Network Configuration

Nightmare on Elm Street

Olaf Kirch <okir@suse.com>

Agenda

- Why Change?
- Overview of Goals, Current Status
- Feedback, questions

Overview of Concepts

Why change?

- Quick, can you tell me how to...
  - ... disable IPv6 on a specific interface?
  - ... set up an interface for DHCPv4 and DHCPv6?
  - ... change the link speed on an Ethernet interface?
  - ... reconfigure a bonding device without bringing it down?
  - ... set up a bridge on a bond
  - ... change the firewall rules on your UMTS modem?
  - ... set up 802.1x authentication for your Ethernet NIC?
  - ... set up persistent names for your System z devices?

Why change, cont'd

- libvirt wants to set up interfaces at run-time
  - uses XML and a library called netcf
- openvswitch wants to mess with network setup as well
- We have a zoo of services already, more or less well-integrated
  - NetworkManager
  - ifplugd
  - dhcp4/dhcp6 client, IPv4 zeroconf
  - and probably a few more
- Marius is doing a great job maintaining the code *and* his sanity

It could be so simple

- After all, networking is pretty much a layered affair
- So why are some things hard to do?
  - Shell scripts weren't made for structured programming or handling structured data
  - “Set up and forget” worked well ten years ago
  - Lots of things have changed in the kernel (rtnetlink was really designed for a management daemon)
  - Incredible number of tunables (giving rise to lots of INCREDIBLY_LONG_VARIABLE_NAMES_NOBODY_WILL_EVER_REMEMBER)
- A lot of the sysconfig design is around system facilities, not around semantics

Goals

- Cope with increasingly complex configurations
  - Bridge over VLANs over bonded NICs
- Fix the device naming problem
- Model interface dependencies
- Needs small footprint (initrd use)
- Extensible

Nice-to-haves

- Networking is a layering exercise
  - if we get the layering right, adding new functionality becomes much simpler
- Having a uniform API for network interfaces
  - you can still use vconfig, brctl, ip(8) and all that
- Server/client architecture benefits
  - monitor and react to rtnetlink events
  - but retain as little state as possible (recover from daemon restart)
  - make reconfiguration incremental, as much as possible

There must be something we can use?

- NetworkManager: okay for desktop, but not really made for servers
- WICD, connman, ...: none of them seem to go into the right direction
- Sorry, but nothing we could use

Non-goals

- Replacing NetworkManager completely
- World domination (aka locking users into a specific tool set)

Enter: The Project with no Name

Wicked history

- Started as a hack week project for network monitoring
- Morphed into “try to do better than ifup”
- Original design was based on a REST interface
  - Worked, but ugly
- Second Iteration moved to a dbus interface
  - Much better, but steep learning curve
  - Great for layering: just use DBus “Interfaces”
- Tried to do the object management in C, and all methods in bash scripts
  - Nicely extensible, but... some things are better done in C

Some basic design decisions

- Use XML for configuration files
- Use DBus for communication
- Don't be afraid to re-implement low level utilities like vconfig, brctl as dbus services in C
- Heck, don't be afraid of re-implementing dhcpcd :)
- Support extensions via shell scripts etc
- Client should have no knowledge about specific device types

On DBus concepts

- DBus servers
  - publish a name on DBus, like com.suse.Wicked
- objects
  - identified by a path, eg /com/suse/Wicked/Interface/1
- interfaces
  - interfaces are a collection of methods
  - an object has one or more interfaces
- there is no notion of an object class in DBus
  - so I added that to my implementation
  - a class is a collection of interfaces
- to avoid confusion with network interfaces, I used the term “service” instead of “interface”

Wicked DBus Handling

- Using basic libdbus, which is really no-frills
  - Like RPC programming without XDR
- Add a somewhat higher level API, including
  - C object representing objects, the interfaces they export, their hierarchy, etc
  - Object classes for dbus objects
  - DBus signal handling
  - Marshal/unmarshal functions that operate on a common polymorphic data type
  - Support generic DBus interfaces like org.freedesktop.DBus.Properties and o.f.D.ObjectManager

Wicked NIH Symptoms

- Why doesn't wicked use...
  - libxml2, glib2, augeas, you name it?
- size matters if you want to put it in initrd
  - libxml2 is 1.4 MB
  - glib2 + dbus-glib binding is another 1MB
  - ... and you probably need many more knick-knacks like these if you want to maximize reuse of existing facilities

Object Model and General Mode of Operation

Wicked object model

- Loopback interface has ifindex 1, eth0 usually has ifindex 2, etc
  - /com/suse/Wicked/Interface/1 is the path for the loopback device.
  - pro: invariant against interface renames
  - con: no object without existing interface (chicken/egg issue)
- Associate an object class with a link-layer type
  - Loopback devices have a class of netif-loopback
  - Ethernet devices have a class of netif-ethernet
  - etc

