YAML Netlink

Machine readable description for netlink protocols

Jakub Kicinski

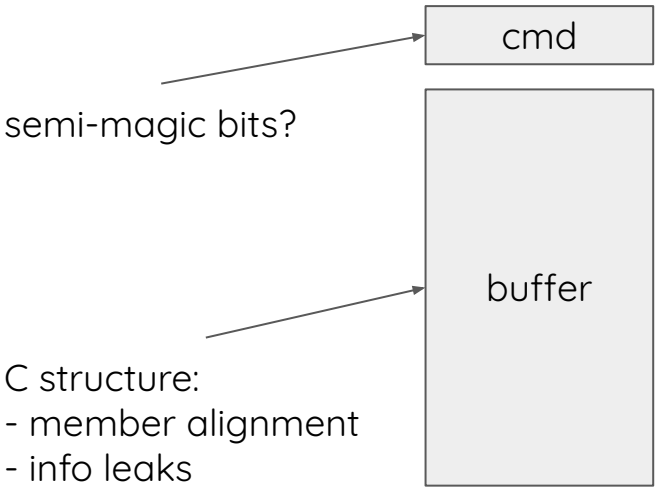
LPC 2022, Dublin

Brief history (mostly according to Wikipedia)

- Linux 2.0 - Netlink character device (Alexey Kuznetsov)
- Linux 2.2 - Netlink socket
- RFC 3549 (2003)
- Linux 2.6.14/15 (2005) - Generic Netlink, nlattr etc. (Thomas Graf)

What netlink is supposed to be

ioctl



semi-magic bits?

C structure:
- member alignment
- info leaks
- member presence
- extensibility?

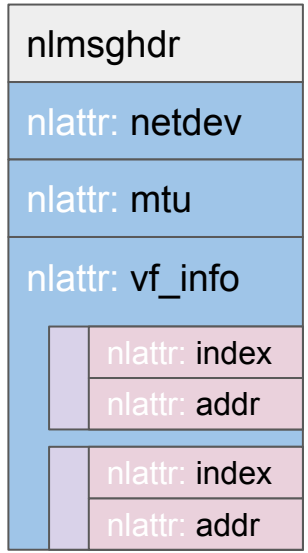
Netlink

```
cmd: RTM_SETLINK
netdev: “eth0”
mtu: 6000
vf_info: [
     index: 1,
     mac-address: 00:11:22:33:44:55
], [
     index: 2,
     mac-address: 00:11:22:33:44:56
]
```

