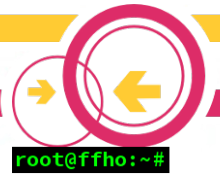


Building your own SDN

with Debian Linux, Salt Stack and Python

FrOSCon 13

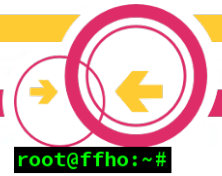
Maximilian Wilhelm
<max@sdn.clinic>
@BarbarossaTM

Who am I?

- Maximilian Wilhelm
  - @BarbarossaTM
- Senior Infrastructure Architect, Uni Paderborn
- Infrastructure Archmage, Freifunk Hochstift
- Fanboy of
  - (Debian) Linux
  - ifupdown2
- Networker
- OpenSource Hacker

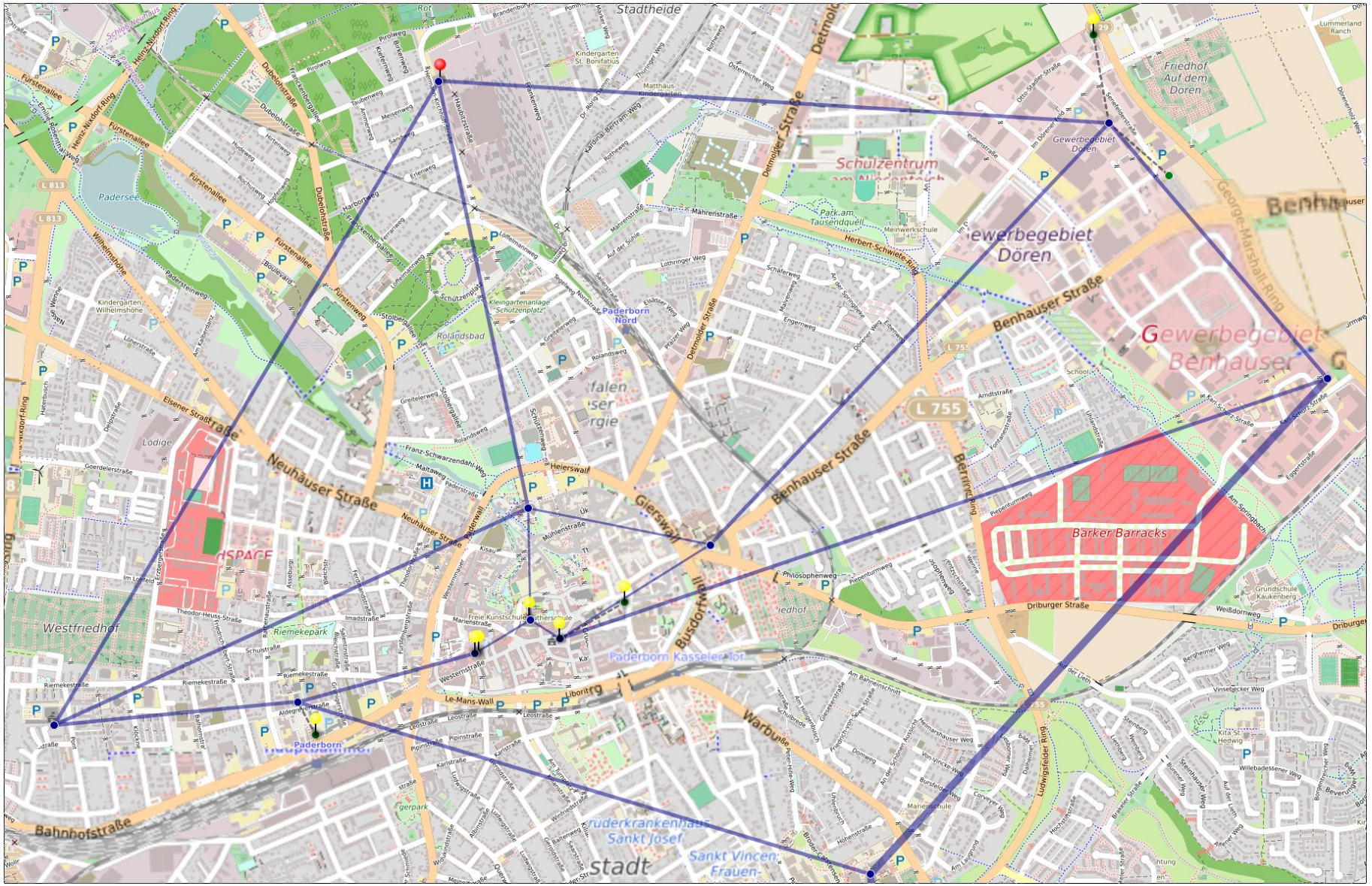
Agenda

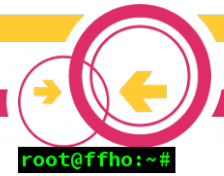
- Contemporary Linux Networking
  - Batteries included
  - ifupdown2
  - Salt Stack
- DIY-SDN
  - Python modules
- Putting it all together



Why bother building an SDN?

FFHO WiFi backbone (planned)

