NAME

ng_pppoe - RFC 2516 PPPoE protocol netgraph node type

SYNOPSIS

#include <sys/types.h>
#include <net/ethernet.h>
#include <netgraph.h>
#include <netgraph/ng_pppoe.h>

DESCRIPTION

The **pppoe** node type performs the PPPoE protocol. It is used in conjunction with the netgraph(4) extensions to the Ethernet framework to divert and inject Ethernet packets to and from a PPP agent (which is not specified).

The NGM_PPPOE_GET_STATUS control message can be used at any time to query the current status of the PPPoE module. The only statistics presently available are the total packet counts for input and output. This node does not yet support the NGM_TEXT_STATUS control message.

HOOKS

This node type supports the following hooks:

ethernet The hook that should normally be connected to an ng_ether(4) node. Once connected, **ng_pppoe** will send a message down this hook to determine Ethernet address of the underlying node. Obtained address will be stored and then used for outgoing datagrams.

debug Presently no use.

[unspecified] Any other name is assumed to be a session hook that will be connected to a PPP client agent, or a PPP server agent.

CONTROL MESSAGES

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

NGM_PPPOE_GET_STATUS
This command returns status information in a struct ngpppoestat:

```
struct ngpppoestat {
    u_int   packets_in;     /* packets in from Ethernet */
    u_int   packets_out;    /* packets out towards Ethernet */
};
```

NGM_TEXT_STATUS
This generic message returns a human-readable version of the node status. (not yet)

NGM_PPPOE_CONNECT (**pppoe_connect**)
Tell a nominated newly created hook that its session should enter the state machine as a client. It must be newly created and a service name can be given as an argument. It is legal to specify a zero-length service name, this is common on some DSL setups. It is possible to request a connection to a specific access concentrator, and/or set a specific Host-Uniq tag, required by some Internet providers, using the "[AC-Name][Host-Uniq|]Service-Name" syntax. To set a binary Host-Uniq, it must be encoded as a hexadecimal lowercase string and prefixed with "0x", for example "0x6d792d746167" is equivalent to "my-tag". A session request packet will be broadcast on the Ethernet. This command uses the ngpppoe_init_data structure shown below. For example, this init data argument can be used to connect to "my-isp" service with "my-host" uniq tag, accepting only "remote-ac" as access concentrator:

"remote-ac\my-host|my-isp"

NGM_PPPOE_LISTEN (**pppoe_listen**)
Tell a nominated newly created hook that its session should enter the state machine as a server listener. The argument given is the name of the service to listen for. A zero-length service name will match all requests for service. A matching service request packet will be passed unmodified back to the process responsible for starting the service. It can then examine it and pass it on to the session that is started to answer the request. This command uses the ngpppoe_init_data structure shown below.

NGM_PPPOE_OFFER (**pppoe_offer**)
Tell a nominated newly created hook that its session should enter the state machine as a server. The argument given is the name of the service to offer. A zero-length service is legal. The State machine will progress to a state where it will await a request packet to be forwarded to it from the startup server, which in turn probably received it from a LISTEN mode hook (see above). This is so that information that is required for the session that is embedded in the original session request packet, is made available to the state machine that eventually answers the request. When the Session request packet is received, the session negotiation will proceed. This command uses the ngpppoe_init_data structure shown below.

The three commands above use a common data structure:

```
struct ngpppoe_init_data {
  char      hook[NG_HOOKSIZ];      /* hook to monitor on */
  uint16_t  data_len;              /* length of the service name */
```

```
    char    data[0];        /* init data goes here */
};
```

NGM_PPPOE_SUCCESS (**pppoe_success**)
: This command is sent to the node that started this session with one of the above messages, and reports a state change. This message reports successful Session negotiation. It uses the structure shown below, and reports back the hook name corresponding to the successful session.

NGM_PPPOE_FAIL (**pppoe_fail**)
: This command is sent to the node that started this session with one of the above messages, and reports a state change. This message reports failed Session negotiation. It uses the structure shown below, and reports back the hook name corresponding to the failed session. The hook will probably have been removed immediately after sending this message.

