NAME

ng_patch - trivial mbuf data modifying netgraph node type

SYNOPSIS

#include <netgraph/ng_patch.h>

DESCRIPTION

The **patch** node performs data modification of packets passing through it. Modifications are restricted to a subset of C language operations on unsigned integers of 8, 16, 32 or 64 bit size. These are: set to new value (=), addition (+=), subtraction (-=), multiplication (*=), division (/=), negation (= -), bitwise AND (&=), bitwise OR (|=), bitwise eXclusive OR (^=), shift left (<<=), shift right (>>=). A negation operation is the one exception: integer is treated as signed and second operand (the *value*) is not used. If there is more than one modification operation, they are applied to packets sequentially in the order they were specified by the user. The data payload of a packet is viewed as an array of bytes, with a zero offset corresponding to the very first byte of packet headers, and the *length* bytes beginning from *offset* as a single integer in network byte order. An additional offset can be optionally requested at configuration time to account for packet type.

HOOKS

This node type has two hooks:

in Packets received on this hook are modified according to rules specified in the configuration and then forwarded to the *out* hook, if it exists. Otherwise they are reflected back to the *in* hook.

out Packets received on this hook are forwarded to the *in* hook without any changes.

CONTROL MESSAGES

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

NGM_PATCH_SETDLT (**setdlt**)

Sets the data link type on the *in* hook (to help calculate relative offset). Currently, supported types are **DLT_RAW** (raw IP datagrams , no offset applied, the default) and **DLT_EN10MB** (Ethernet). DLT_ definitions can be found in <*net/bpf.h*>. If you want to work on the link layer header you must use no additional offset by specifying **DLT_RAW**. If **EN10MB** is specified, then the optional additional offset will take into account the Ethernet header and a QinQ header if present.

NGM_PATCH_GETDLT (**getdlt**)

This control message returns the data link type of the *in* hook.

NGM_PATCH_SETCONFIG (**setconfig**)

This command sets the sequence of modify operations that will be applied to incoming data on a hook. The following *struct ng_patch_config* must be supplied as an argument:

```
struct ng_patch_op {
          uint32_t  offset;
          uint16_t  length; /* 1,2,4 or 8 bytes */
          uint16_t  mode;
          uint64_t  value;
};
/* Patching modes */
#define NG_PATCH_MODE_SET      1
#define NG_PATCH_MODE_ADD      2
#define NG_PATCH_MODE_SUB      3
#define NG_PATCH_MODE_MUL      4
#define NG_PATCH_MODE_DIV      5
#define NG_PATCH_MODE_NEG      6
#define NG_PATCH_MODE_AND      7
#define NG_PATCH_MODE_OR       8
#define NG_PATCH_MODE_XOR      9
#define NG_PATCH_MODE_SHL      10
#define NG_PATCH_MODE_SHR      11

struct ng_patch_config {
          uint32_t  count;
          uint32_t  csum_flags;
          uint32_t  relative_offset;
          struct ng_patch_op ops[];
};
```

The *csum_flags* can be set to any combination of CSUM_IP, CSUM_TCP, CSUM_SCTP and CSUM_UDP (other values are ignored) for instructing the IP stack to recalculate the corresponding checksum before transmitting packet on output interface. The **ng_patch** node does not do any checksum correction by itself.

NGM_PATCH_GETCONFIG (**getconfig**)
This control message returns the current set of modify operations, in the form of a *struct ng_patch_config*.

NGM_PATCH_GET_STATS (**getstats**)
Returns the node's statistics as a *struct ng_patch_stats*.

NGM_PATCH_CLR_STATS (**clrstats**)
Clears the node's statistics.

NGM_PATCH_GETCLR_STATS (**getclrstats**)
This command is identical to NGM_PATCH_GET_STATS, except that the statistics are also atomically cleared.

SHUTDOWN

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

EXAMPLES

This **ng_patch** node was designed to modify TTL and TOS/DSCP fields in IP packets. As an example, suppose you have two adjacent simplex links to a remote network (e.g. satellite), so that the packets expiring in between will generate unwanted ICMP-replies which have to go forth, not back. Thus you need to raise TTL of every packet entering link by 2 to ensure the TTL will not reach zero there. To achieve this you can set an ipfw(8) rule to use the **netgraph** action to inject packets which are going to the simplex link into the patch node, by using the following ngctl(8) script:

```
/usr/sbin/ngctl -f- <<-SEQ
        mkpeer ipfw: patch 200 in
        name ipfw:200 ttl_add
        msg ttl_add: setconfig { count=1 csum_flags=1 ops=[      \
                { mode=2 value=3 length=1 offset=8 } ] }
SEQ
/sbin/ipfw add 150 netgraph 200 ip from any to simplex.remote.net
```

Here the "ttl_add" node of type **ng_patch** is configured to add (mode NG_PATCH_MODE_ADD) a *value* of 3 to a one-byte TTL field, which is 9th byte of IP packet header.

Another example would be two consecutive modifications of packet TOS field: say, you need to clear the IPTOS_THROUGHPUT bit and set the IPTOS_MINCOST bit. So you do:

```
/usr/sbin/ngctl -f- <<-SEQ
        mkpeer ipfw: patch 300 in
        name ipfw:300 tos_chg
        msg tos_chg: setconfig { count=2 csum_flags=1 ops=[      \
                { mode=7 value=0xf7 length=1 offset=1 }           \
                { mode=8 value=0x02 length=1 offset=1 } ] }
SEQ
```

```
    /sbin/ipfw add 160 netgraph 300 ip from any to any not dst-port 80
```

This first does NG_PATCH_MODE_AND clearing the fourth bit and then NG_PATCH_MODE_OR setting the third bit.

In both examples the *csum_flags* field indicates that IP checksum (but not TCP or UDP checksum) should be recalculated before transmit.

Note: one should ensure that packets are returned to ipfw after processing inside netgraph(4), by setting appropriate sysctl(8) variable:

```
    sysctl net.inet.ip.fw.one_pass=0
```

SEE ALSO

netgraph(4), ng_ipfw(4), ngctl(8)

HISTORY

The **ng_patch** node type was implemented in FreeBSD 8.1.

AUTHORS

Maxim Ignatenko <gelraen.ua@gmail.com>.

Relative offset code by
DMitry Vagin

This manual page was written by
Vadim Goncharov <vadimnuclight@tpu.ru>.

BUGS

The node blindly tries to apply every patching operation to each packet (except those which offset if greater than length of the packet), so be sure that you supply only the right packets to it (e.g. changing bytes in the ARP packets meant to be in IP header could corrupt them and make your machine unreachable from the network).

!!! WARNING !!!

The output path of the IP stack assumes correct fields and lengths in the packets - changing them by to incorrect values can cause unpredictable results including kernel panics.