NAME

ng_pipe - Traffic manipulating netgraph node type

SYNOPSIS

#include <netgraph/ng_pipe.h>

DESCRIPTION

The **pipe** node type manipulates traffic by emulating bandwidth and delay, as well as random packet losses.

HOOKS

This node type supports the following hooks:

upper Hook leading to upper layer protocols.

lower Hook leading to lower layer protocols.

Traffic flowing from *upper* to *lower* is considered **downstream** traffic. Traffic flowing from *lower* to *upper* is considered **upstream** traffic.

MODE OF OPERATION

Data received on a hook - both in upstream and downstream direction - is put into an inbound queue. If inbound queue is full, discard one frame depending on dropping policy (from the head or from the tail of the queue).

There are three mutually exclusive modes for the input queue:

First In First Out (FIFO)
: A single queue holds packets in chronological order.

Weighted fair queuing (WFQ)
: There are multiple queues for different traffic flows (based on IPv4 IPs). The longest queue is truncated if necessary. This approach assumes that the stalling flow is the flow with the most packets currently on hold.

Deficit Round Robin (DRR)
: This mode is similar to WFQ, but packets are not taken out in strict chronological order. In principle oldest packets come first, but not too many packets from the same flow.

It is possible to configure a duplication probability. As the dice decides, the currently active packet

stays in the queue while a copy of the packet is sent out. Nothing prevents a packet from being duplicated multiple times.

Packets are dropped with an increasing probability depending on the size of the packet, if a *ber* (bit error rate) is configured.

Surviving packets are delayed by the time the packet would need to travel through a link of the configured bandwidth. If this outbound queue is full, the packet is dropped.

CONTROL MESSAGES

This node type supports the generic control messages and the following specific messages.

NGM_PIPE_SET_CFG (**setcfg**)

Set node configuration to the one specified in *struct ng_pipe_cfg*

Note: To set a value to zero, specify -1 instead. This allows omitting configuration values, which should not be modified.

NGM_PIPE_GET_CFG (**getcfg**)

Return current node configuration as *struct ng_pipe_cfg*

```
struct ng_pipe_cfg {
  u_int64_t  bandwidth;     /* bits per second */
  u_int64_t  delay;         /* additional delay, usec */
  u_int32_t  header_offset; /* offset of IP header in bytes */
  u_int32_t  overhead;      /* assumed L2 overhead in bytes */
  struct ng_pipe_hookcfg  downstream;
  struct ng_pipe_hookcfg  upstream;
};

/* Config structure for one hook */
struct ng_pipe_hookcfg {
  u_int64_t  bandwidth;       /* bits per second */
  u_int64_t  ber;             /* avg. interval between bit errors (1 / BER) */
  u_int32_t  qin_size_limit;  /* number of queue items */
  u_int32_t  qout_size_limit; /* number of queue items */
  u_int32_t  duplicate;       /* probability in % */
  u_int32_t  fifo;            /* 0 = off, 1 = on */
  u_int32_t  drr;             /* 0 = off, 1 = 2048 bytes, or x bytes */
  u_int32_t  wfq;              /* 0 = off, 1 = on */
```

```
  u_int32_t  droptail;       /* 0 = off, 1 = on */
  u_int32_t  drophead;       /* 0 = off, 1 = on */
};
```

NGM_PIPE_GET_STATS (**getstats**)
Return node statistics as *struct ng_pipe_stats*

```
/* Statistics structure for one hook */
struct ng_pipe_hookstat {
  u_int64_t  fwd_octets;
  u_int64_t  fwd_frames;
  u_int64_t  in_disc_octets;
  u_int64_t  in_disc_frames;
  u_int64_t  out_disc_octets;
  u_int64_t  out_disc_frames;
};

/* Statistics structure returned by NGM_PIPE_GET_STATS */
struct ng_pipe_stats {
  struct ng_pipe_hookstat  downstream;
  struct ng_pipe_hookstat  upstream;
};
```

NGM_PIPE_CLR_STATS (**clrstats**)
Clear node statistics.

NGM_PIPE_GETCLR_STATS (**getclrstats**)
Atomically return and clear node statistics.

NGM_PIPE_GET_RUN (**getrun**)
Return node statistics as *struct ng_pipe_run*

```
/* Runtime structure for one hook */
struct ng_pipe_hookrun {
  u_int32_t  fifo_queues;
  u_int32_t  qin_octets;
  u_int32_t  qin_frames;
  u_int32_t  qout_octets;
  u_int32_t  qout_frames;
};
```

```
/* Runtime structure returned by NGM_PIPE_GET_RUN */
struct ng_pipe_run {
  struct ng_pipe_hookrun  downstream;
  struct ng_pipe_hookrun  upstream;
};
```

SHUTDOWN

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

EXAMPLES

Limit outgoing data rate over fxp0 Ethernet interface to 20Mbps in fifo mode and incoming to 50kbps in drr mode and 2% duplicate probability.

```
/usr/sbin/ngctl -f- <<-SEQ
  mkpeer fxp0: pipe lower lower
  name fxp0:lower fxp0_pipe
  connect fxp0: fxp0_pipe: upper upper
  msg fxp0_pipe: setcfg { downstream={ bandwidth=20000000 fifo=1 } }
  msg fxp0_pipe: setcfg { upstream={ bandwidth=500000 drr=1 duplicate=2 } }
SEQ
```

SEE ALSO

netgraph(4), ngctl(8)

AUTHORS

Lutz Donnerhacke <*lutz@donnerhacke.de*> (man page)

BUGS

Error handling for memory issues is missing. If kernel memory cannot be allocated immediately, a kernel panic will be triggered. Same happens if an mbuf is fragmented within the transport headers.