NGM_PPPOE_CLOSE (**pppoe_close**)
: This command is sent to the node that started this session with one of the above messages, and reports a state change. This message reports a request to close a session. It uses the structure shown below, and reports back the hook name corresponding to the closed session. The hook will probably have been removed immediately after sending this message. At present this message is not yet used and a NGM_PPPOE_FAIL message will be received at closure instead.

NGM_PPPOE_ACNAME
: This command is sent to the node that started this session with one of the above messages, and reports the Access Concentrator Name.

The four commands above use a common data structure:

```
struct ngpppoe_sts {
    char    hook[NG_HOOKSIZ];
};
```

NGM_PPPOE_GETMODE (**pppoe_getmode**)
: This command returns the current compatibility mode of the node as a string. ASCII form of this message is "pppoe_getmode". The following keywords can be returned:

"standard"
: The node operates according to RFC 2516.

"3Com"
: When **ng_pppoe** is a PPPoE client, it initiates a session encapsulating packets into incorrect

3Com ethertypes. This compatibility option does not affect server mode. In server mode **ng_pppoe** supports both modes simultaneously, depending on the ethertype, the client used when connecting.

"D-Link"
When **ng_pppoe** is a PPPoE server serving only specific Service-Name(s), it will respond to a PADI requests with empty Service-Name tag, returning all available Service-Name(s) on node. This option is necessary for compatibility with D-Link DI-614+ and DI-624+ SOHO routers as clients, when serving only specific Service-Name. This compatibility option does not affect client mode.

NGM_PPPOE_SETMODE (**pppoe_setmode**)
Configure node to the specified mode. The string argument is required. This command understands the same keywords that are returned by the NGM_PPPOE_GETMODE command. ASCII form of this message is "pppoe_setmode". For example, the following command will configure the node to initiate the next session in the proprietary 3Com mode:

```
ngctl msg fxp0:orphans pppoe_setmode '"3Com"'
```

NGM_PPPOE_SETENADDR (**setenaddr**)
Set the node Ethernet address for outgoing datagrams. This message is important when a node has failed to obtain an Ethernet address from its peer on the ethernet hook, or when user wants to override this address with another one. ASCII form of this message is "setenaddr".

NGM_PPPOE_SETMAXP (**setmaxp**)
Set the node PPP-Max-Payload value as described in RFC 4638. This message applies only to a client configuration. ASCII form of this message is "setmaxp".

Data structure returned to client is:

```
struct ngpppoe_maxp {
  char     hook[NG_HOOKSIZ];
  uint16_t data;
};
```

NGM_PPPOE_SEND_HURL (**send_hurl**)
Tell a nominated hook with an active session to send a PADM message with a HURL tag. The argument is the URL to be delivered to the client:

```
ngctl msg fxp0:orphans send_hurl '{ hook="myHook" data="http://example.net/cpe" }'
```

NGM_PPPOE_SEND_MOTM (**send_motm**)
Tell a nominated hook with an active session to send a PADM message with a MOTM tag. The argument is the message to be delivered to the client:

```
ngctl msg fxp0:orphans send_motm '{ hook="myHook" data="Welcome aboard" }'
```

The two commands above use the same ngpppoe_init_data structure described above.

NGM_PPPOE_HURL
This command is sent to the node that started this session when a PADM message with a HURL tag is received, and contains a URL that the host can pass to a web browser for presentation to the user.

NGM_PPPOE_MOTM
This command is sent to the node that started this session when a PADM message with a MOTM tag is received, and contains a Message Of The Minute that the host can display to the user.

The two commands above use a common data structure:

```
struct ngpppoe_padm {
   char    msg[PPPOE_PADM_VALUE_SIZE];
};
```

SHUTDOWN

This node shuts down upon receipt of a NGM_SHUTDOWN control message, when all session have been disconnected or when the ethernet hook is disconnected.