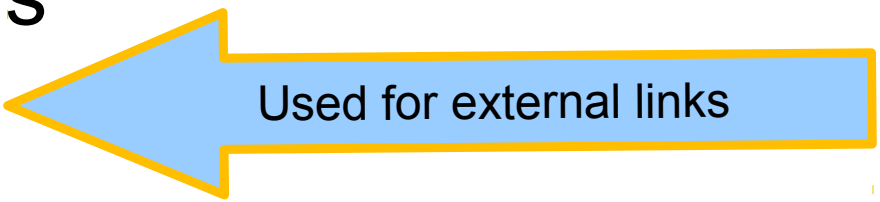


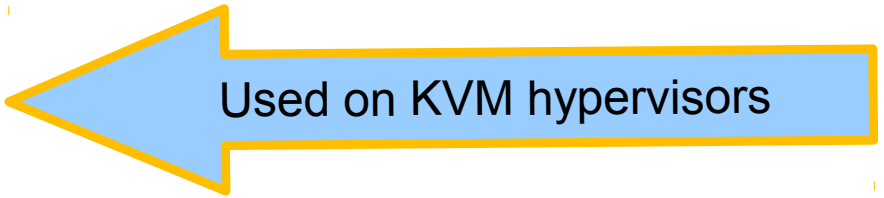


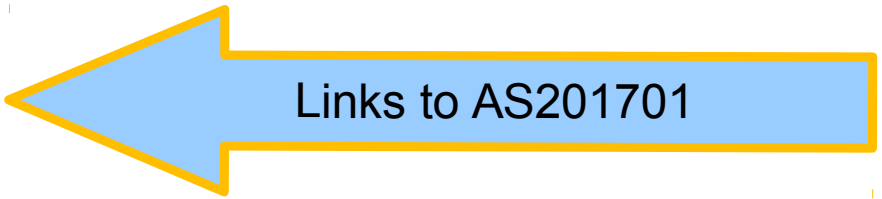
OK, now what?!

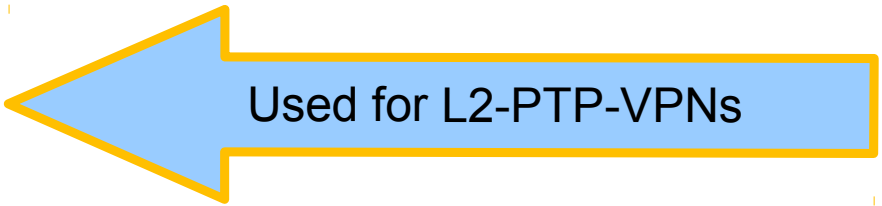
What's in our cards?

- Latest networking additions
  - VRFs
  - MPLS forwarding plane
  - VLAN-aware-bridges
- Tunneling
  - GRE
  - IPsec
  - OpenVPN
  - Wireguard









Network interface configuration

- Classic ifupdown not easily automated
- Generating /etc/network/interfaces simple
- How to reload?
  - »service networking restart« disruptive
  - No tool for “reload” present
  - Isn't trivial to build
- ➔ CumulusNetworks Ifupdown2
  - Rewrite of ifupdown in Python
  - https://github.com/CumulusNetworks/ifupdown2