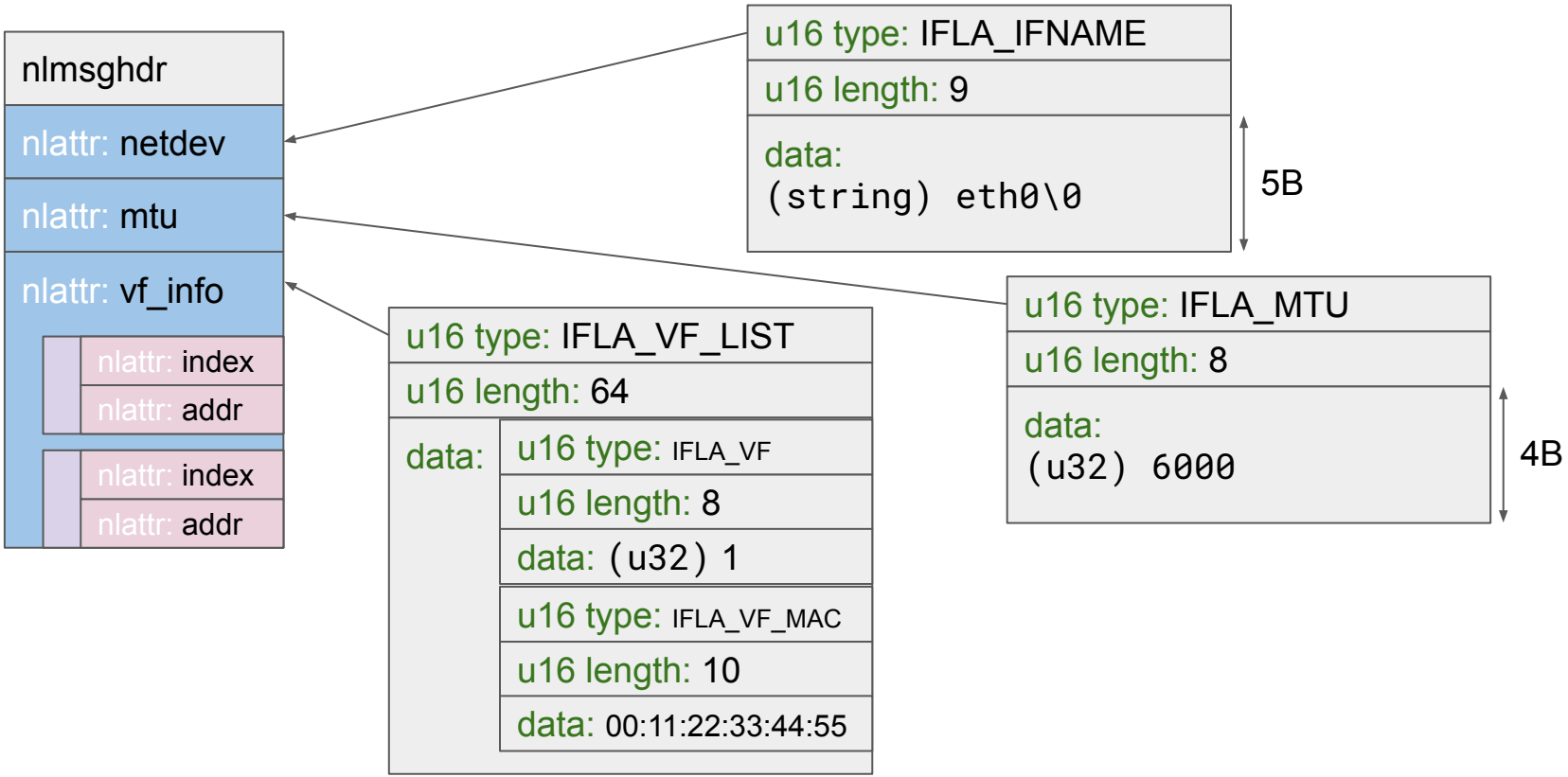
Binary format

Netlink

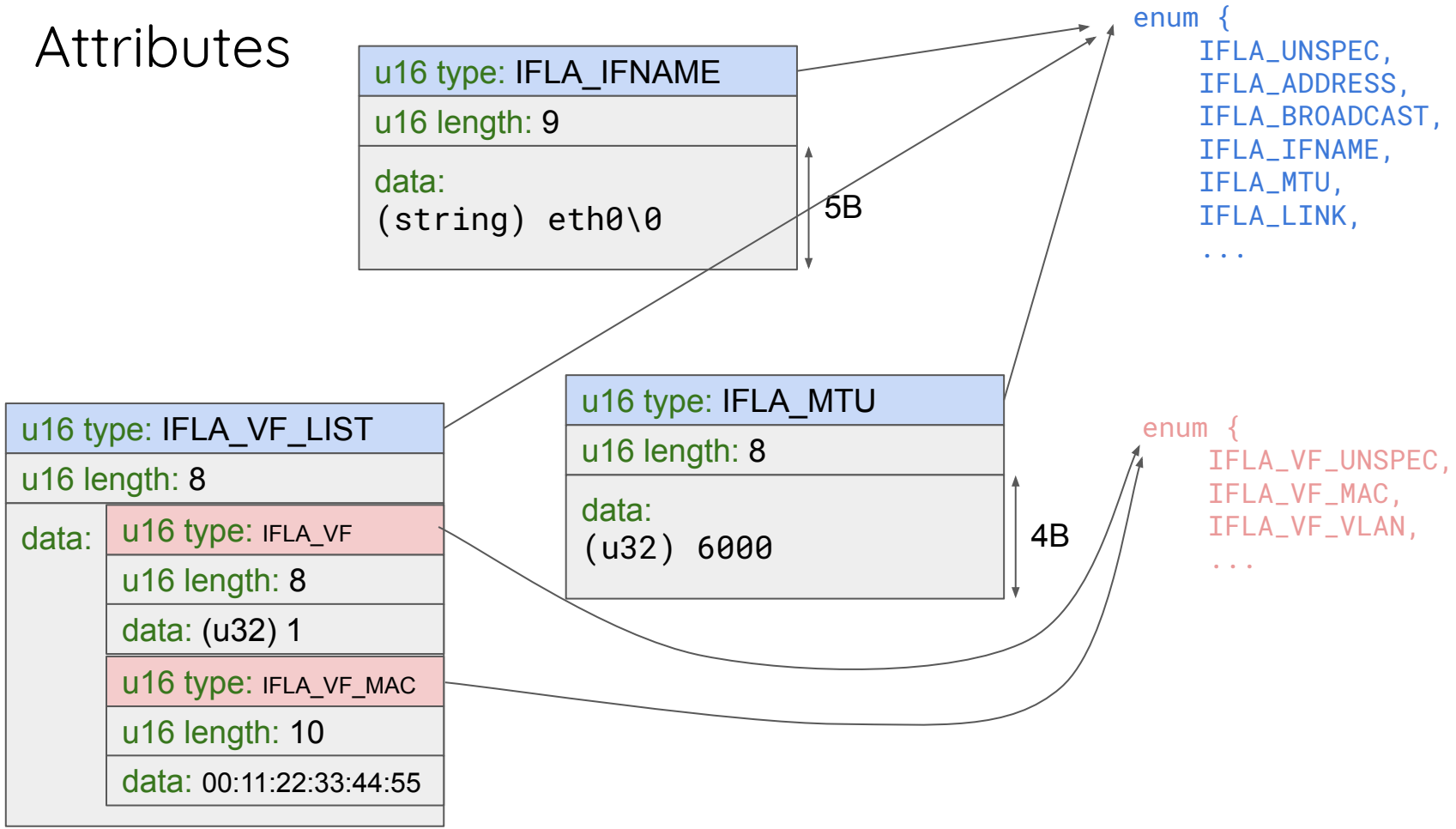
```
cmd: RTM_SETLINK
netdev: “eth0”
mtu: 6000
vf_info: [
     index: 1
     mac-address: 00:11:22:33:44:55
], [
     index: 2
     mac-address: 00:11:22:33:44:56
]
```



