

class pfNETLINKsocket

Juha Hyttinen

August 1, 2000

Abstract

OVOPS++ has many kind socket interfaces. This document describes netlink socket implementation in OVOPS++. It also describes how netlink socket is used and how there can be added more capabilities to it. Netlink socket implementation work only with Linux.

Contents

1	Introduction	1
2	NETLINK_ROUTE	2
2.1	Sending requests to kernel	2
2.2	Receiving information from kernel	2
2.3	Some general information	3
2.4	Frames	3
2.4.1	Frame for GET requests and common header for all frame types	3
2.4.2	IPv4 Route information frame	4
2.4.3	IPv6 Route information frame	4
2.4.4	IPv4 Address information frame	5
2.4.5	IPv6 Address information frame	6
2.4.6	Link information frame	7
2.4.7	Message DONE frame	8
2.4.8	Error frame	8
3	Description for each of pfNETLINK class methods	9
3.1	class pfNETLINKsocket : public pfDevice	9
3.1.1	public:	9
3.1.2	protected:	9
3.1.2.1	NETLINK_ROUTE	10
3.1.2.1.1	RTM_[ADD/DEL/GET]ROUTE	10
3.1.2.1.2	RTM_GETADDR	10
3.1.2.1.3	RTM_GETLINK	10
3.1.2.1.4	route attribute parsing methods	11
4	Example code of netlink adapter: routeAdapter	11

1 Introduction

pfNETLINK class defines socket for netlink. Netlink can be used for handling kernel information according different kind things. For example routetable information. When socket is opened, there must be give family type for netlink. So when socket is opened

```
socket(AF_NETLINK, SOCK_RAW, _netlinkFamily)
```

`_netlinkfamily` defines what kind information netlink handles.

For now there is implemented family types for `NETLINK_ROUTE` for IPv4 and IPv6. Implementation means porting information from `pfFrame` format to kernel message format and vice versa. `pfFrame` format is used for route information presentation, because it is easy to use. `pfFrame` structures are defined in section 2.4. Porting implementation is implemented in overwritten methods `writeCallBack` and `readCallBack`. `pfNETLINK` derives from `pfDevice`, which contains original `readCallBack` and `writeCallBack`. There is only added `switch` -statement to these two methods, which check what is netlink family type (family type is stored to object when new socket is opened). And for matching family type is called right method. So there is now implementations only for `NETLINK_ROUTE` in `readCallBack` and `writeCallBack` methods.

2 NETLINK_ROUTE

There is two cases when changing information with kernel through `NETLINK`. When sending information to kernel and when receiving. When we are sending information to kernel we have to parse information from `pfFrame` to kernel message and when receiving information we have to parse information from kernel message to `pfFrame`. Both of these cases are divided in to two phases.

2.1 Sending requests to kernel

So there is two level when parsing requests to kernel. In first level we parse from `pfFrame` one request at time and build netlink message for single request. This netlink message is added to `sendbuffer`, which contains hole kernel message. So `pfFrame` can contain several requests to kernel.

So when `writeCallBack` is called and netlink family type is `NETLINK_ROUTE` then *`parseRequestToNetlinkRoute(...)`* method is called. This method parses one request from received `pfFrame` and then this one request is parsed by using correct method for it (listed below in section 2.4).

There is implemented for `NETLINK_ROUTE` only `RTM_GETROUTE`, `RTM_NEWROUTE`, `RTM_DELROUTE`, `RTM_GETADDR` and `RTM_GETLINK` message types. See manpages for `rtnetlink` for other `NETLINK_ROUTE` message types.

2.2 Receiving information from kernel

So there is two level when parsing information from kernel. In first level we parse from kernel information (is in `recvbuffer`) one netlink message at time and then add its contained information to end of `pfFrame`. So message which is received from kernel can contain several separated information.

So when `readCallBack` is called and netlink family type is `NETLINK_ROUTE` then is *`parseInformationFromNetlinkRoute(...)`* method called. This method parses one netlink message from hole kernel message. Then it's compared which kind information it is and committed parse method for single netlink message (below is listed in section 2.4 all parse methods for single entry, from `pfFrame` to kernel message and vice versa).

There is implemented for `NETLINK_ROUTE` only `RTM_GETROUTE`, `RTM_NEWROUTE`, `RTM_DELROUTE`, `RTM_GETADDR` and `RTM_GETLINK` message types. See manpages for `rtnetlink` for other `NETLINK_ROUTE` message types.

Notice: If returned frame size is zero then there is bigger error in parsing. Normal errors from kernel are received in specified format, which are ported to frame, which is defined in section 2.4.

Lower method adds to netlink message the information for different message types, for example RTM_ADDROUTE. It add one attribute at time. upper parses attributes from message.

```
void parseattr(struct rtattr **tb_, int max_, struct rtattr *rta_, int len_);
int addattr_l(struct nlmsghdr *nlHeader_, unsigned int maxlen_, int type_, void
*data_, unsigned int alen_);
```

2.4 Frames

If information, which is can be received trough rtnetlink (NETLINK_ROUTE), is compared with implemented part in pfNETLINK, will be noticed that, that all information isn't implemented to parse routines. But if there is need to add additional information to pfFrame it can be added to end of frame, then it won't harm other methods (If we assume that "msg len field" in frame is used to parse information at user plane. This is done by modifying methods which parses single entry to another format (Next methods are for rtnetlink message and pfFrame parses):

Below is described frames. All frames has same kind heading, which contains netlink message information. There is additional information "msg tot len", which informs one informations (entry's) length in 32 bit words.