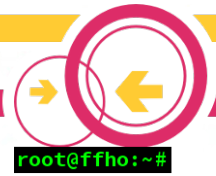
ifupdown2

- No full feature parity with ifupdown
- Shipped with batteries included
  - dependency resolution
  - **`ifreload`**
  - VRFs
  - VXLAN
  - VLAN-aware bridges
  - ppp

ifupdown2 Patches

- Easy to extend, thanks to Python
- Upstream open for ideas (Hi Julien & Roopa)
- Added support for
  - B.A.T.M.A.N. interfaces
  - Tunnel (GRE, SIT, IPIP, GRETAP)
  - Condoning bridge interfaces for configuration
  - Setting phys-dev for VXLAN
- Open Pull-Requests for
  - Setting vEth peer name

VXLAN / VTEPs with ifupdown2

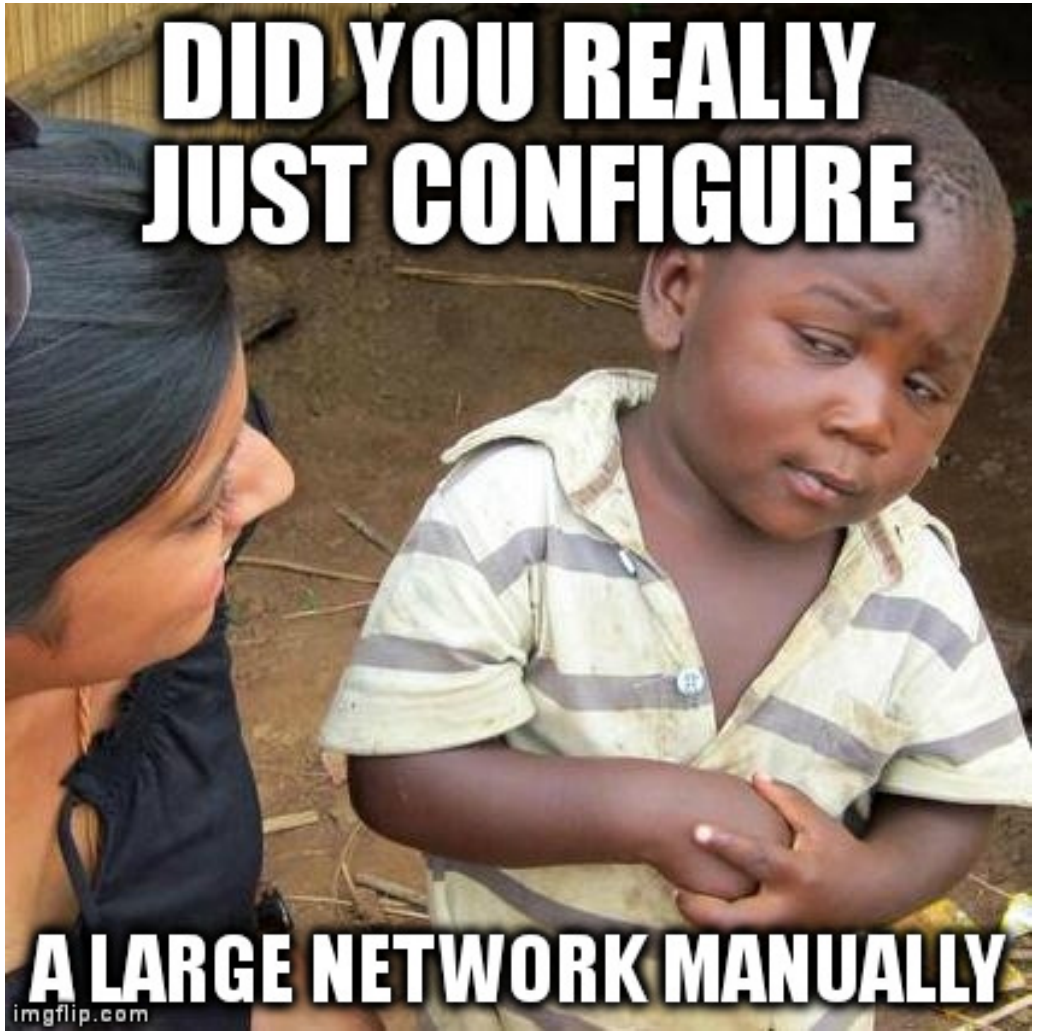



```
# vx_v2001_padcty

auto vx_v2001_padcty
iface vx_v2001_padcty
        vxlan-id          1310977
        vxlan-physdev     vlan2001
        vxlan-svcnodeip 225.20.1.1
        #
        hwaddress f2:00:c1:01:20:01
        mtu 1560
```

