Netlink Design Document

by	Sean Ng
mentor	Alexander Chernikov
date	14th June 2021

Description

Netlink is a linux kernel interface used for communication between userspace and kernel processes.

This project aims to port Netlink over to FreeBSD. The goal is to implement NETLINK_ROUTE in FreeBSD, and eventually support NETLINK_GENERIC as well. Eventually, the project aims to have a program use netlink. Currently, systems in FreeBSD which use Netlink sockets in their equivalent implementations in linux are currently supported by routing sockets.

Project is part of GSOC 2021. More details can be found in [GSOC 2021]

Overview of Design

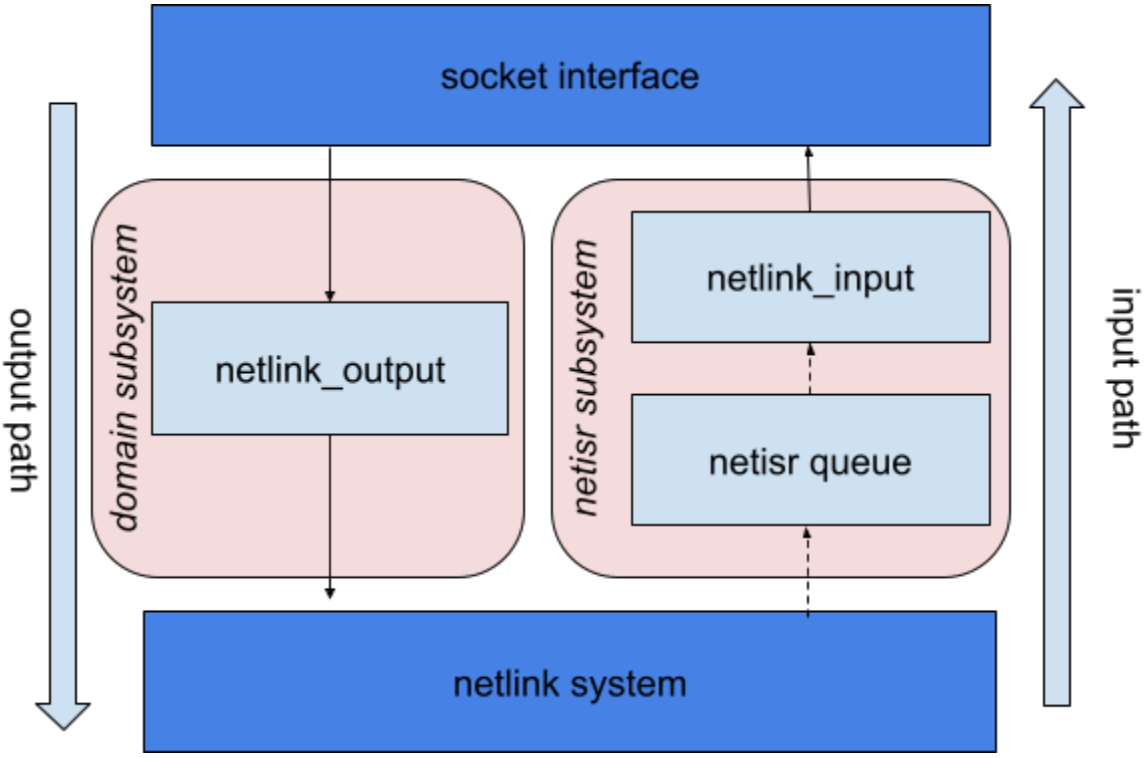


Figure: Overview of the various components needed to *interface* with the netlink system

