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
- Linux 2.6.14/15 (2005) - Generic Netlink, nlattr etc. (Thomas Graf)
What netlink is supposed to be

- Semi-magic bits?
- C structure:
  - member alignment
  - info leaks
  - member presence
  - extensibility?

**ioctl**
- 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

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

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

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

- **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_UNSPEC,
    IFLA_ADDRESS,
    IFLA_BROADCAST,
    IFLA_IFNAME,
    IFLA_MTU,
    IFLA_LINK,
    ...
}
```

```
enum {
    IFLA_VF_UNSPEC,
    IFLA_VF_MAC,
    IFLA_VF_VLAN,
    ...
}
```
Attributes

<table>
<thead>
<tr>
<th>u16 type: IFLA_IFNAME</th>
<th>u16 length: 9</th>
</tr>
</thead>
<tbody>
<tr>
<td>data: (string) eth0\0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>u16 type: IFLA_VF_LIST</th>
<th>u16 length: 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>data:</td>
<td>u16 type: IFLA_VF</td>
</tr>
<tr>
<td></td>
<td>u16 length: 8</td>
</tr>
<tr>
<td></td>
<td>data: (u32) 1</td>
</tr>
<tr>
<td></td>
<td>u16 type: IFLA_VF_MAC</td>
</tr>
<tr>
<td></td>
<td>u16 length: 10</td>
</tr>
<tr>
<td></td>
<td>data: 00:11:22:33:44:55</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>u16 type: IFLA_MTU</th>
<th>u16 length: 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>data:</td>
<td>(u32) 6000</td>
</tr>
</tbody>
</table>
Binary format

Netlink

\textbf{cmd}: RTM_SETLINK
\textbf{netdev}: “eth0”
\textbf{mtu}: 6000
\textbf{vf_info}: [
  \textbf{index}: 1
  \textbf{mac-address}: 00:11:22:33:44:55
], [
  \textbf{index}: 2
  \textbf{mac-address}: 00:11:22:33:44:56
]
<table>
<thead>
<tr>
<th>nlmsghdr</th>
<th>u32 length</th>
</tr>
</thead>
<tbody>
<tr>
<td>nlattr: netdev</td>
<td>u16 type</td>
</tr>
<tr>
<td>nlattr: mtu</td>
<td>u32 seq</td>
</tr>
<tr>
<td>nlattr: vf_info</td>
<td>u32 pid</td>
</tr>
<tr>
<td>nlattr: index</td>
<td></td>
</tr>
<tr>
<td>nlattr: addr</td>
<td></td>
</tr>
<tr>
<td>nlattr: index</td>
<td></td>
</tr>
<tr>
<td>nlattr: addr</td>
<td></td>
</tr>
</tbody>
</table>
Basic exchanges - 1/3 : Do

- User
  - SETLINK(seq=123, flags=REQUEST|ACK)
  - ACK(seq=123)
  - ERROR(seq=123, errno=EINVAL)

- Kernel
Basic exchanges - 2/3 : Notify

User 1: register for notifications

User 2:
- SETLINK(seq=123, flags=REQUEST/ACK)
- ACK(seq=123)
- LINK CHANGED (#1)
- LINK CHANGED (#2)

Kernel:
- carrier event
Basic exchanges - 3/3 : Dump

GETLINK(seq=123, flags=REQUEST|DUMP)

LINKINFO(seq=123)

DONE(seq=123)
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)
attribute-sets:
  -
    name: link
    attributes:
      -
        name: netdev
        type: string
      -
        name: mtu
        type: u32
      -
        name: vf-list
        multi-attr: true
        nested-attributes:
          -
            name: vf-list
            attributes:
              -
                name: vf
                type: u32
              -
                name: mac-address
                type: binary
                length: 6

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

YAML Spec
Putting it together

```python
1 family = YNL("iflink.yaml")
2 family.setlink({
3     "netdev": "eth0",
4     "mtu": 6000,
5     "vf-list": [{
6         "vf": 1,
7         "mac-address": b'\x00\x11\x22\x33\x44\x55'
8     }, {
9         "vf": 2,
10        "mac-address": b'\x00\x11\x22\x33\x44\x56'
11     }]
12 )
```
YAML Spec

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

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

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

    attributes:
      -
        name: vf
type: u32
      -
        name: mac-address
type: binary
length: 6
YAML Spec

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

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

YAML Spec

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

operations:
  - name: getlink
    value: 4
    attribute-set: link
do:
  request:
    u16 type: IFLA_MTU
    u16 length: 8
    data: (u32) 6000
    ...  
  reply:
      netdev
      mtu
      vf-list
      ...  
  dump:
    reply:
      netdev
      mtu
      ...  

YAML Spec

(u32) 6000
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

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

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

operations:

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

- name: getlink
  attribute-set: link
  ...

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