About that routing thing

- Protocols
  - OSPF
  - IS-IS
  - BGP
- Implentations:
  - Bird
  - ~~Quagga~~FRR
  - ExaBGP
  - Bio Routing (Written in go :-))

Automation


DID YOU REALLY
JUST CONFIGURE
A LARGE NETWORK MANUALLY
imgflip.com

Salt Stack

- Continuous Management
  - (De)install packages
  - Generate and manage config files
  - Manage system services/units
  - Configure the network
  - Distribute Host Certificates
  - ...
- Written in Python
- Extensible by own Python modules

States

- Represent state $something should have
- Usually YAML format
- Bunch of definitions of..
  - Packages to (de)install
  - Services to (de)activate
  - File contents
  - Users
  - …
- Allows definition of dependencies

State example (bird excerpt)

```
/etc/bird/bird.d/IGP.conf:
  file.managed:
    - source: salt://bird/IGP.conf
    - template: jinja
      proto: v4
    - watch_in:
      - cmd: bird-configure
    - require:
      - file: /etc/bird/bird.d
    - require_in:
      - service: bird
```

https://github.com/FreifunkHochstift/ffho-salt-public/blob/master/bird/init.sls

Pillar

- Structured key value store
- Usually YAML format
  - Eval Jinja2 → Read YAML
- Data can be read from templates
  - As data structures. Yay.
- Predestined for
  - Keys
  - Host specific configuration
  - Any other configuration data

Pillar Example (TE excerpt)

```
te:
  community_map:
    cr01.in.ffho.net:
      ffrl:
        - EXPORT_ONLY_AT_CR01
  [...]

  prefixes:
#   <prefix/mask>:
#     desc: "my magic prefix"
#     communities:
#       - COMMUNITY_ONE
#       - "(12345, 4711)"
#     nodes:
#       - node01.in.ffho.net
```

```
2a03:2260:2342::/52:
  desc: "Mesh Prefixes"
  communities:
    - EXPORT_RESTRICT
    - EXPORT_ONLY_AT_CR03
  nodes:
    - cr03.in.ffho.net

10.132.32.0/22:
  desc: "gw03 Paderborn (Kernstadt
  communities:
    - GATEWAY_TE_ROUTE
  nodes:
    - gw03.in.ffho.net
  dhcp_range: "10.132.32.20 10.132
```

Templating - Jinja2

- Default language: Jinja2
  - Yay, Python again – kind of
- Usable within file templates and states

```
bird-pkg:
  pkg.installed:
    - name: bird
{% if grains.oscodename in ['jessie', 'wheezy'] %}
    - require:
      - pkgrepo: bird-repo
{% endif %}
```

Jinja2 state example

```
  # Generate config file for every client known to pillar
  {% for node_id, node_config in salt['pillar.get']('nodes', {}).items () %}
    {% if node_config.get ('icinga2', "") != 'ignore' %}
/etc/icinga2/ffho-conf.d/hosts/generated/{{ node_id }}.conf:
  file.managed:
    - source: salt://icinga2/host.conf.tmpl
    - template: jinja
    - context:
      node_id: {{ node_id }}
      node_config: {{ node_config }}
    - require:
      - file: Create /etc/icinga2/ffho-conf.d/hosts/generated/
    - require_in:
      - file: Cleanup /etc/icinga2/ffho-conf.d/hosts/generated/
    - watch_in:
      - service: icinga2
    {% endif %}
  {% endfor %}
```

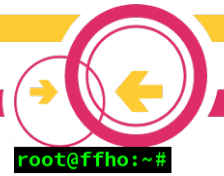
Modules

- No regex support in Jinja2 though. D'oh.
- ➔ 1. highly complex extension module:

```
# ffho.py

import re

def re_replace (pattern, replacement, string):
    return re.sub (pattern, replacement, string)

def re_search (pattern, string, flags = 0):
    return re.search (pattern, string, flags)
```

Hello regular expressions

```
# /etc/bird/bird.d/l3-access.conf

protocol direct l3_access {

{%- for iface in salt['pillar.get']('nodes:' ~
grains['id'] ~ ':ifaces') %}

  {%- if salt['ffho.re_search']('^vlan3\d\d$',
iface) %}

        interface "{{ iface }}";

  {%- endif %}

{%- endfor %}

}
```



Jinja or real Python?