SYSCTL VARIABLES

The node can mark transmitted LCP Ethernet packets (protocol 0xc021) with 3-bit Priority Code Point (PCP) referring to IEEE 802.1p class of service with following sysctl(8) variable.

net.graph.pppoe.lcp_pcp: 0..7 (default: 0)
Set it to non-zero value to be used by parent network interface driver like vlan(4)

EXAMPLES

The following code uses libnetgraph to set up a **ng_pppoe** node and connect it to both a socket node and an Ethernet node. It can handle the case of when a **ng_pppoe** node is already attached to the Ethernet. It then starts a client session.

```
#include <stdio.h>
#include <stdlib.h>
```

```
#include <string.h>
#include <ctype.h>
#include <unistd.h>
#include <sysexits.h>
#include <errno.h>
#include <err.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <sys/select.h>
#include <net/ethernet.h>

#include <netgraph.h>
#include <netgraph/ng_ether.h>
#include <netgraph/ng_pppoe.h>
#include <netgraph/ng_socket.h>
static int setup(char *ethername, char *service, char *sessname,
                                        int *dfd, int *cfd);

int
main()
{
        int  fd1, fd2;
        setup("xl0", NULL, "fred", &fd1, &fd2);
        sleep (30);
}

static int
setup(char *ethername, char *service, char *sessname,
                                int *dfd, int *cfd)
{
        struct ngm_connect ngc;        /* connect */
        struct ngm_mkpeer mkp;        /* mkpeer */
        /******** nodeinfo stuff **********/
        u_char          rbuf[2 * 1024];
        struct ng_mesg *const resp = (struct ng_mesg *) rbuf;
        struct hooklist *const hlist
                                = (struct hooklist *) resp->data;
        struct nodeinfo *const ninfo = &hlist->nodeinfo;
        int             ch, no_hooks = 0;
```

```
struct linkinfo *link;
struct nodeinfo *peer;
/****message to connect PPPoE session*****/
struct {
        struct ngpppoe_init_data idata;
        char        service[100];
}        message;
/********tracking our little graph ********/
char        path[100];
char        source_ID[NG_NODESIZ];
char        pppoe_node_name[100];
int         k;

/*
 * Create the data and control sockets
 */
if (NgMkSockNode(NULL, cfd, dfd) < 0) {
        return (errno);
}
/*
 * find the ether node of the name requested by asking it for
 * it's inquiry information.
 */
if (strlen(ethername) > 16)
        return (EINVAL);
sprintf(path, "%s:", ethername);
if (NgSendMsg(*cfd, path, NGM_GENERIC_COOKIE,
            NGM_LISTHOOKS, NULL, 0) < 0) {
        return (errno);
}
/*
 * the command was accepted so it exists. Await the reply (It's
 * almost certainly already waiting).
 */
if (NgRecvMsg(*cfd, resp, sizeof(rbuf), NULL) < 0) {
        return (errno);
}
/**
 * The following is available about the node:
 * ninfo->name                (string)
```

```
 * ninfo->type                  (string)
 * ninfo->id                    (uint32_t)
 * ninfo->hooks                 (uint32_t) (count of hooks)
 * check it is the correct type. and get it's ID for use
 * with mkpeer later.
 */
if (strncmp(ninfo->type, NG_ETHER_NODE_TYPE,
        strlen(NG_ETHER_NODE_TYPE)) != 0) {
    return (EPROTOTYPE);
}
sprintf(source_ID, "[%08x]:", ninfo->id);

/*
 * look for a hook already attached.
 */
for (k = 0; k < ninfo->hooks; k++) {
    /**
     * The following are available about each hook.
     * link->ourhook    (string)
     * link->peerhook   (string)
     * peer->name               (string)
     * peer->type               (string)
     * peer->id                 (uint32_t)
     * peer->hooks              (uint32_t)
     */
    link = &hlist->link[k];
    peer = &hlist->link[k].nodeinfo;

    /* Ignore debug hooks */
    if (strcmp("debug", link->ourhook) == 0)
        continue;

    /* If the orphans hook is attached, use that */
    if (strcmp(NG_ETHER_HOOK_ORPHAN,
       link->ourhook) == 0) {
        break;
    }
    /* the other option is the 'divert' hook */
    if (strcmp("NG_ETHER_HOOK_DIVERT",
       link->ourhook) == 0) {
```

```
                        break;