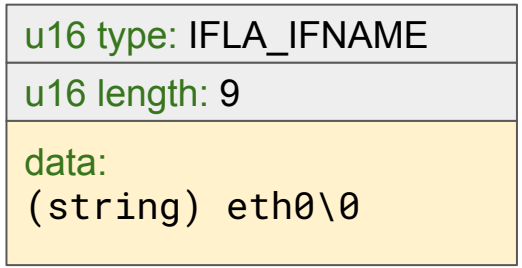
Attributes

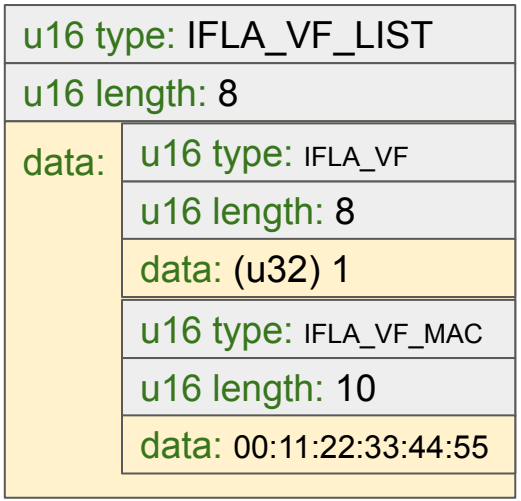

nlmsghdr
nlattr: netdev
nlattr: mtu
nlattr: vf_info
nlattr: index
nlattr: addr
nlattr: index
nlattr: addr
u16 type: IFLA_IFNAME
u16 length: 9
data:
(string) eth0\0
5B
u16 type: IFLA_MTU
u16 length: 8
data:
(u32) 6000
4B
u16 type: IFLA_VF_LIST
u16 length: 64
data:
u16 type: IFLA_VF
u16 length: 8
data: (u32) 1
u16 type: IFLA_VF_MAC
u16 length: 10
data: 00:11:22:33:44:55

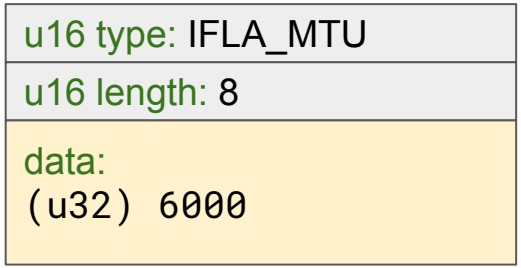
Attributes


u16 type: IFLA_IFNAME
u16 length: 9
data:
(string) eth0\0
5B
enum {
IFLA_UNSPEC,
IFLA_ADDRESS,
IFLA_BROADCAST,
IFLA_IFNAME,
IFLA_MTU,
IFLA_LINK,
...
u16 type: IFLA_MTU
u16 length: 8
data:
(u32) 6000
4B
u16 type: IFLA_VF_LIST
u16 length: 8
data:
u16 type: IFLA_VF
u16 length: 8
data: (u32) 1
u16 type: IFLA_VF_MAC
u16 length: 10
data: 00:11:22:33:44:55
enum {
IFLA_VF_UNSPEC,
IFLA_VF_MAC,
IFLA_VF_VLAN,
...