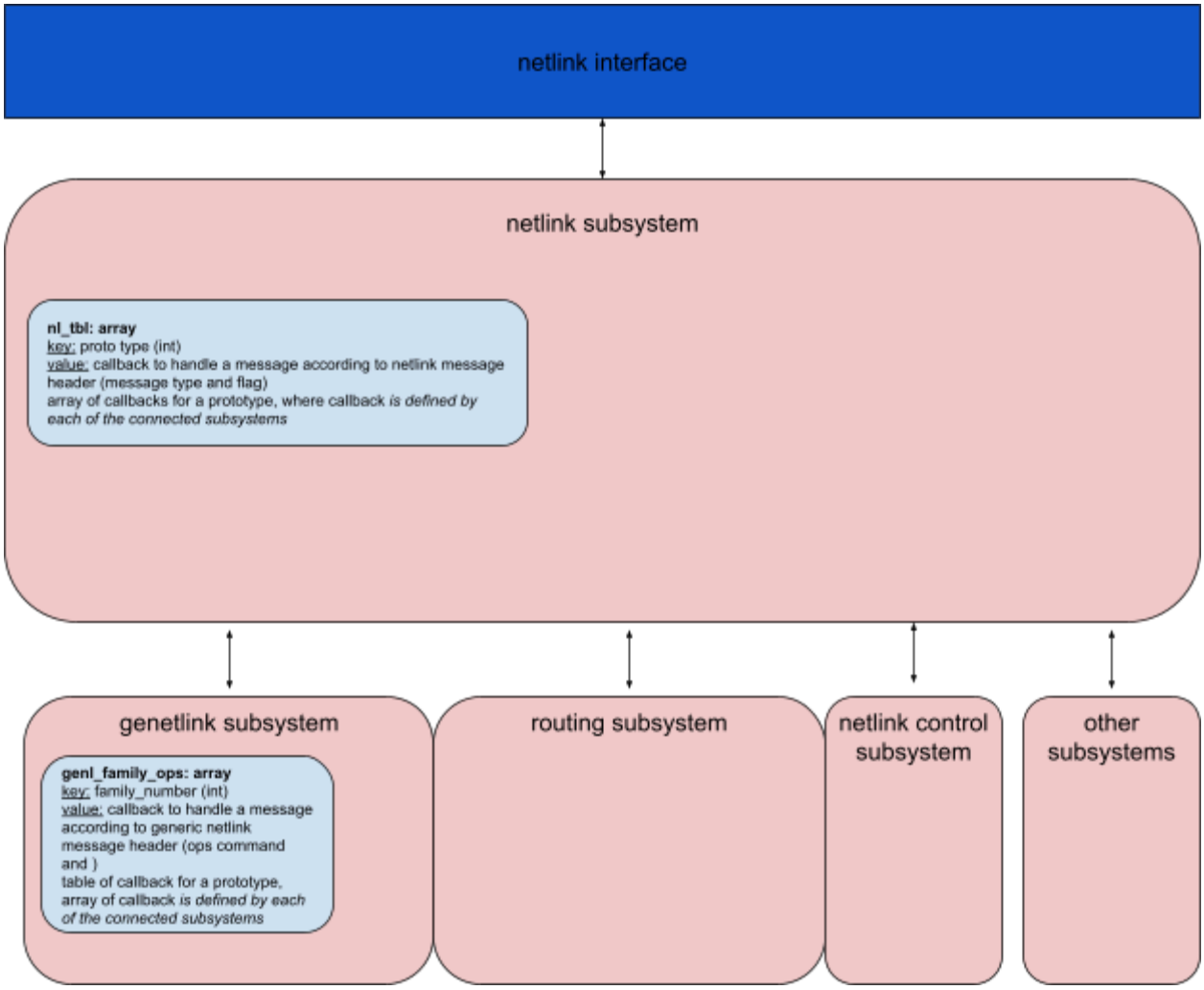


Figure: Overview of the netlink system

Design Details

1.Installing Netlink with netisr (used for deferred dispatch)

Description	*netisr* is the kernel network dispatch service. *netisr_register* is a kernel routine that manages a *netisr_handler.*
Structs	Implement *netisr_handle:* - *nh_handler:* field which is called for all input, refer to **packet input (from kernel) section** - *nh_proto:* new unique protocol number - *nh_policy:* use NETISR_POLICY_SOURCE

Functions	netlink_input: handles messages from kernel. Refer to netlink_input sysctl_netlink_netisr_maxqlen is used when checking queue, needed to register in netisr
man pages	- netisr man page - SYSINIT is useful
TODO	

2.Registering Netlink as a socket (VNET)

Description	*domain* is an abstraction that network protocols are installed under (i.e. inetdomain/localdomain). each domain defines an array of protocol switch structures, one for each socket type.
Structs	Implement *domain* - *dom_family* as PF_NETLINK - dom_protosw as struct defined in next bullet pointj Implement *protosw* - *pr_output:* function output to protocol, which we will define as *netlink_output* - *pr_type: SOCK_RAW* - *pr_usrreqs: pr_usrreqs* - refer to rt_sock for the others Implement *sockproto, sockaddr* Implement *pr_usrreqs* static struct pr_usrreqs netlink_usrreqs = { .pru_abort = l /*This request indicates an abnormal termination of service. The protocol should delete any existing association(s). */ .pru_attach = netlink_attach, /*When a protocol is bound to a socket (with the socket system call) the protocol module is called with this request. It is the responsibility of the protocol module to allocate any resources necessary. The ‘‘attach’’ request will always precede any of the other requests, and should not occur more than once.*/ .pru_bind = netlink_bind, /**/ .pru_connect = netlink_connect, nl_pid is the unicast address of netlink socket. It's always 0 if the destination is in the kernel. For a user-space process, nl_pid is usually the PID of the process owning the destination socket. However, nl_pid identifies a netlink socket, not a process. If a process owns several netlink sockets, then nl_pid

	can be equal to the process ID only for at most one socket. There are two ways to assign nl_pid to a netlink socket. If the application sets nl_pid before calling bind(2), then it is up to the application to make sure that nl_pid is unique. If the application sets it to 0, the kernel takes care of assigning it. The kernel assigns the process ID to the first netlink socket the process opens and assigns a unique nl_pid to every netlink socket that the process subsequently creates. from netlink man page note that the length is updated in the process of the syscall .pru_detach = netlink_detach, .pru_disconnect = netlink_disconnect, .pru_peeraddr = netlink_peeraddr, .pru_send = netlink_send, .pru_shutdown = netlink_shutdown, .pru_sockaddr = netlink_sockaddr, .pru_close = soisdisconnected };
/	sending functions will be discussed in the later sections
man pages	- domain man page - VNET_DOMAIN_SET/DOMAIN_SET is useful
TODO	- What is exact difference between vnet vs non-vnet, or specifically, why we call VNET_DOMAIN_SET instead of DOMAIN_SET - protosw.ctloutput: I don't have a good idea on when this is called so I will leave it out for now - Read https://flylib.com/books/en/2.849.1.145/1/