Wicked object model, example

- Ethernet device; class netif-ethernet

DBus Interface	DBus methods
com.suse.Wicked.Ethernet	changeDevice
com.suse.Wicked.Firewall	firewallUp, firewallDown
com.suse.Wicked.Interface	linkUp, linkDown
...Addrconf.ipv4.static	requestLease, dropLease
...Addrconf.ipv6.static	requestLease, dropLease
...Addrconf.ipv4.dhcp	requestLease, dropLease
...Addrconf.ipv6.dhcp	requestLease, dropLease

Ethernet device configuration

```
<interface>
  <name>eth0</name>
  <ethernet>...  </ethernet>
  <link>...</link>
  <firewall> ... </firewall>
  <ipv4:static>
     <address>...</address>
     <route>...</route>
  </ipv4:static>
  <ipv4:dhcp/>
</interface>
```

Mapping XML to DBus Arguments

- In a nutshell, pretty schematic
  - An element containing a string, number, or any other scalar, is marshalled as the corresponding dbus type
  - An element containing a collection of other elements is encoded as a dbus dictionary with string keys
  - Arrays are also supported
- Actually, it's so schematic that it's automated
  - Major improvement over the parsing mess in the first version of wicked :-)
- Corollary: using XML attributes are not worth the hassle

If a VLAN is the egg, where's the chicken?

- To recap, an object is identified by its ifindex
  - IOW, we cannot have objects for interfaces that don't exist yet
- Solution: factory interfaces
  - There's one factory interface for every virtual device type:
    - com.suse.Wicked.VLAN.Factory
    - com.suse.Wicked.Bridge.Factory
    - com.suse.Wicked.OpenVPN.Factory
  - Factory interfaces are provided by /com/suse/Wicked/Interface

The Chicken Coop

- One chicken for every type of egg:

DBus Interface	DBus methods
com.suse.Wicked.InterfaceList	...
com.suse.Wicked.VLAN.Factory	newDevice
com.suse.Wicked.Bridge.Factory	newDevice
com.suse.Wicked.Bond.Factory	newDevice
com.suse.Wicked.OpenVPN.Factory	newDevice

The VLAN Egg

- VLAN.Factory.newDevice() takes two arguments
  - The requested interface name (potentially empty)
  - A dict corresponding to the <vlan> element of the config file
    - containing the name of the underlying eth device, and a VLAN tag
- The factory function creates the VLAN device, and returns the object path to the caller
- Caller then proceeds as usual

Other example: bridges

- Bridge.Factory.newDevice() takes two arguments
  - The requested interface name (potentially empty)
  - A dict corresponding to the <bridge> element of the config file
- The factory function creates a plain bridge device, ignoring the dict
- The caller then invokes Bridge.changeDevice() with the same <bridge> data, which now configures the bridge ports etc
- Rest of the story: as above

Avoiding the Naming Mess

What's the Problem, anyway

- Customers want a unique, persistent handle that's easy to use
- Current solution: use udev + biosdevname + ...
- Using *naming services* to identify a device
  - Identify an ethernet device by its MAC address
  - Identify it by some BIOS information, à la biosdevname
  - Identify it by some obscure System z specific naming convention

The wicked way of addressing this

- In an <interface> element, you can reference devices by their ifname
- Alternatively, you can use <identify>
  - This uses “naming services” configured in the daemon
  - Extensible (shared libraries loaded via config file)
- Use this anywhere in the config file where you refer to a device
  - Also works for bridge ports, bond slaves, etc
- Currenly supports interface alias, Ethernet MAC
  - More to be added (biosdevname, System z, PCI hotplug)