Attributes


u16 type: IFLA_IFNAME
u16 length: 9
data:
(string) eth0\0

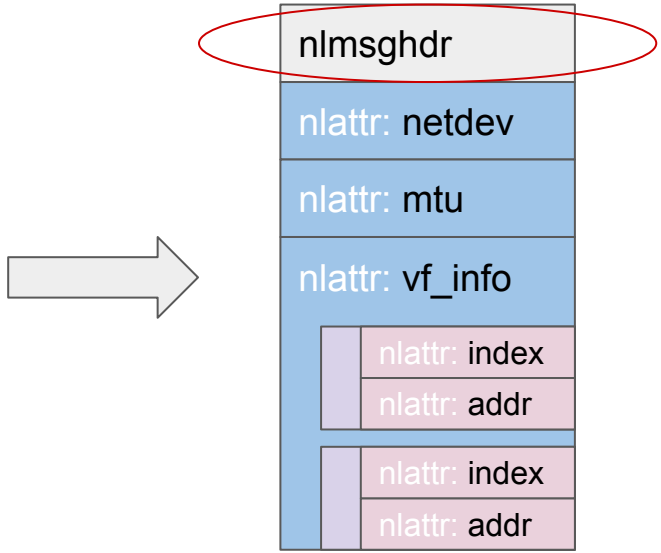

u16 type: IFLA_VF_LIST
u16 length: 8
data:
u16 type: IFLA_VF
u16 length: 8
data: (u32) 1
u16 type: IFLA_VF_MAC
u16 length: 10
data: 00:11:22:33:44:55