- Templating is cool
- Complex code within Jinja2 is not
- Why not move this to modules?
  - Use full Python potential
  - Structure code into nice functions
  - Functions callable from Jinja2
  - Profit

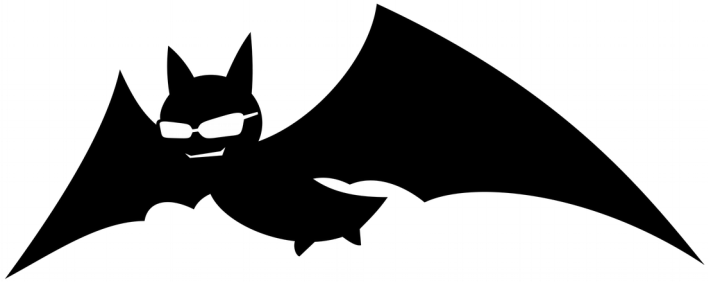
The SDN part

#SDN

“Freifunk Hochstift Superior Saltstack Automated Backbone des Todes of Doom oVXLAN”
-- hexa

Disclaimer: Font on special request of AbraXXL

SDN ingredients



ifupdown2

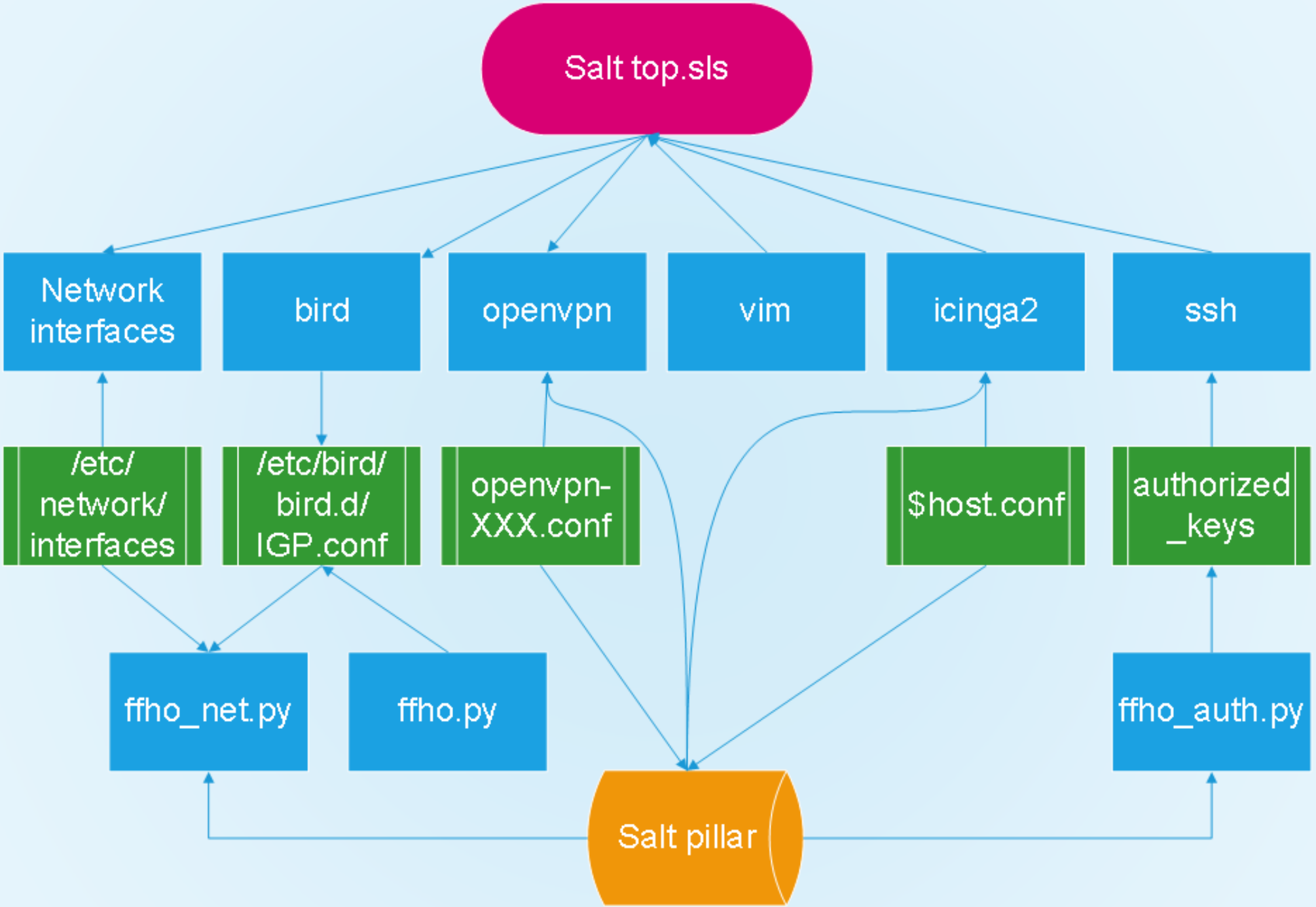
OSPF

RUN BGP

ffho_net.py

- Generate / compute
  - Interfaces config (for /etc/network/interfaces)
  - Bird OSPF configuration
  - Bird BGP configuration
    - iBGP / eBGP peerings / Traffic Engineering routes
  - B.A.T.M.A.N. + VXLAN overlay ifaces
    - Uniquie MAC addresses for B.A.T.M.A.N. ifaces
  - DNS entries from interface configuration

`https://github.com/FreifunkHochstift/ffho-salt-public/blob/master/_modules/ffho_net.py`