                }
        }

        /*
         * See if we found a hook there.
         */
        if (k < ninfo->hooks) {
                if (strcmp(peer->type, NG_PPPOE_NODE_TYPE) == 0) {
                        /*
                         * If it's a type PPPoE, we skip making one
                         * ourself, but we continue, using
                         * the existing one.
                         */
                        sprintf(pppoe_node_name, "[%08x]:", peer->id);
                } else {
                        /*
                         * There is already someone hogging the data,
                         * return an error. Some day we'll try
                         * daisy-chaining..
                         */
                        return (EBUSY);
                }
        } else {

                /*
                 * Try make a node of type PPPoE against node "ID"
                 * On hook NG_ETHER_HOOK_ORPHAN.
                 */
                snprintf(mkp.type, sizeof(mkp.type),
                        "%s", NG_PPPOE_NODE_TYPE);
                snprintf(mkp.ourhook, sizeof(mkp.ourhook),
                        "%s", NG_ETHER_HOOK_ORPHAN);
                snprintf(mkp.peerhook, sizeof(mkp.peerhook),
                        "%s", NG_PPPOE_HOOK_ETHERNET);
                /* Send message */
                if (NgSendMsg(*cfd, source_ID, NGM_GENERIC_COOKIE,
                            NGM_MKPEER, &mkp, sizeof(mkp)) < 0) {
                        return (errno);
                }
```

```
                /*
                 * Work out a name for the new node.
                 */
                sprintf(pppoe_node_name, "%s:%s",
                        source_ID, NG_ETHER_HOOK_ORPHAN);
        }
        /*
         * We now have a PPPoE node attached to the Ethernet
         * card. The Ethernet is addressed as ethername: The PPPoE
         * node is addressed as pppoe_node_name: attach to it.
         * Connect socket node to specified node Use the same hook
         * name on both ends of the link.
         */
        snprintf(ngc.path, sizeof(ngc.path), "%s", pppoe_node_name);
        snprintf(ngc.ourhook, sizeof(ngc.ourhook), "%s", sessname);
        snprintf(ngc.peerhook, sizeof(ngc.peerhook), "%s", sessname);

        if (NgSendMsg(*cfd, ".:", NGM_GENERIC_COOKIE,
                    NGM_CONNECT, &ngc, sizeof(ngc)) < 0) {
                return (errno);
        }

#ifdef  NONSTANDARD
        /*
         * In some cases we are speaking to 3Com hardware, so
         * configure node to non-standard mode.
         */
        if (NgSendMsg(*cfd, ngc.path, NGM_PPPOE_COOKIE,
                        NGM_PPPOE_SETMODE, NG_PPPOE_NONSTANDARD,
                        strlen(NG_PPPOE_NONSTANDARD) + 1) == -1) {
                return (errno);
        }
#endif

        /*
         * Send it a message telling it to start up.
         */
        bzero(&message, sizeof(message));
        snprintf(message.idata.hook, sizeof(message.idata.hook),
                                        "%s", sessname);
```

```
        if (service == NULL) {
                message.idata.data_len = 0;
        } else {
                snprintf(message.idata.data,
                        sizeof(message.idata.data), "%s", service);
                message.idata.data_len = strlen(service);
        }
        /* Tell session/hook to start up as a client */
        if (NgSendMsg(*cfd, ngc.path,
                    NGM_PPPOE_COOKIE, NGM_PPPOE_CONNECT, &message.idata,
                    sizeof(message.idata) + message.idata.data_len) < 0) {
                return (errno);
        }
        return (0);
}
```

SEE ALSO

netgraph(3), netgraph(4), ng_ether(4), ng_ppp(4), ng_socket(4), ngctl(8), ppp(8), vlan(4)

L. Mamakos, K. Lidl, J. Evarts, D. Carrel, D. Simone, and R. Wheeler, *A Method for transmitting PPP over Ethernet (PPPoE)*, RFC 2516.

HISTORY

The **ng_pppoe** node type was implemented in FreeBSD 4.0.

AUTHORS

Julian Elischer <*julian@FreeBSD.org*>