u16 type: IFLA_MTU
u16 length: 8
data:
(u32) 6000

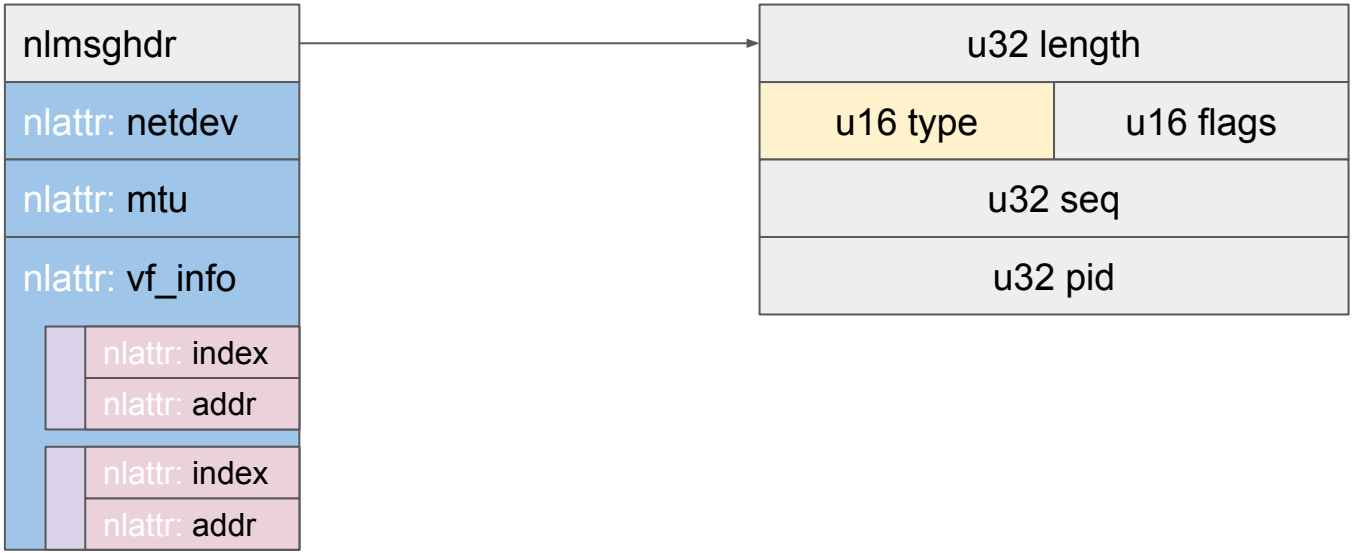
Binary format

Netlink

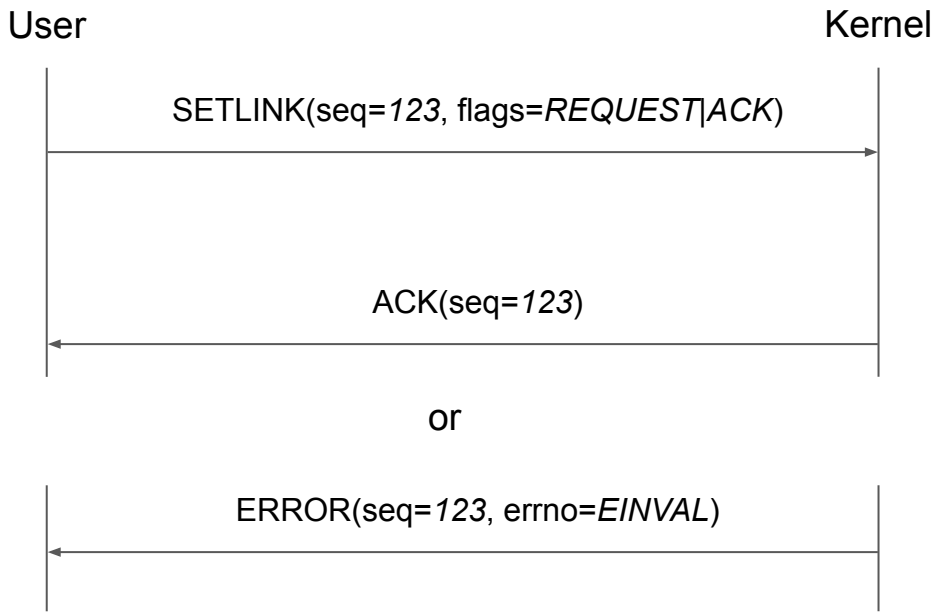
```
cmd: RTM_SETLINK
netdev: “eth0”
mtu: 6000
vf_info: [
    index: 1
    mac-address: 00:11:22:33:44:55
], [
    index: 2
    mac-address: 00:11:22:33:44:56
]
```



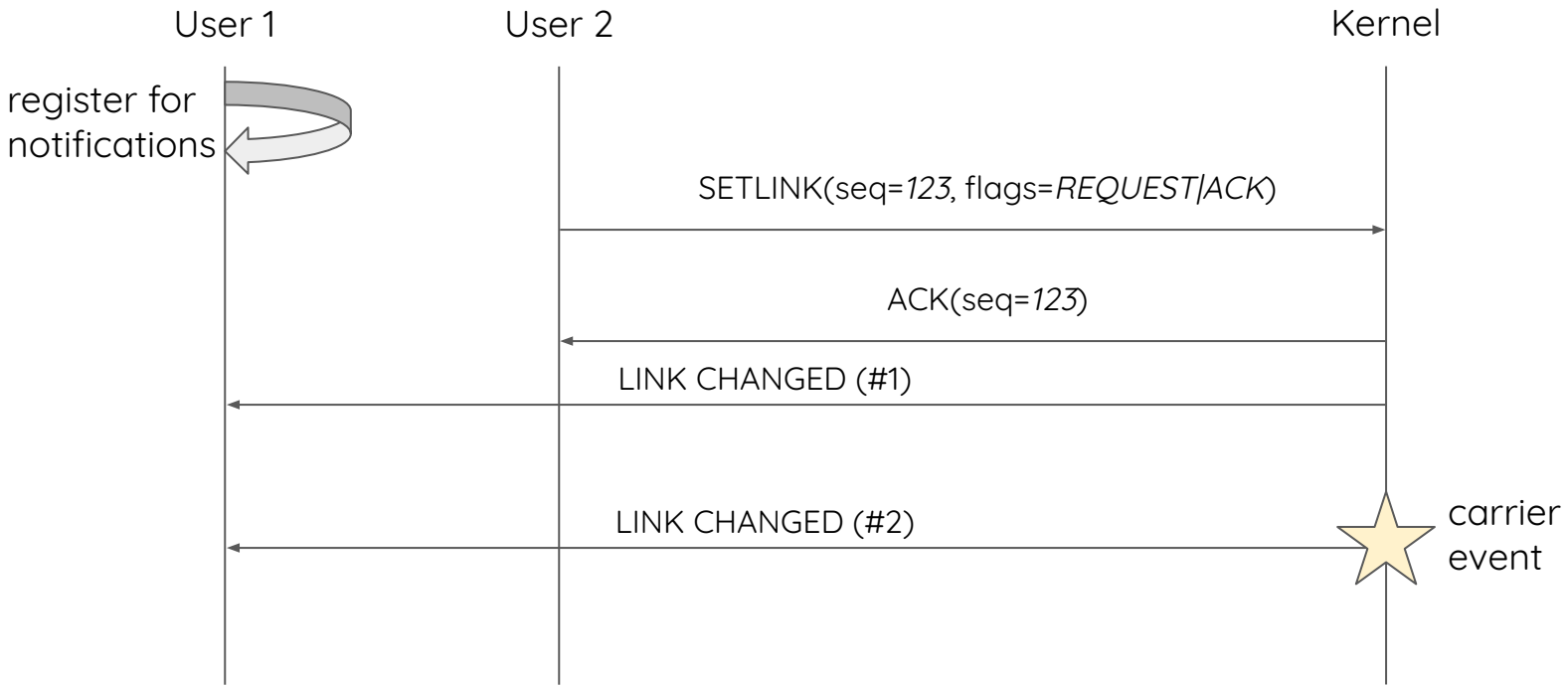
Header


nlmsghdr
nlattr: netdev
nlattr: mtu
nlattr: vf_info
nlattr: index
nlattr: addr
nlattr: index
nlattr: addr
u32 length
u16 type
u16 flags
u32 seq
u32 pid