2.4.1 Frame for GET requests and common header for all frame types

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          nl_msg type          |          nl_flags          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          sequence          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          pid          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          msg_tot_len (in 32 bit words)          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

```

|                                     rtm_family = AF_INET/AF_INET6                                     |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

2.4.2 IPv4 Route information frame

This is for IPv4 route table entry information. Requests and received information is contained in this frame. Implemented types are: nl_msg type = RTM_NEWROUTE, RTM_DELROUTE

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          nl_msg type          |          nl_flags          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          sequence          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          pid          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          msg tot len (in 32 bit words)          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          rtm_family = AF_INET          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|  rtm_table  |  rtm_scope  |  rtm_protocol  |  rtm_type  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          rta_destination          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          rta_gateway          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          rta_generalmask          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          rta_metric          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          rta_priority          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          rta_output interface          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          rta_flags          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

2.4.3 IPv6 Route information frame

This is for IPv6 route table entry information. Requests and received information is contained in this frame. Implemented type are: nl_msg type = RTM_NEWROUTE, RTM_DELROUTE

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          nl_msg type          |          nl_flags          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          sequence          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          pid          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          msg tot len (in 32 bit words)          |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|          rtm_family = AF_INET6          |

```



```

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     msg tot len (in 32 bit words)         |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     rtm_family = AF_INET                   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   flags       |   scope       |   interface index       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     interface address                     |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     local address                         |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     broadcast addr                       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     anycast addr                         |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     address mask                         |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   prefix length       |   label length in bytes       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     label                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     ...           pads if needed         |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

Address mask is also informed as prefix length.

2.4.5 IPv6 Address information frame

This frame is used to get address information. This means with this we will get information for each hardware. Hardware's own information is received with RTM_GETLINK messages which is described after ADDR frames. Current there is implemented only request side: nl_msg = RTM_GETADDR. Its mean that it's only possible to get information but not change. So when kernel sends information to up, will pfNETLINK parse information for address-data to pfFrame, which is showed below (Just one entry.) pfFrame, which pfNetlink sends to up can contain more than on e entry).Notice that in IPv6 there isn't broadcast address.

```

      0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   nl_msg type       |   nl_flags       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               sequence               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               pid               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               msg tot len (in 32 bit words)       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               rtm_family = AF_INET               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|   flags       |   scope       |   interface index       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               interface address                     |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|               interface address                     |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

```

|                                     interface address                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     interface address                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     local address                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     local address                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     local address                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     local address                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     anycast addr                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     anycast addr                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     anycast addr                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     anycast addr                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     address mask                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     address mask                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     address mask                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     address mask                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|      prefix length           |      label length in bytes           |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     label                                     |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|                                     ...           pads if needed           |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+

```

Address mask is also informed as prefix length.

2.4.6 Link information frame

This frame describes link layer hardware. Its return hardware address for example for network device. Current there is implemented only request side: `nl_msg = RTM_GETLINK`. Its mean that it's only possible to get information but not change. So when kernel sends information to up, will `pfNETLINK` parse information for link-data to `pfFrame`, which is showed below (Just one entry). `pfFrame`, which `pfNetlink` sends to up can contain more than one entry)

```

0                                     1                                     2                                     3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|      nl_msg type           |      nl_flags           |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|      sequence           |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|      pid           |
+-+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|      msg tot len (in 32 bit words)           |

```

```

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                               rtm_family = AF_INET                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           dev type           |           interface index           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           flags              |           0                        |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           Link type          |           mtu                |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           hardware addr length                                     |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           interface hware address                               |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           ...                               Pads if needed       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           interface hware broadcast addr                       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           ...                               Pads if needed       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           0              |           label length in bytes           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           label                                                  |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           ...              pads if needed                       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

2.4.7 Message DONE frame

If there is no more information for sequence number, will be this kind information received. When message done is received , will be next frame build with message type : nl_msg type = NLMSG_DONE

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           nl_msg type           |           nl_flags           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           sequence              |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           pid                   |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           msg tot len           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

2.4.8 Error frame

When error is received will be this frame received. If error number is zero, it's means success. So when error frame is created it has message type set to: nl_msg type = NLMSG_ERROR. This means that there is error in request. Notice that empty pfFrame is returned if there is error in parsing methods. These two error cases are different kind.

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           nl_msg type           |           nl_flags           |

```



```

+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     sequence                                     |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     pid                                       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     msg tot len                             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     error number                             |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|      nl_msg type                    |      nl_msg flags                    |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     sequence                                     |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|                                     pid                                       |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

```

3 Description for each of pfNETLINK class methods

3.1 class pfNETLINKsocket : public pfDevice

3.1.1 public:

- explicit pfNETLINKsocket(pfUlong bufferSize _); : This is constructor for pfNETLINKsocket. Given value is size of read- and writebuffers.
- virtual ~pfNETLINKsocket(void); : Destructor. This also free read -and writebuffers memories
- virtual bool openDevice(int netlinkFamily _); : Open netlink - socket for given netlink family type. For Example NETLINK_ROUTE
- void setFamily(int netlinkFamily _); : Saves given netlink family type to object
- int getFamily(void); : Get saved netlinkfamily type from object
- void setNlAddr(const struct sockaddr_nl &addr _); : Set netlink socket address struct to object.
- const struct sockaddr_nl &getNlAddr(void) const; : Get netlink socket address struct from object.
- virtual pfBoolean isUsingAcknowledge(void); : Return true if acknowledgement is “on” and false if it is “off”. This is for requests, BUT NOT FOR GET REQUESTS. If you add, or can be say that you request kernel to add, a route to system routingtable, acknowledgement is received if this is “on”. Notice that if GET request is sent, there should be appear, after information is received, NL_MSGDONE message and acknowledgement is not needed.
- virtual void setUsageOfAcknowledge(pfBoolean ack _); : Set acknowledge flag on/ off. See above.

3.1.2 protected:

- void freeBuffers(void); : Free read -and writebuffers memories
- virtual int readFromSocket(void); : Lower level function for reading socket. This exactly receives data from socket.

- virtual int writeToSocket(char *start_); : Lower level function for writing socket. This exactly sends data to socket.
- virtual void readCallback(void); :readCallback is called when OVOPS++ scheduler gives time to netlink to read socket. This is overwritten, because we need to parse pfFrame format to kernel message and vice versa. So these parse methods are called from callback -methods. These call backs can be also called directly is polling status is “on”
- virtual void writeCallback(void); :writeCallback is called when OVOPS++ scheduler gives time to netlink to write socket. This is overwritten, because we need to parse pfFrame format to kernel message and vice versa. So these parse methods are called from call back -methods. These call backs can be also called directly is polling status is “on”.

3.1.2.1 NETLINK_ROUTE

- int parseRequestToNetlinkRoute(const pfFrame &fromBig_, char *to_); : This method is called from call back method for NETLINK_ROUTE informations. This parses from given pfFrame the one request at time and after it calls lower parse methods for that one entry.
- pfFrame parseInformationFromNetlinkRoute(const char *information_, const int size_); : This method is called from call back method for NETLINK_ROUTE informations. This parses from given kernel message the one info at time and after it calls lower parse methods for that one entry.

3.1.2.1.1 RTM_[ADD/DEL/GET]ROUTE

- int parseNetlinkRouteInfoForRouteFrameToMsg(const pfFrame &fromSub_, char *to_); : Parses request considering route information from user program to kernel. All types implemented. So can be added and deleted routingtable entries.
- int parseNetlinkRouteInfoForRouteMsgToFrame(const struct nlmsghdr *h, pfFrame &frame_); : Parses information considering route information from kernel to user program. All types implemented. So can be added and deleted routing table entries.

3.1.2.1.2 RTM_GETADDR

- int parseNetlinkRouteInfoForAddrFrameToMsg(const pfFrame &fromSub_, char *to_); : Parses request considering link address information from user program to kernel. Only GET request type is implemented.
- int parseNetlinkRouteInfoForAddrMsgToFrame(const struct nlmsghdr *h, pfFrame &frame_); : Parses information considering link address information from kernel to user program. To this direction all types are implemented. So returned msg type is RTM_NEWADDR.(can also be RTM_DELADDR)

3.1.2.1.3 RTM_GETLINK

- int parseNetlinkRouteInfoForLinkFrameToMsg(const pfFrame &fromSub_, char *to_); : Parses request considering link information from user program to kernel. Only GET request type is implemented.
- int parseNetlinkRouteInfoForLinkMsgToFrame(const struct nlmsghdr *h, pfFrame &frame_); : Parses information considering link information from kernel to user program. To this direction all types are implemented. So returned msg type is RTM_NEWLINK. (can also be RTM_DELLINK)

3.1.2.1.4 route attribute parsing methods

- `void parseattr(struct rtattr **tb_, int max_, struct rtattr *rta_, int len_);` : Parses from one netlink msg's payload the attributes. Notice that payload contains also rtnetlink msg, which contains the `rt_attr` messages. (Look `pfNetlink` parse methods to make clear this method functionality)
- `int addattr_l(struct nlmsghdr *nlHeader_, unsigned int maxlen_, int type_, void *data_, unsigned int alen_);` : Parses to one netlink msg's payload the attribute. Notice that payload contains also rtnetlink msg, which contains the `rt_attr` messages. (Look `pfNetlink` parse methods to make clear this method functionality)

4 Example code of netlink adapter: routeAdapter

Next there is one example how to use `pfNETLINK` in adapter. This adapter is made for routing table information updates and to monitor routing information changes. Example is not included to this document, but it should be in the same directory as this document is.