NAME

ng_bridge - Ethernet bridging netgraph node type

SYNOPSIS

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

DESCRIPTION

The **bridge** node type performs Ethernet bridging over one or more links. Each link (represented by a connected hook) is used to transmit and receive raw Ethernet frames. As packets are received, the node learns which link each host resides on. Packets unicast to a known host are directed out the appropriate link only, and other links are spared the traffic. This behavior is in contrast to a hub, which always forwards every received packet to every other link.

LOOP DETECTION

The **bridge** node incorporates a simple loop detection algorithm. A loop is when two ports are connected to the same physical medium. Loops are important to avoid because of packet storms, which severely degrade performance. A packet storm results when the same packet is sent and received over and over again. If a host is detected on link A, and is then detected on link B within a certain time period after first being detected on link A, then link B is considered to be a looped back link. The time period is called the minimum stable time.

A looped back link will be temporarily muted, i.e., all traffic received on that link is ignored.

IPFW PROCESSING

Processing of IP packets via the ipfirewall(4) mechanism on a per-link basis is not yet implemented.

HOOKS

This node type supports an unlimited number of hooks. Each connected hook represents a bridged link. The hooks are named *link0*, *link1*, etc. Typically these hooks are connected to the *lower* hooks of one or more ng_ether(4) nodes. To connect the host machine to a bridged network, simply connect the *upper* hook of an ng_ether(4) node to the bridge node.

Instead of naming a hook *linkX* the hook might be also named *uplinkX*. The node does not learn MAC addresses on uplink hooks, which keeps the internal address table small. This way it is desirable to connect the *lower* hook of an ng_ether(4) node to an *uplink* hook of the bridge, and ignore the complexity of the outside world. Frames with unknown MACs are always sent out to *uplink* hooks, so no functionality is lost.

Frames with unknown destination MAC addresses are replicated to any available hook, unless the first

connected hook is an *uplink* hook. In this case the node assumes, that all unknown MAC addresses are located soley on the *uplink* hooks and only those hooks will be used to send out frames with unknown destination MACs. If the first connected hook is an *link* hook, the node will replicate such frames to all types of hooks, even if *uplink* hooks are connected later.

CONTROL MESSAGES

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

NGM_BRIDGE_SET_CONFIG (*setconfig*)

Set the node configuration. This command takes a *struct ng_bridge_config* as an argument:

```
/* Node configuration structure */
struct ng_bridge_config {
  u_char     debugLevel;         /* debug level */
  uint32_t   loopTimeout;        /* link loopback mute time */
  uint32_t   maxStaleness;       /* max host age before nuking */
  uint32_t   minStableAge;       /* min time for a stable host */
};
```

The *debugLevel* field sets the debug level on the node. At level of 2 or greater, detected loops are logged. The default level is 1.

The *loopTimeout* determines how long (in seconds) a looped link is muted. The default is 60 seconds. The *maxStaleness* parameter determines how long a period of inactivity before a host's entry is forgotten. The default is 15 minutes. The *minStableAge* determines how quickly a host must jump from one link to another before we declare a loopback condition. The default is one second.

NGM_BRIDGE_GET_CONFIG (*getconfig*)

Returns the current configuration as a *struct ng_bridge_config*.

NGM_BRIDGE_RESET (*reset*)

Causes the node to forget all hosts and unmute all links. The node configuration is not changed.

NGM_BRIDGE_GET_STATS (*getstats*)

This command takes a four byte link number as an argument and returns a *struct ng_bridge_link_stats* containing statistics for the corresponding *link*, which must be currently connected:

```
/* Statistics structure (one for each link) */
```

```
struct ng_bridge_link_stats {
  uint64_t   recvOctets;     /* total octets rec'd on link */
  uint64_t   recvPackets;    /* total pkts rec'd on link */
  uint64_t   recvMulticasts; /* multicast pkts rec'd on link */
  uint64_t   recvBroadcasts; /* broadcast pkts rec'd on link */
  uint64_t   recvUnknown;    /* pkts rec'd with unknown dest addr */
  uint64_t   recvRunts;      /* pkts rec'd less than 14 bytes */
  uint64_t   recvInvalid;    /* pkts rec'd with bogus source addr */
  uint64_t   xmitOctets;     /* total octets xmit'd on link */
  uint64_t   xmitPackets;    /* total pkts xmit'd on link */
  uint64_t   xmitMulticasts; /* multicast pkts xmit'd on link */
  uint64_t   xmitBroadcasts; /* broadcast pkts xmit'd on link */
  uint64_t   loopDrops;      /* pkts dropped due to loopback */
  uint64_t   loopDetects;    /* number of loop detections */
  uint64_t   memoryFailures; /* times couldn't get mem or mbuf */
};
```

Negative numbers refer to the *uplink* hooks. So querying for -7 will get the statistics for hook *uplink7*.

NGM_BRIDGE_CLR_STATS (*clrstats*)
: This command takes a four byte link number as an argument and clears the statistics for that link.

NGM_BRIDGE_GETCLR_STATS (*getclrstats*)
: Same as NGM_BRIDGE_GET_STATS, but also atomically clears the statistics as well.

NGM_BRIDGE_GET_TABLE (*gettable*)
: Returns the current host mapping table used to direct packets, in a *struct ng_bridge_host_ary*.

NGM_BRIDGE_SET_PERSISTENT (*setpersistent*)
: This command sets the persistent flag on the node, and takes no arguments.

NGM_BRIDGE_MOVE_HOST (*movehost*)
: This command takes a *struct ng_bridge_move_host* as an argument. It assigns the MAC *addr* to the *hook*. If the *hook* is the empty string, the incoming hook of the control message is used as fallback.

  If necessary, the MAC is removed from the currently assigned hook and moved to the new one. If the MAC is moved faster than *minStableAge*, the hook is considered as a loop and will block traffic for *loopTimeout* seconds.

```
struct ng_bridge_move_host {
  u_char  addr[ETHER_ADDR_LEN];  /* ethernet address */
  char    hook[NG_HOOKSIZ];      /* link where addr can be found */
};
```

SHUTDOWN

This node shuts down upon receipt of a NGM_SHUTDOWN control message, or when all hooks have been disconnected. Setting the persistent flag via a NGM_BRIDGE_SET_PERSISTENT control message disables automatic node shutdown when the last hook gets disconnected.

FILES

/usr/share/examples/netgraph/ether.bridge
Example script showing how to set up a bridging network

SEE ALSO

if_bridge(4), netgraph(4), ng_ether(4), ng_hub(4), ng_one2many(4), ngctl(8)

HISTORY

The **ng_bridge** node type was implemented in FreeBSD 4.2.

AUTHORS

Archie Cobbs <*archie@FreeBSD.org*>