3.Attaching the netlink socket

Description	As with most socket APIs, upon attaching a socket, the raw socket for that *address family* needs to be instantiated and the fields filled up.
Structs	Need to specify and implement a data structure to keep track of the protocols registered for the netlink address family
Functions	netlink_attach: retrieves raw socket pointer, and initializes the fields in the raw socket
man pages	https://flylib.com/books/en/2.849.1.150/1/
TODO	need more documentation for dealing w raw sockets

4.Sending a message

Description	User creates a message using mbuf, and sends it through netlink. When creating a packet for the socket, used both by protosw and netlinkisr
Structs	Need to specify and implement a data structure to keep track of the protocols and callbacks for each netlink family
Functions	*netlink_output* called with mbuf and socket 1. netlink receives packet, saves the protocol and the portid to the mbuf, 2. arrange packet content into a linear buffer 3. calling a callback on the linear buffer Note that both header and body are padded to the next 4 byte boundary Note that ACK messages need to be send for NLM_F_ACK messages Note that each message should have NLM_F_REQUEST
man pages	https://datatracker.ietf.org/doc/html/rfc3549#section-2.3.2
TODO	In Luigi's implementation, he: 1. copies out into a linear buffer, is it necessary for me to do this? 2. I don't understand his receive packet implementation. in netlink receive packet, each mbuf in the mbuf chain is assumed to have a packet header, and it appears that he calls the callback for EACH of these packets, which means that I should treat these packets as separate requests? Then why the while loop?

5.Netisr packet input

Description	When processing an incoming packet from the netisr queue (nh_handler field in netisr_handler), function needs to implement callback that identifies the right port
Structs	Implement *sockaddr* within the implementation of the *nh_handler* itself - From kernel, the pid of the sockaddress is 0. Implement *sockproto* within the implementation of the *nh_handler* itself
Functions	*raw_input_netlink:* handler defined in netisr_handler. Will create nl_src and nl_proto, using both to pass into *raw_input_ext* for the dispatcher to route the request properly *raw_input_ext* will map the sockaddr with the right port.
man pages	sys/net/raw_cb.h sys/net/*raw_usrreq.c* mbuf manpage (mtod is useful)
TODO	Why do we bother with sockproto and not read the values directly?

	In Luigi, implementation he doesn't even use sockproto in *raw_input_netlink_cb*?

6.Netlink connect

Description	implementation for a connect syscall
Structs	Modify fields in the *socket* struct - nl_src_portid corresponds to so_fibnum - nl_dst_portid corresponds to so_user_cookie
Functions	netlink_connect: sets the portid on the socket, call soisconnected, checks size of nla
man pages	sys/sys/socketvar.h
TODO	Figure out how linux netlink handles the portnumbers

7.Initialize netlink generic functionality

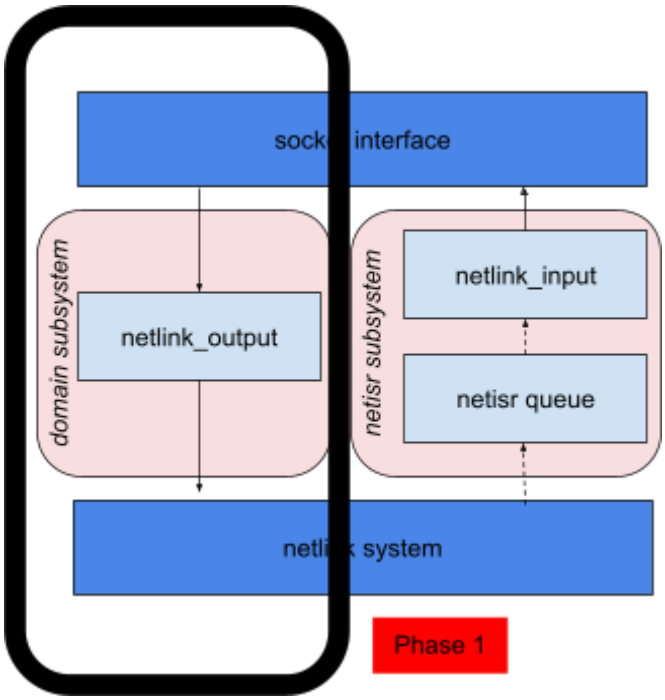
Description	Register new proto in the netlink family, initialize structs for generic. Will be similar to linux's implementation for uniformity.
Structs	Define and implement a struct *genl_ops* that specifies registered generic message handlers: - u8 generic message type - doit and dumpit calllbacks - nla policy - more defined in *Theory and Implementation* book
Functions	genetlinkload that initializes all the functions
man pages	- LIST_INIT and related macros are useful - SYS_INIT is useful - https://elixir.bootlin.com/linux/latest/source/include/net/genetlink.h#L48 - Theory and Implementation book page 27
TODO	figure out exact difference between the doit callback and the dumpit callback