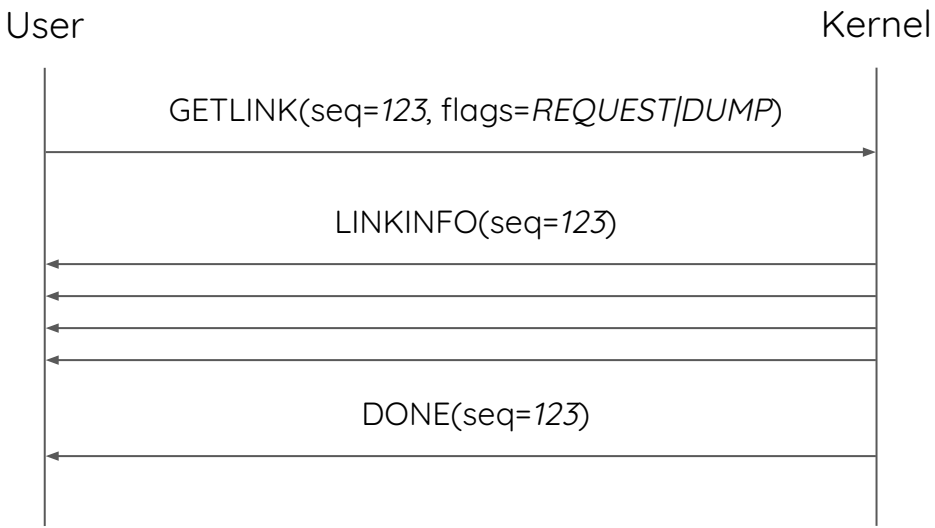
Basic exchanges - 1/3 : Do



Basic exchanges - 2/3 : Notify



Basic exchanges - 3/3 : Dump



all of the previous slides contain simplifications, lies and omissions

Netlink capabilities we don’t have time for

- **introspection** - discovering which Netlink APIs, commands, attributes are supported by the kernel, incl. attribute bounds (e.g. what flags)
- **error reporting** - kernel can return errors and warnings both in machine readable form indicating the attribute and violation type, as well as natural language messages (ext_ack)
- **local objects** - objects created by user space may be automatically cleaned up when socket is closed
- **kernel -> user upcalls** - kernel can also send requests to user space

Why YAML?

User space side of the story

1. Netlink is just a format, it does not describe the contents
2. Discover the interface
    - read documentation, read headers, read the kernel code, ask the developer
3. Translate all the kernel types (attribute, cmd ids) to the language of choice
4. Manually parse all the attributes into sane language specific types
5. Render language specific types into request
6. Keep going back to step 1 until types and attribute nesting is right
7. Profit (until you need to go back to step 1 to add support for a new cmd / attr)