Identify an eth device by MAC address

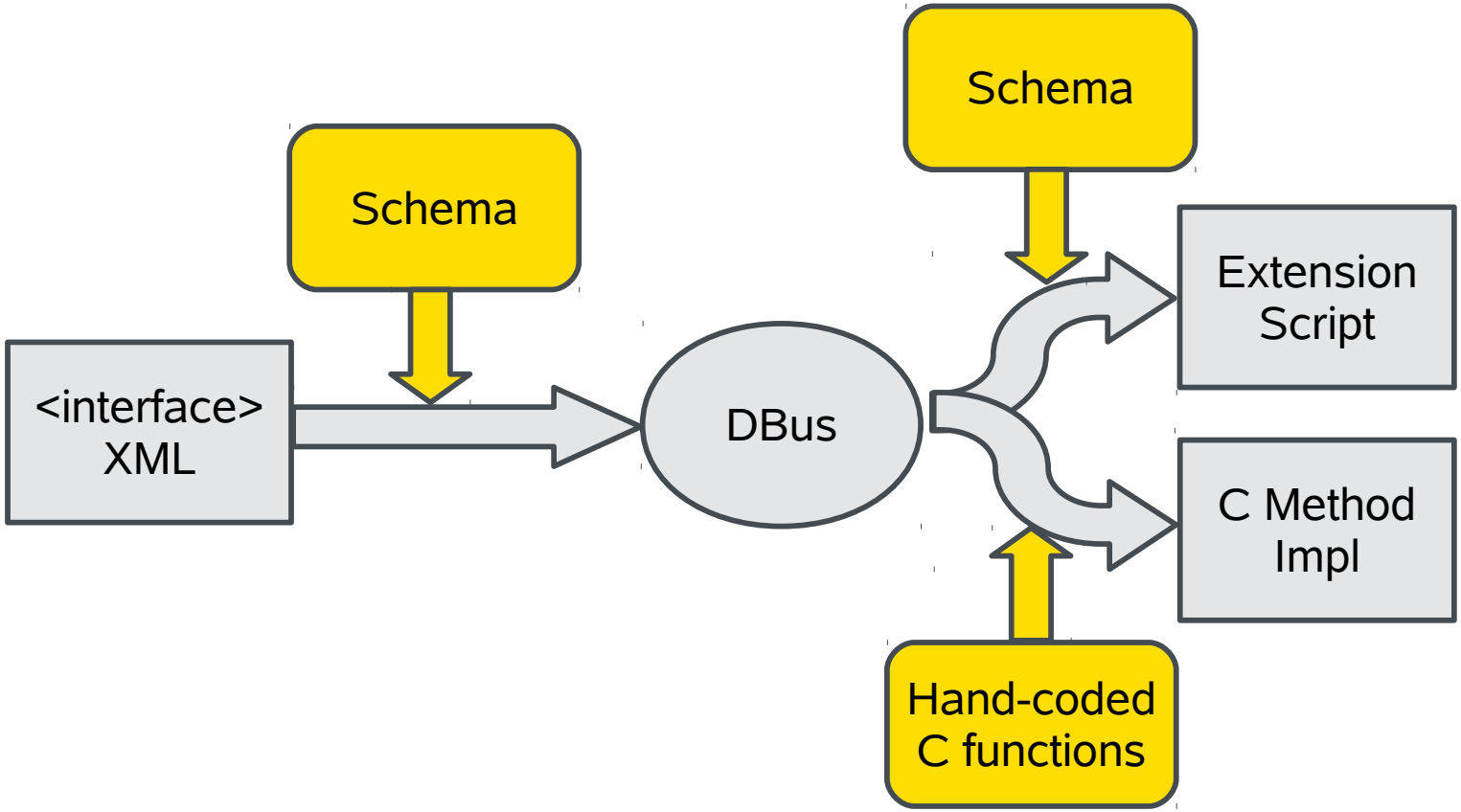
```
<interface>
  <identify>
    <ethernet:permanent-address>
      00:24:E8:CD:34:9B
    </ethernet:permanent-address>
  </identify>
  ...
</interface>
```

Using Schema to Define Behavior

Introduction to Schema

- No, it's not the standard XML schema
- The point is to automate marshalling of XML data to DBus and vice versa
- Nice side effects
  - Can define symbolic names for enums, bit fields and such
  - Can validate user supplied configuration for correctness
  - Can use schema metadata information to make behavioral choices
    - For instance, “before you bring up the link on this VLAN device, its underlying ethernet device must be up”

Marshal/Unmarshal



Current Status and Next Steps

Current Status

- Available from gitorious
- Ethernet, VLAN, bridge, bond: fairly complete
- wireless (using wpa-supplicant): 80%
- openvpn: worksforme (butmaybenotforyou)
- umts: preliminary support
- dhcp4, IPv4 zeroconf, static addressing
- dhcp6 being done by Marius
- autoconf support thanks to Marius
- ibft: untested
- Sort-of-current documentation

Next Steps

- Work with the openSUSE community
- Long list of things to do
  - Improve documentation (terribly outdated)
  - Put into OBS
  - support ibft, dhcp6, 802.1x, ...
  - firewalling support
  - systemd integration
  - conversion of sysconfig files -> wicked
- Support all the nitty gritty stuff around it, like xen vifs, libvirt, openvswitch, yadda yadda

Feedback welcome

We look forward to your questions and feedback!

Thank you.

Corporate Headquarters
Maxfeldstrasse 5
90409 Nuremberg
Germany

+49 911 740 53 0 (Worldwide)
www.suse.com

Join us on:
www.opensuse.org