Design Decisions

1. Decision: Linux uses an netlink_sock struct to store details relevant to a connection such as dst_port_id, but this pattern does not appear to be a common pattern in FreeBSD sockets, hence I will not use this pattern. Currently, these values are only needed once and hence I will parse the header to retrieve these values when needed, which is the same as Luigi's implementation.
2. Decision: Data structure for generic netlink callbacks. Luigi uses a single list that he iterates on to find the right generic family structure for a generic generic family id/name. Instead, I decided to use an array of families, similar to that of the structure that stores netlink callbacks. Linux uses a radix tree (linux implementation) (this is an idr)

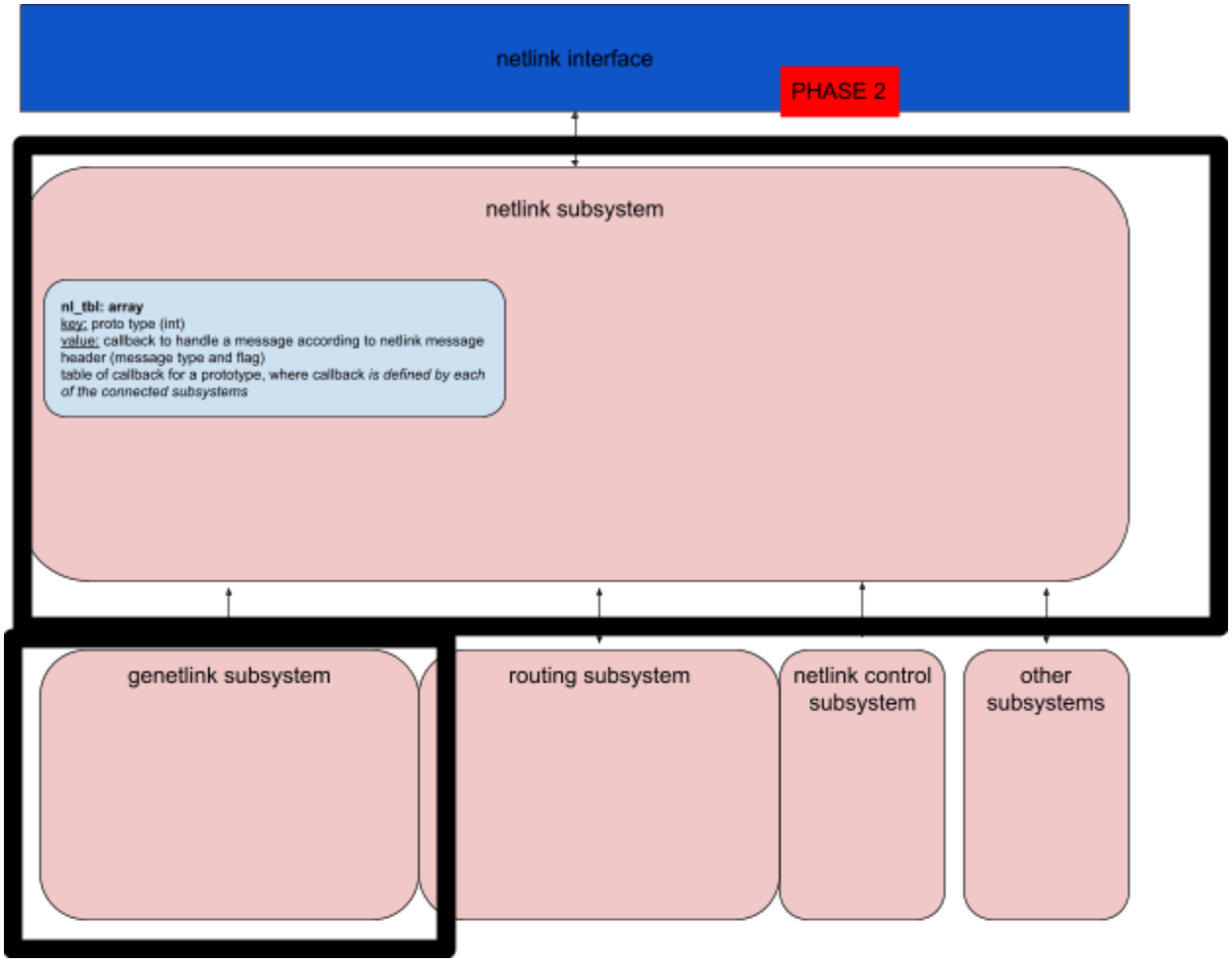
Development Milestones

Phase 1



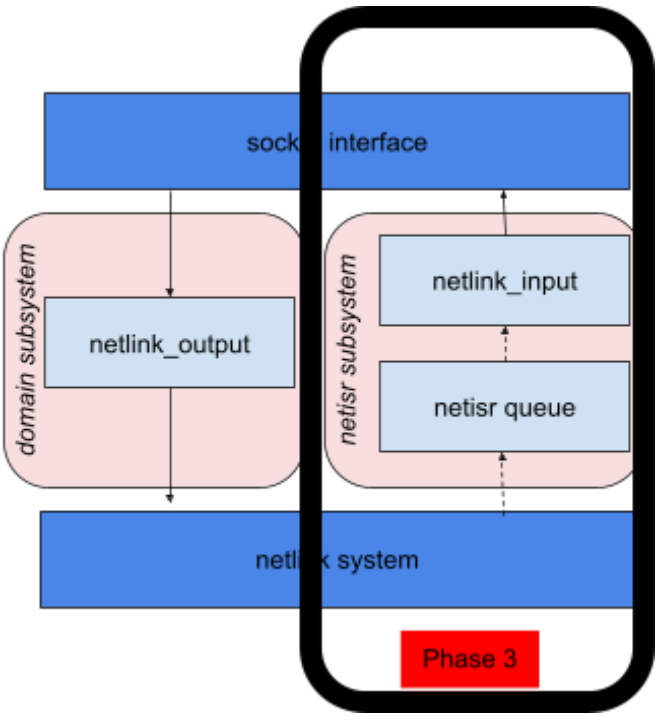
Description: Define interfaces to interact the domain subsystem with the netlink subsystem
Goal: Calls the functions when the sockets are open and message is sent
Time: 1 week
Detailed Description Section 2, 3, 4

Phase 2



Description: Defining the netlink subsystem, defining the genetlink subsystem and interfacing both. Does not have to pimplement detailed control subsystem for generic netlink.
Goal: Ensure that the right functions are called when calling a generic socket