Salt top.sls
Network interfaces
bird
openvpn
vim
icinga2
ssh
/etc/ network/ interfaces
/etc/bird/ bird.d/ IGP.conf
openvpn-XXX.conf
$host.conf
authorized_keys
ffho_net.py
ffho.py
ffho_auth.py
Salt pillar

Pillar Example (node config)

```
bbr-vega.in.ffho.net:
   id: 198
   sysLocation: Vega
   roles:
      - router
      - batman
      - bbr
   sites:
      - pad-cty
```

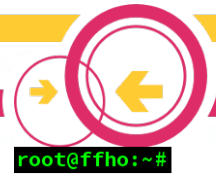
Source for Loopback-IP

Bird config (OSPF + iBGP)

Generate Batman interfaces

Batman instances

Pillar Example (node config) contd.



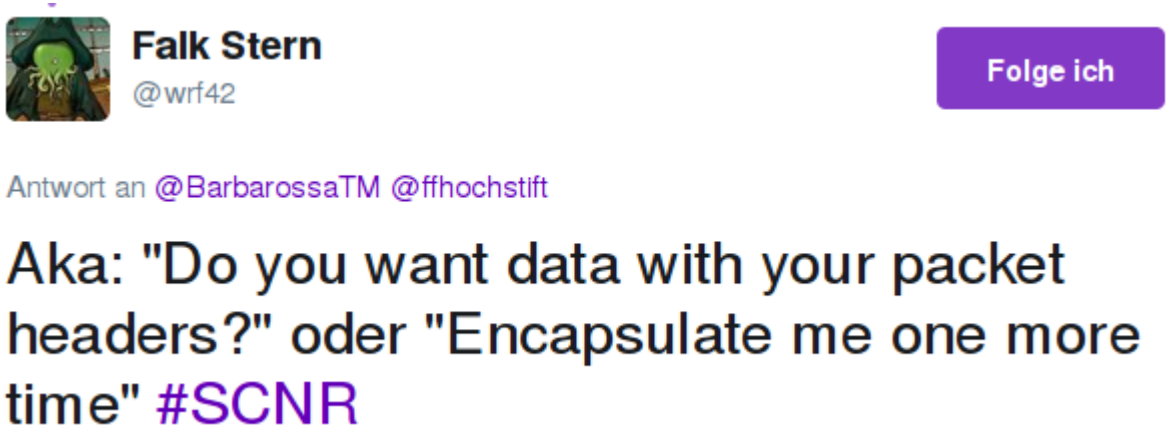

```
ifaces:
    bond0:
      bond-slaves: "eth0 eth1"

    vlan1002:
      desc: "<-> gw04"
      vlan-raw-device: bond0
      prefixes:
        - 10.132.253.58/31
        - 2a03:2260:2342:fe1 :1/126
      batman_connect_sites: pad-cty
[...]
```

