doesn’t need data provided by lookups, he/she can disable them, to reduce load on routers.

NGM_NETFLOW_SETTEMPLATE (**settemplate**)

Sets various timeouts to announce data flow templates (NetFlow v9-specific). This message requires *struct ng_netflow_settemplate* as an argument:

```
        struct ng_netflow_settemplate {
                uint16_t time;           /* max time between announce */
                uint16_t packets;    /* max packets between announce */
        };
```

Value of time field represents time in seconds to re-announce data templates. Value of packets field represents maximum packets count between re-announcing data templates.

NGM_NETFLOW_SETMTU (**setmtu**)

Sets export interface MTU to build packets of specified size (NetFlow v9-specific). This message requires *struct ng_netflow_setmtu* as an argument:

```
struct ng_netflow_setemtu {
        uint16_t mtu;             /* MTU for packet */
};
```

Default is 1500 bytes.

NGM_NETFLOW_SHOW

This control message asks a node to dump the entire contents of the flow cache. It is called from flowctl(8), not directly from ngctl(8).

NGM_NETFLOW_V9INFO (**v9info**)

Returns some NetFlow v9 related values in a

```
struct ng_netflow_v9info {
  uint16_t    templ_packets;  /* v9 template packets */
  uint16_t    templ_time;     /* v9 template time */
  uint16_t    mtu;            /* v9 MTU */
};
```

SHUTDOWN

This node shuts down upon receipt of a NGM_SHUTDOWN control message, or when all hooks have been disconnected.

EXAMPLES

The simplest possible configuration is one Ethernet interface, where flow collecting is enabled.

```
/usr/sbin/ngctl -f- <<-SEQ
        mkpeer fxp0: netflow lower iface0
        name fxp0:lower netflow
        connect fxp0: netflow: upper out0
        mkpeer netflow: ksocket export inet/dgram/udp
        msg netflow:export connect inet/10.0.0.1:4444
SEQ
```

This is a more complicated example of a router with 2 NetFlow-enabled interfaces fxp0 and ng0. Note that the *ng0:* node in this example is connected to ng_tee(4). The latter sends us a copy of IP packets, which we analyze and free. On *fxp0:* we do not use tee, but send packets back to either node.

```
/usr/sbin/ngctl -f- <<-SEQ
        # connect ng0's tee to iface0 hook
        mkpeer ng0:inet netflow right2left iface0
        name ng0:inet.right2left netflow
        # set DLT to raw mode
        msg netflow: setdlt { iface=0 dlt=12 }
        # set interface index (5 in this example)
        msg netflow: setifindex { iface=0 index=5 }

        # Connect fxp0: to iface1 and out1 hook
        connect fxp0: netflow: lower iface1
        connect fxp0: netflow: upper out1

        # Create ksocket node on export hook, and configure it
        # to send exports to proper destination
        mkpeer netflow: ksocket export inet/dgram/udp
        msg netflow:export connect inet/10.0.0.1:4444
SEQ
```

SEE ALSO

setfib(2), netgraph(4), ng_ether(4), ng_iface(4), ng_ksocket(4), ng_tee(4), flowctl(8), ngctl(8)

B. Claise, Ed, *Cisco Systems NetFlow Services Export Version 9*, RFC 3954.

http://www.cisco.com/en/US/docs/ios/solutions_docs/netflow/nfwhite.html

AUTHORS

The **ng_netflow** node type was written by Gleb Smirnoff <*glebius@FreeBSD.org*>, Alexander Motin <*mav@FreeBSD.org*>, Alexander Chernikov <*melifaro@ipfw.ru*>. The initial code was based on **ng_ipacct** written by Roman V. Palagin <*romanp@unshadow.net*>.

BUGS

Cache snapshot obtained via NGM_NETFLOW_SHOW command may lack some percentage of entries under severe load.

The **ng_netflow** node type does not fill in AS numbers. This is due to the lack of necessary information in the kernel routing table. However, this information can be injected into the kernel from a routing daemon such as GNU Zebra. This functionality may become available in future releases.