Time: 1-2 weeks
Detailed Description Section 4

Phase 3



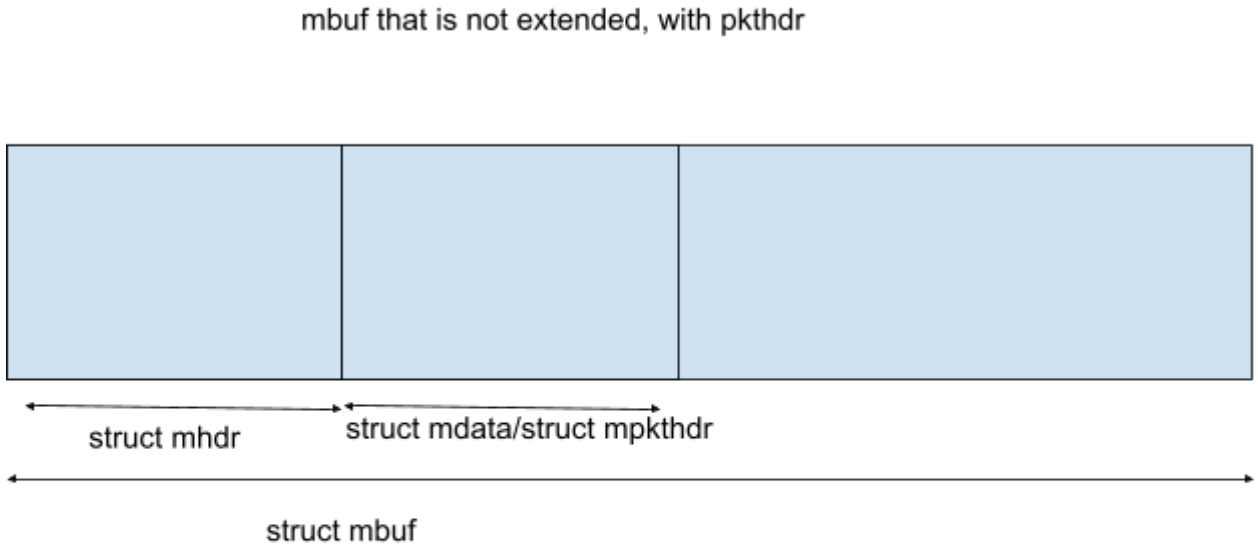
Description: Interfacing with netisr subsystem
Goal: Ensures that the socket interface should receive a message
Time: 1 week
Detailed Description Section 1, 5

Phase 4

[TBC, will confirm at a later date]

Appendix

My understanding about each packet



1 mbuf chain, 1 packet

Previous header (to be phased out)

```
05 A message header consists of one of the following:
06 .Bd -literal
07 struct rt_msghdr {
08     u_short rtm_msglen;         /* to skip over non-understood messages */
09     u_char  rtm_version;        /* future binary compatibility */
10     u_char  rtm_type;           /* message type */
11     u_short rtm_index;          /* index for associated ifp */
12     int     rtm_flags;          /* flags, incl. kern & message, e.g. DONE */
13     int     rtm_addrs;          /* bitmask identifying sockaddrs in msg */
14     pid_t   rtm_pid;            /* identify sender */
15     int     rtm_seq;            /* for sender to identify action */
16     int     rtm_errno;          /* why failed */
17     int     rtm_fmask;          /* bitmask used in RTM_CHANGE message */
18     u_long  rtm_inits;          /* which metrics we are initializing */
19     struct  rt_metrics rtm_rmx;     /* metrics themselves */
20 };
21
22 struct if_msghdr {
```