Source for /etc/network/interfaces

Generate VXLAN overlay

IPoBATMANoVXLANoIPoVLANoRF

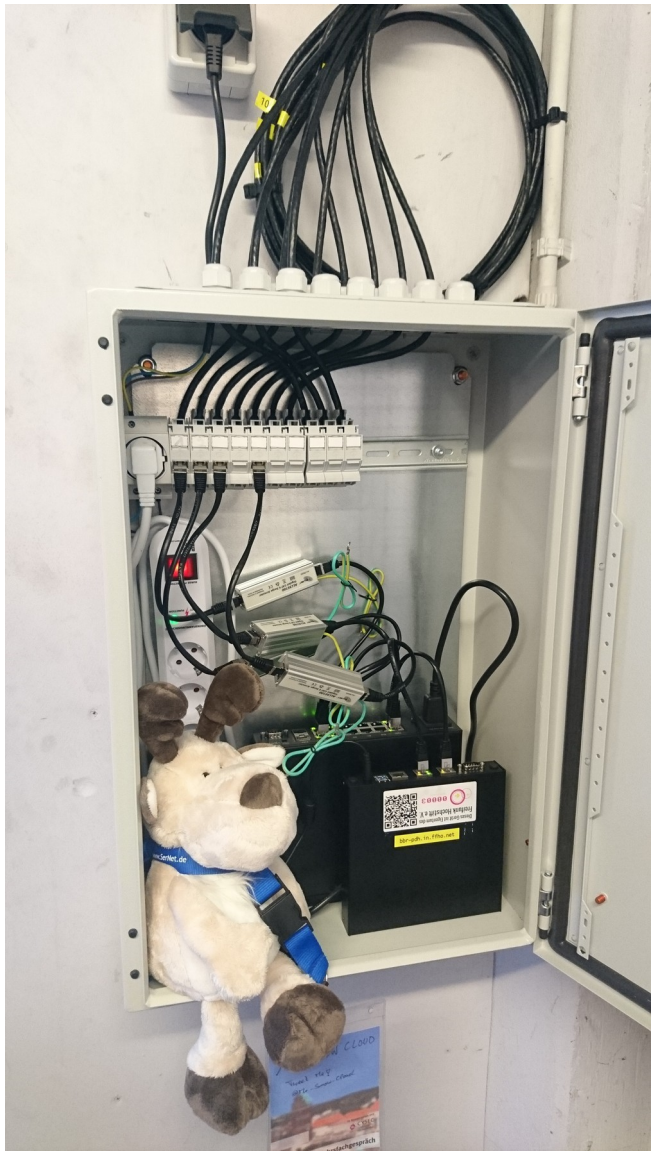
- Wait, what?