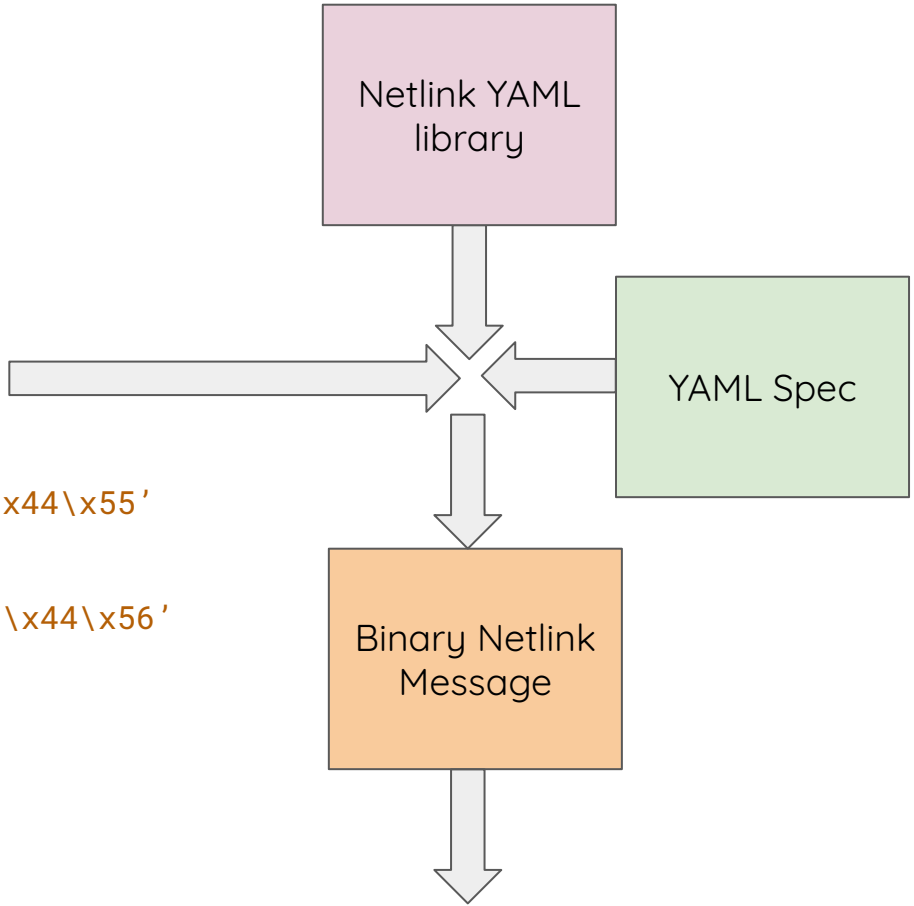
YAML Spec

```yaml
operations:
  -
    name: setlink
    attribute-set: link
    do:
      request:
        - netdev
        - mtu
        - vf-list
        ...
  -
    name: getlink
    attribute-set: link
    ...
```

```yaml
attribute-sets:
  -
    name: link
    attributes:
      -
        name: netdev
        type: string
      -
        name: mtu
        type: u32
      -
        name: vf-list
        type: nested
        multi-attr: true
        nested-attributes: vf-list
  -
    name: vf-list
    attributes:
      -
        name: vf
        type: u32
      -
        name: mac-address
        type: binary
        length: 6
```

Putting it together

```
family = YNL("iflink.yaml")
family.setlink({
  "netdev": "eth0",
  "mtu": 6000,
  "vf-list": [{
    "vf": 1,
    "mac-address": b'\x00\x11\x22\x33\x44\x55'
   }, {
     "vf": 2,
     "mac-address": b'\x00\x11\x22\x33\x44\x56'
   }],
})
```



YAML Spec

```yaml
operations:
  -
    name: getlink
    attribute-set: link
    do:
      request:
        - netdev
        - ifindex
      reply:
        - netdev
        - mtu
        - vf-list
        ...
    dump:
      reply:
        - netdev
        - mtu
        ...
```

```yaml
attribute-sets:
  -
    name: link
    attributes:
      -
        name: netdev
        type: string
      -
        name: mtu
        type: u32
      -
        name: vf-list
        type: nested
        multi-attr: true
        nested-attributes: vf-list
  -
    name: vf-list
    attributes:
      -
        name: vf
        type: u32
      -
        name: mac-address
        type: binary
        length: 6
```

YAML Spec

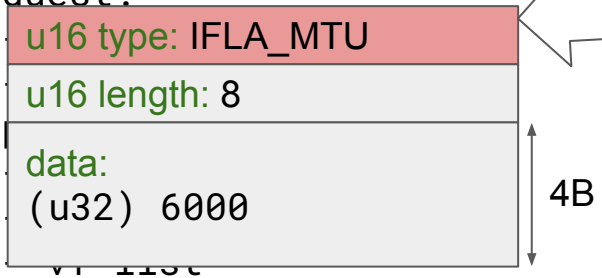
```
operations:
  -
    name: getlink
    value: 4
    attribute-set: link
    do:
      request:
        - netdev
        - ifindex
      reply:
        - netdev
        - mtu
        - vf-list
        ...
    dump:
      reply:
        - netdev
        - mtu
        ...
```

```
attribute-sets:
  -
    name: link
    attributes:
      -
        name: netdev
        type: string
        value: 1
      -
        name: mtu
        type: u32
      -
        name: vf-list
        type: nested
        value: 7
        multi-attr: true
        nested-attributes: vf-list
```

(value 2 implied)

YAML Spec

```
operations:
  -
    name: getlink
    value: 4
    attribute-set: link
    do:
      request:
        ...
      reply:
        ...
    dump:
      reply:
        - netdev
        - mtu
        ...
```




```
attribute-sets:
  -
    name: link
    attributes:
      -
        name: netdev
        type: string
        value: 1
      -
        name: mtu
        type: u32
      -
        name: vf-list
        type: nested
        value: 7
        multi-attr: true
        nested-attributes: vf-list
```