/* rtm_type */

```
270
271 /*
272  * Message types.
273  *
274  * The format for each message is annotated below using the following
275  * identifiers:
276  *
277  * (1) struct rt_msghdr
278  * (2) struct ifa_msghdr
279  * (3) struct if_msghdr
280  * (4) struct ifma_msghdr
281  * (5) struct if_announcemsghdr
282  *
283  */
284 #define RTM_ADD     0x1 /* (1) Add Route */
285 #define RTM_DELETE  0x2 /* (1) Delete Route */
286 #define RTM_CHANGE  0x3 /* (1) Change Metrics or flags */
287 #define RTM_GET     0x4 /* (1) Report Metrics */
288 #define RTM_LOSING  0x5 /* (1) Kernel Suspects Partitioning */
289 #define RTM_REDIRECT    0x6 /* (1) Told to use different route */
290 #define RTM_MISS    0x7 /* (1) Lookup failed on this address */
291 #define RTM_LOCK    0x8 /* (1) fix specified metrics */
292             /*  0x9  */
293             /*  0xa  */
294 #define RTM_RESOLVE 0xb /* (1) req to resolve dst to LL addr */
```

in route.h

/* rtm_flags */

```
172 #define RTF_UP      0x1     /* route usable */
173 #define RTF_GATEWAY 0x2     /* destination is a gateway */
174 #define RTF_HOST    0x4     /* host entry (net otherwise) */
175 #define RTF_REJECT  0x8     /* host or net unreachable */
176 #define RTF_DYNAMIC 0x10        /* created dynamically (by redirect) */
177 #define RTF_MODIFIED    0x20        /* modified dynamically (by redirect) */
178 #define RTF_DONE    0x40        /* message confirmed */
179 /*      0x80            unused, was RTF_DELCLONE */
180 /*      0x100           unused, was RTF_CLONING */
181 #define RTF_XRESOLVE    0x200       /* external daemon resolves name */
182 #define RTF_LLINFO  0x400       /* DEPRECATED - exists ONLY for backward
183                    compatibility */
184 #define RTF_LLDATA  0x400       /* used by apps to add/del L2 entries */
185 #define RTF_STATIC  0x800       /* manually added */
186 #define RTF_BLACKHOLE   0x1000      /* just discard pkts (during updates) */
187 #define RTF_PROTO2  0x4000      /* protocol specific routing flag */
188 #define RTF_PROTO1  0x8000      /* protocol specific routing flag */
189 /*      0x10000         unused, was RTF_PRCLONING */
190 /*      0x20000         unused, was RTF_WASCLONED */
191 #define RTF_PROTO3  0x40000     /* protocol specific routing flag */
192 #define RTF_FIXEDMTU    0x80000     /* MTU was explicitly specified */
193 #define RTF_PINNED  0x100000    /* route is immutable */
194 #define RTF_LOCAL   0x200000    /* route represents a local address */
195 #define RTF_BROADCAST   0x400000    /* route represents a bcast address */
196 #define RTF_MULTICAST   0x800000    /* route represents a mcast address */
197                 /* 0x8000000 and up unassigned */
198 #define RTF_STICKY   0x10000000 /* always route dst->src */
199
200 /*      0x40000000      unused, was RTF_RNH_LOCKED */
201
202 #define RTF_GWFLAG_COMPAT 0x80000000    /* a compatibility bit for interacting
203                    with existing routing apps */
204
205 /* Mask of RTF flags that are allowed to be modified by RTM_CHANGE. */
206 #define RTF_FMASK   \
207     (RTF_PROTO1 | RTF_PROTO2 | RTF_PROTO3 | RTF_BLACKHOLE | \
208      RTF_REJECT | RTF_STATIC | RTF_STICKY)
209
```

/*rtm_addrs*/

```
315
316 /*
317  * Bitmask values for rtm_addrs.
318  */
319 #define RTA_DST     0x1 /* destination sockaddr present */
320 #define RTA_GATEWAY 0x2 /* gateway sockaddr present */
321 #define RTA_NETMASK 0x4 /* netmask sockaddr present */
322 #define RTA_GENMASK 0x8 /* cloning mask sockaddr present */
323 #define RTA_IFP     0x10    /* interface name sockaddr present */
324 #define RTA_IFA     0x20    /* interface addr sockaddr present */
325 #define RTA_AUTHOR  0x40    /* sockaddr for author of redirect */
326 #define RTA_BRD     0x80    /* for NEWADDR, broadcast or p-p dest addr */
```

/*rtm_inits*/

```
303 /*
304  * Bitmask values for rtm_inits and rmx_locks.
305  */
306 #define RTV_MTU     0x1 /* init or lock _mtu */
307 #define RTV_HOPCOUNT    0x2 /* init or lock _hopcount */
308 #define RTV_EXPIRE  0x4 /* init or lock _expire */
309 #define RTV_RPIPE   0x8 /* init or lock _recvpipe */
310 #define RTV_SPIPE   0x10    /* init or lock _sendpipe */
311 #define RTV_SSTHRESH    0x20    /* init or lock _ssthresh */
312 #define RTV_RTT     0x40    /* init or lock _rtt */
313 #define RTV_RTTVAR  0x80    /* init or lock _rttvar */
314 #define RTV_WEIGHT  0x100   /* init or lock _weight */
```