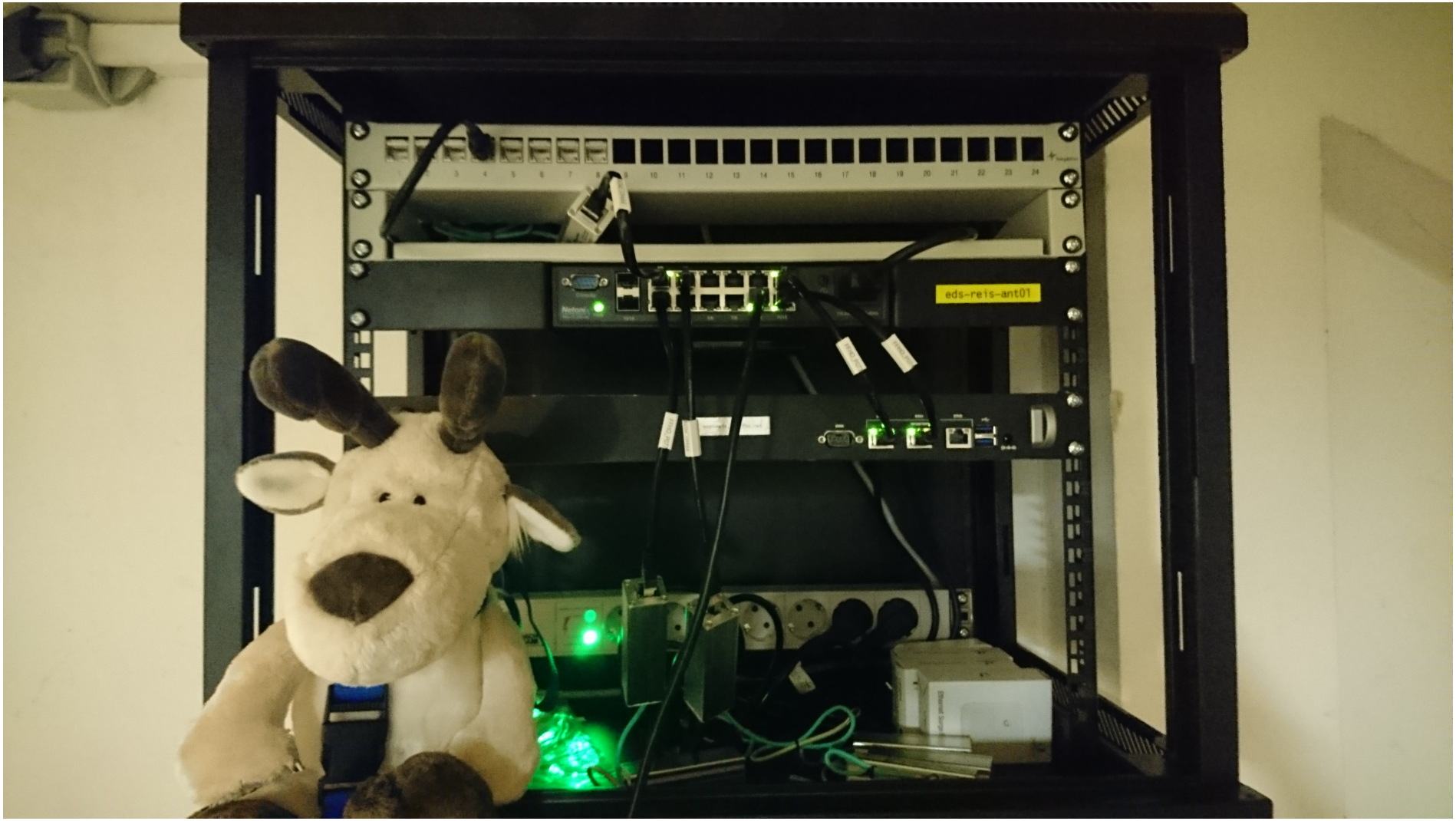
Hardware

- Zoo of sponsored Hardware
  - Servers, Switches, WiFi, ..
- ➔ Unify hardware pool
  - PCengines APU2
  - Netonix WISP Switches
  - Ubiquiti Networks
    - PowerBeam
    - LiteBeam
    - AC Mesh Pro

Before / After (PaderHalle)



We can do 19” as well



What's next?

- Migrate IPAM from RackTables to NetBox
  - Netbox as a nifty API
  - Query nodes and interfaces life from NetBox
- Finish DNS anycast setup
- Activate BGP ADD-PATH for iBGP (RFC7911)
- Achieve world domination

Interfaces from Netbox

```
{% set ifaces =
salt['grubenlampe.get_tunnel_interfaces'](node_id) %}

{% for interface in ifaces %}
auto {{ interface.name }}
iface {{ interface.name }} inet tunnel
        mode gre
        mtu 1400
        ttl 64
        local {{ interface.local }}
        endpoint {{ interface.endpoint }}
        #
  {%- for prefix in interface.ips  %}
        address {{ prefix }}
  {%- endfor %}

{% endfor %}
```

BGP sessions based on netbox

```
{%- set tunnels =
salt['grubenlampe.get_tunnel_interfaces'](...) %}

{%- for tunnel in tunnels %}
  {%- set peer_ip = tunnel.peer_ips[0].split ('/')[0] %}
  {%- set peer_ip_str = salt['ffho.re_replace']('[.:]',
'_', peer_ip) %}

protocol bgp
community_as{{ tunnel.peer_as }}_{{ peer_ip_str }} from
ebgp_community {
        neighbor {{ peer_ip }} as {{ tunnel.peer_as }};

        import filter community_{{ tunnel.tenant }}_in;
        import limit 100;

        multihop;
}

{%- endfor %}
```

Freifunkromantik

More Freifunkromantik

Further Reading

- Salt-Orchestrated Software Defined (Freifunk) Network (german)
  - https://www.slideshare.net/BarbarossaTM/software-defined-freifunk-backbones-78288014
  - https://blog.sdn.clinic/category/freifunk/ffho/

Questions? Remarks?

Thank you :-)

Maximilian Wilhelm
<max@elitepeer.de>
@BarbarossaTM