(value 2 implied)

YAML Spec

```yaml
definitions:
  -
    type: enum
    name: dpll-status
    entries: [ none, calibrating, locked ]
  -
    type: flags
    name: dpll-flags
    entries: [ sources, outputs, status ]
  -
    type: const
    name: IFNAMSIZ
    value: 16
    header: linux/if.h
```

```yaml
attribute-sets:
  -
    name: dpll
    attributes:
      -
        name: name
        type: string
        value: 1
      -
        name: status
        type: u32
        enum: dpll-status
      -
        name: flags
        type: u32
        enum: dpll-flags
```

YAML Spec - kernel bits

```
operations:
  -
    name: setlink
    attribute-set: link
    flags: [ admin-only ]
    do:
      request:
        - netdev
        - mtu
        - vf-list
        ...
  -
    name: getlink
    attribute-set: link
    ...
```

```
attribute-sets:
  -
    name: link
    attributes:
      -
        name: netdev
        type: string
      -
        name: mtu
        type: u32
        checks:
          min: 1
          max: 65535
      -
        name: vf-list
        type: nested
        multi-attr: true
        nested-attributes: vf-list
```

C and kernel codegen

- Kernel
  - Generate uAPI
  - Generate the ops tables
  - Generate the policy tables
  - Generate ReST docs from the descriptions in the spec?
- User
  - Generate struct types
  - Generate parsers
  - Generate policy tables
  - Generate message info metadata for ext_ack reverse-parsing
- In C/C++ format enum / define names automatically
  - commands: $family_CMD_$cmd-name => DPLL_CMD_SOURCE
  - attributes: $family_A_$set-name_$attr-name => DPLL_A_SOURCE_ID

A word on error handling

- Use policy / metadata tables to reverse parse messages
- Netlink ext ack can carry
  - error messages (**strings**)
  - pointers to attributes (**attribute offset**)

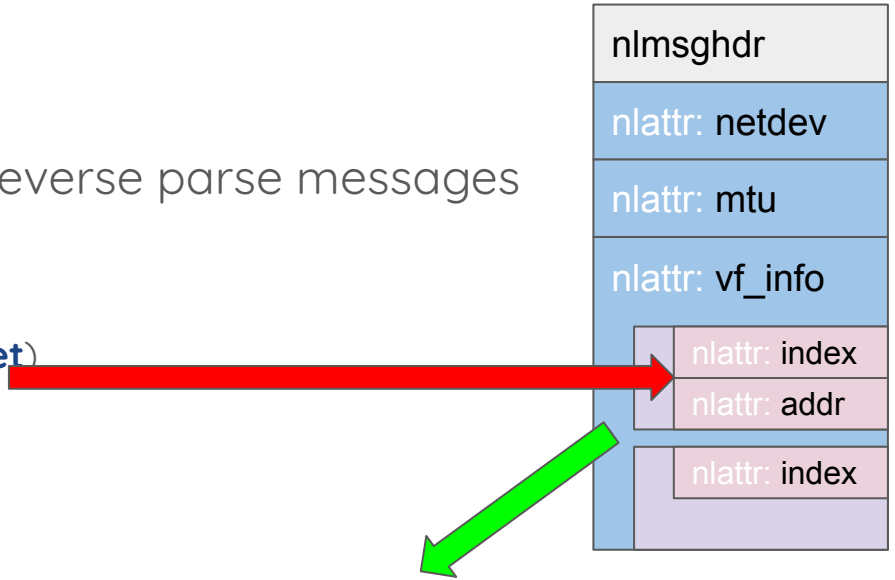
Example errors:

```
setlink: Invalid MAC address (invalid attribute .vf-list.mac-address)

setlink: missing attribute .vf-list.index
```

A word on error handling

- Use policy / metadata tables to reverse parse messages
- Netlink ext ack can carry
  - error messages (**strings**)
  - pointers to attributes (**attribute offset**)



Example errors:

```
setlink: Invalid MAC address (invalid attribute .vf-list.mac-address)

setlink: missing attribute .vf-list.index
```

Reality check

- Most existing families have quirks
- 4 schemas:
  - genetlink - for new families
  - genetlink-c - additional attributes controlling C names
  - genetlink-legacy - all sort of doodoo
  - netlink-raw - pre-Generic Netlink stuff (i.e. all of NETLINK_ROUTE 😢)

The first two are pretty much ready.

Initial targets: DPLL, FOU, devlink, ethtool

WIP tree: https://github.com/kuba-moo/ynl