currently using

```
228 struct rtmsg {
229     unsigned char       rtm_family;
230     unsigned char       rtm_dst_len;
231     unsigned char       rtm_src_len;
232     unsigned char       rtm_tos;
233
234     unsigned char       rtm_table;  /* Routing table id */
235     unsigned char       rtm_protocol;   /* Routing protocol; see below  */
236     unsigned char       rtm_scope;  /* See below */
237     unsigned char       rtm_type;   /* See below    */
238
239     unsigned        rtm_flags;
240 };
241
```

rtm_family

```
rtm->rtm_family = AF_INET;
rtm->rtm_family = AF_INET6;
```

nlmsg_type

```
22 /* Types of messages */
23
24 //enum {
25 //  RTM_BASE    = 16,
26 //#define RTM_BASE  RTM_BASE
27 //
28 //  RTM_NEWLINK = 16,
29 //#define RTM_NEWLINK   RTM_NEWLINK
30 //  RTM_DELLINK,
31 //#define RTM_DELLINK   RTM_DELLINK
32 //  RTM_GETLINK,
33 //#define RTM_GETLINK   RTM_GETLINK
34 //  RTM_SETLINK,
35 //#define RTM_SETLINK   RTM_SETLINK
36 //
37 //  RTM_NEWADDR = 20,
38 //#define RTM_NEWADDR   RTM_NEWADDR
39 //  RTM_DELADDR,
40 //#define RTM_DELADDR   RTM_DELADDR
41 //  RTM_GETADDR,
42 //#define RTM_GETADDR   RTM_GETADDR
```

rtm_type

```
enum {
    RTN_UNSPEC,
    RTN_UNICAST,        /* Gateway or direct route      */
    RTN_LOCAL,          /* Accept locally               */
    RTN_BROADCAST,      /* Accept locally as broadcast,
                   send as broadcast */
    RTN_ANYCAST,        /* Accept locally as broadcast,
                   but send as unicast */
    RTN_MULTICAST,      /* Multicast route              */
    RTN_BLACKHOLE,      /* Drop                         */
    RTN_UNREACHABLE,    /* Destination is unreachable   */
    RTN_PROHIBIT,       /* Administratively prohibited  */
    RTN_THROW,          /* Not in this table            */
    RTN_NAT,            /* Translate this address       */
    RTN_XRESOLVE,       /* Use external resolver        */
    __RTN_MAX
};
```

rtm_protocol

```
/* rtm_protocol */

#define RTPROT_UNSPEC           0
#define RTPROT_REDIRECT         1       /* Route installed by ICMP redirects;
                                           not used by current IPv4 */
#define RTPROT_KERNEL           2       /* Route installed by kernel            */
#define RTPROT_BOOT             3       /* Route installed during boot          */
#define RTPROT_STATIC           4       /* Route installed by administrator     */

/* Values of protocol >= RTPROT_STATIC are not interpreted by kernel;
   they are just passed from user and back as is.
   It will be used by hypothetical multiple routing daemons.
   Note that protocol values should be standardized in order to
   avoid conflicts.
 */

#define RTPROT_GATED            8       /* Apparently, GateD */
#define RTPROT_RA               9       /* RDISC/ND router advertisements */
#define RTPROT_MRT              10      /* Merit MRT */
```

rtm_flags

#define RTM_F_NOTIFY 0x100 /* Notify user of route change */
#define RTM_F_CLONED 0x200 /* This route is cloned */
#define RTM_F_EQUALIZE 0x400 /* Multipath equalizer: NI */
#define RTM_F_PREFIX 0x800 /* Prefix addresses */
#define RTM_F_LOOKUP_TABLE 0x1000 /* set rtm_table to FIB lookup result */
#define RTM_F_FIB_MATCH 0x2000 /* return full fib lookup match */
#define RTM_F_OFFLOAD 0x4000 /* route is offloaded */
#define RTM_F_TRAP 0x8000 /* route is trapping packets */
#define RTM_F_OFFLOAD_FAILED 0x20000000 /* route offload failed, this value
* is chosen to avoid conflicts with
* other flags defined in
* include/uapi/linux/ipv6_route.h
*/

need to convert to

```
struct rt_addrinfo {
    int rti_addrs;          /* Route RTF_ flags */
    int rti_flags;          /* Route RTF_ flags */
    struct  sockaddr *rti_info[RTAX_MAX];   /* Sockaddr data */
    struct  ifaddr *rti_ifa;        /* value of rt_ifa addr */
    struct  ifnet *rti_ifp;         /* route interface */
    rib_filter_f_t  *rti_filter;        /* filter function */
    void    *rti_filterdata;        /* filter paramenters */
    u_long  rti_mflags;         /* metrics RTV_ flags */
    u_long  rti_spare;          /* Will be used for fib */
    struct  rt_metrics *rti_rmx;        /* Pointer to route metrics */
};
```

sockaddr array

```
327
328 /*
329  * Index offsets for sockaddr array for alternate internal encoding.
330  */
331 #define RTAX_DST    0   /* destination sockaddr present */
332 #define RTAX_GATEWAY    1   /* gateway sockaddr present */
333 #define RTAX_NETMASK    2   /* netmask sockaddr present */
334 #define RTAX_GENMASK    3   /* cloning mask sockaddr present */
335 #define RTAX_IFP    4   /* interface name sockaddr present */
336 #define RTAX_IFA    5   /* interface addr sockaddr present */
337 #define RTAX_AUTHOR 6   /* sockaddr for author of redirect */
338 #define RTAX_BRD    7   /* for NEWADDR, broadcast or p-p dest addr */
339 #define RTAX_MAX    8   /* size of array to allocate */
340
```

