NAME

ng_source - netgraph node for traffic generation

SYNOPSIS

#include <sys/types.h>
#include <netgraph/ng_source.h>

DESCRIPTION

The **source** node acts as a source of packets according to the parameters set up using control messages and input packets. The **ng_source** node type is used primarily for testing and benchmarking.

HOOKS

The **source** node has two hooks: *input* and *output*. The *output* hook must remain connected, its disconnection will shutdown the node.

OPERATION

The operation of the node is as follows. Packets received on the *input* hook are queued internally. When *output* hook is connected, **ng_source** node assumes that its neighbour node is of ng_ether(4) node type. The neighbour is queried for its interface name. The **ng_source** node then uses queue of the interface for its evil purposes. The **ng_source** node also disables *autosrc* option on neighbour ng_ether(4) node. If interface name cannot be obtained automatically, it should be configured explicitly with the NGM_SOURCE_SETIFACE control message, and *autosrc* should be turned off on ng_ether(4) node manually.

If the node is connected to a netgraph network, which does not terminate in a real ng_ether(4) interface, limit the packet injection rate explicitly with the *NGM_SOURCE_SETPPS* control message.

Upon receipt of a NGM_SOURCE_START control message the node starts sending the previously queued packets out the *output* hook on every clock tick as fast as the connected interface will take them. While active, on every clock tick the node checks the available space in the interface queue and sends that many packets out its *output* hook. Once the number of packets indicated in the start message has been sent, or upon receipt of a NGM_SOURCE_STOP message, the node stops sending data.

CONTROL MESSAGES

This node type supports the generic control messages as well as the following, which must be sent with the NGM_SOURCE_COOKIE attached.

NGM_SOURCE_GET_STATS (**getstats**)
Returns a structure containing the following fields:

outOctets The number of octets/bytes sent out the *output* hook.

outFrames The number of frames/packets sent out the *output* hook.

queueOctets The number of octets queued from the *input* hook.

queueFrames
The number of frames queued from the *input* hook.

startTime The time the last start message was received.

endTime The time the last end message was received or the output packet count was reached.

elapsedTime Either *endTime* - *startTime* or current time - *startTime*.

NGM_SOURCE_CLR_STATS (**clrstats**)
Clears and resets the statistics returned by **getstats** (except *queueOctets* and *queueFrames*).

NGM_SOURCE_GETCLR_STATS (**getclrstats**)
As **getstats** but clears the statistics at the same time.

NGM_SOURCE_START (**start**)
This message requires a single *uint64_t* parameter which is the number of packets to send before stopping. Node starts sending the queued packets out the *output* hook. The *output* hook must be connected and node must have interface configured.

NGM_SOURCE_STOP (**stop**)
Stops the node if it is active.

NGM_SOURCE_CLR_DATA (**clrdata**)
Clears the packets queued from the *input* hook.

NGM_SOURCE_SETIFACE (**setiface**)
This message requires the name of the interface to be configured as an argument.

NGM_SOURCE_SETPPS (**setpps**)
This message requires a single *uint32_t* parameter which puts upper limit on the amount of packets sent per second.

NGM_SOURCE_SET_TIMESTAMP (**settimestamp**)

This message specifies that a timestamp (in the format of a *struct timeval*) should be inserted in the transmitted packets. This message requires a structure containing the following fields:

offset The offset from the beginning of the packet at which the timestamp is to be inserted.

flags Set to 1 to enable the timestamp.

NGM_SOURCE_GET_TIMESTAMP (**gettimestamp**)
Returns the current timestamp settings in the form of the structure described above.

NGM_SOURCE_SET_COUNTER (**setcounter**)
This message specifies that a counter should be embedded in transmitted packets. Up to four counters may be independently configured. This message requires a structure containing the following fields:

offset The offset from the beginning of the packet at which the counter is to be inserted.

flags Set to 1 to enable the counter.

width The byte width of the counter. It may be 1, 2, or 4.

next_val The value for the next insertion of the counter.

min_val The minimum value to be used by the counter.

max_val The maximum value to be used by the counter.

increment The value to be added to the counter after each insertion. It may be negative.

index The counter to be configured, from 0 to 3.

NGM_SOURCE_GET_COUNTER (**getcounter**)
This message requires a single *uint8_t* parameter which specifies the counter to query. Returns the current counter settings in the form of the structure described above.

SHUTDOWN

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

EXAMPLES

Attach the node to an ng_ether(4) node for an interface. If **ng_ether** is not already loaded you will need to do so. For example, these commands load the **ng_ether** module and attach the *output* hook of a new **source** node to *orphans* hook of the bge0: **ng_ether** node.

```
kldload ng_ether
ngctl mkpeer bge0: source orphans output
```

At this point the new node can be referred to as "bge0:orphans". The node can be given its own name like this:

```
ngctl name bge0:orphans src0
```

After which it can be referred to as "src0:".

Once created, packets can be sent to the node as raw binary data. Each packet must be delivered in a separate netgraph message.

The following example uses a short Perl script to convert the hex representation of an ICMP packet to binary and deliver it to the **source** node's *input* hook via nghook(8):

```
perl -pe 's/(..)[ \t\n]*/chr(hex($1))/ge' <<EOF | nghook src0: input
ff ff ff ff ff ff 00 00 00 00 00 00 08 00 45 00
00 54 cb 13 00 00 40 01 b9 87 c0 a8 2b 65 0a 00
00 01 08 00 f8 d0 c9 76 00 00 45 37 01 73 00 01
04 0a 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15
16 17 18 19 1a 1b 1c 1d 1e 1f 20 21 22 23 24 25
26 27 28 29 2a 2b 2c 2d 2e 2f 30 31 32 33 34 35
36 37
EOF
```

To check that the node has queued these packets you can get the node statistics:

```
ngctl msg bge0:orphans getstats
Args:   { queueOctets=64 queueFrames=1 }
```

Send as many packets as required out the *output* hook:

```
ngctl msg bge0:orphans start 16
```

Either wait for them to be sent (periodically fetching stats if desired) or send the stop message:

```
ngctl msg bge0:orphans stop
```

Check the statistics (here we use **getclrstats** to also clear the statistics):

```
ngctl msg bge0:orphans getclrstats
Args:   { outOctets=1024 outFrames=16 queueOctets=64 queueFrames=1
startTime={ tv_sec=1035305880 tv_usec=758036 } endTime={ tv_sec=1035305880
tv_usec=759041 } elapsedTime={ tv_usec=1005 } }
```

The times are from *struct timeval*s, the *tv_sec* field is seconds since the Epoch and can be converted into a date string via TCL's [clock format] or via the date(1) command:

```
date -r 1035305880
Tue Oct 22 12:58:00 EDT 2002
```

SEE ALSO

netgraph(4), ng_echo(4), ng_hole(4), ng_tee(4), ngctl(8), nghook(8)

HISTORY

The **ng_source** node type was implemented in FreeBSD 4.8.

AUTHORS

Dave Chapeskie