Field (rt_addr_info)	Source (rt_msghdr)	New Source (rtmsg)
rti_mflags	rtm_inits	nlattr
rtm_rmx	rtm_rmx	
rti_flags	rtm_flags	rtm_flags [or can't convert..?]
rti_addrs	rtm_addrs	populated using nl attributes
rti_info	socket addresses are layed out in order after the rtmsg_hdr. it is "initialized" by considering the flags, and will be in the same order	

kgdb /boot/kernel/kernel /var/crash/vmcore.last
./rtnl-route-add xn0 10.0.1.12 32 172.31.18.62
./rtnl-route-get 10.0.1.12
route delete 10.0.1.12

Source	Source (rt_msghdr)	New Source (rtmsg)
	rtm_type	nlmsg_type
	rtm_falgs (subset)	rtm_type

Table for RTM_GETROUTE
Problem: rtsock uses rib_cmd_info and nhop_object to update rt_msghdr
I need to use: rib_cmd_info and nh_object to update struct rtmsg

Source(rt_msghdr)	Linux Source (`struct fib_rt_info`)	Output (rtmsg)
	fri->type	rtm->rtm_type
	fri->scope	rtm->rtm_scope
	fri->fi->fib_flags	rtm->rtm_flags
look at rib_action 	fri->tb_id	rtm->rtm_table
	fi->rtm_protocol	rtm->rtm_protocol

What corresponds to the table?

```
71 struct sockaddr_in {
72     uint8_t     sin_len;
73     sa_family_t     sin_family;
74     in_port_t   sin_port;
75     struct in_addr  sin_addr;
76     char        sin_zero[8];
77 };
```

```
327  */
328 struct sockaddr {
329     unsigned char   sa_len;     /* total length */
330     sa_family_t sa_family;  /* address family */
331     char        sa_data[14];    /* actually longer; address value */
332 };
```

Hey Alex,

I got something simple for RTM_GETROUTE working!

1. Note: There does not appear to be corresponding mappings for the following fields in “struct rtmsg” (netlink’s routing message).

rtm_type - linux retrieves this straight from its version of `const struct fib_rt_info`. freebsd's fib_rt_info does not have this information
rtm_table - refers to *routing* table, and I don't know where this information is.
rtm_scope - the fields don't make sense to freebsd
rtm_flags - [not the same as rt_msg_hdr's rtm_flags] nothing corresponding

2. Is there one routing table? Or multiple routing table?

OIF = rtm_index
IFA/IFP not present in rtmsg

is PRIORITY = Weight?

```
33    if (tb[RTA_TABLE]) {
34        printf("table=%u ", mnl_attr_get_u32(tb[RTA_TABLE]));
35    }
```

Not sure. Only 1 routing table?

```
36    if (tb[RTA_DST]) {
37        struct in_addr *addr = mnl_attr_get_payload(tb[RTA_DST]);
38        printf("dst=%s ", inet_ntoa(*addr));
39    }
```

Done

```
40    if (tb[RTA_SRC]) {
41        struct in_addr *addr = mnl_attr_get_payload(tb[RTA_SRC]);
42        printf("src=%s ", inet_ntoa(*addr));
43    }
```

```
44    if (tb[RTA_OIF]) {
45        printf("oif=%u ", mnl_attr_get_u32(tb[RTA_OIF]));
46    }
```

Done

```
47     if (tb[RTA_FLOW]) {
48         printf("flow=%u ", mnl_attr_get_u32(tb[RTA_FLOW]));
49     }
```
Not sure

```
50     if (tb[RTA_PREFSRC]) {
51         struct in_addr *addr = mnl_attr_get_payload(tb[RTA_PREFSRC]);
52         printf("prefsrc=%s ", inet_ntoa(*addr));
53     }
```
Not sure

```
54     if (tb[RTA_GATEWAY]) {
55         struct in_addr *addr = mnl_attr_get_payload(tb[RTA_GATEWAY]);
56         printf("gw=%s ", inet_ntoa(*addr));
57     }
```
Done

```
58     if (tb[RTA_PRIORITY]) {
59         printf("prio=%u ", mnl_attr_get_u32(tb[RTA_PRIORITY]));
60     }
```
Not sure

```
61     if (tb[RTA_METRICS]) {
62         int i;
63         struct nlattr *tbx[RTAX_MAX+1] = {};
64
65         mnl_attr_parse_nested(tb[RTA_METRICS], data_attr_cb2, tbx);
66
67         for (i=0; i<RTAX_MAX; i++) {
68             if (tbx[i]) {
69                 printf("metrics[%d]=%u ",
70                     i, mnl_attr_get_u32(tbx[i]));
71             }
72         }
73     }
74 }
```