neutron-lan

SDN study environement @ home

github.com/alexanderplatz1999

Last update: April 23th, 2014

Background and Motivation

- My belief
  - SDN = software defines network
- Too many SDN definitions
  - I have been confused a lot.
  - OpenFlow, OVSDB, Netconf, BGP extensions such as FlowSpec...
  - The latest one: OpFlex (DevOps-like)
- What’s the real SDN?
  - Let’s develop SDN by myself and examine every definition.
- But, wait! I need a SDN study environment at home.
  - I am a poor guy, so I cannot buy expensive SDN-capable switches from Cisco, Juniper...

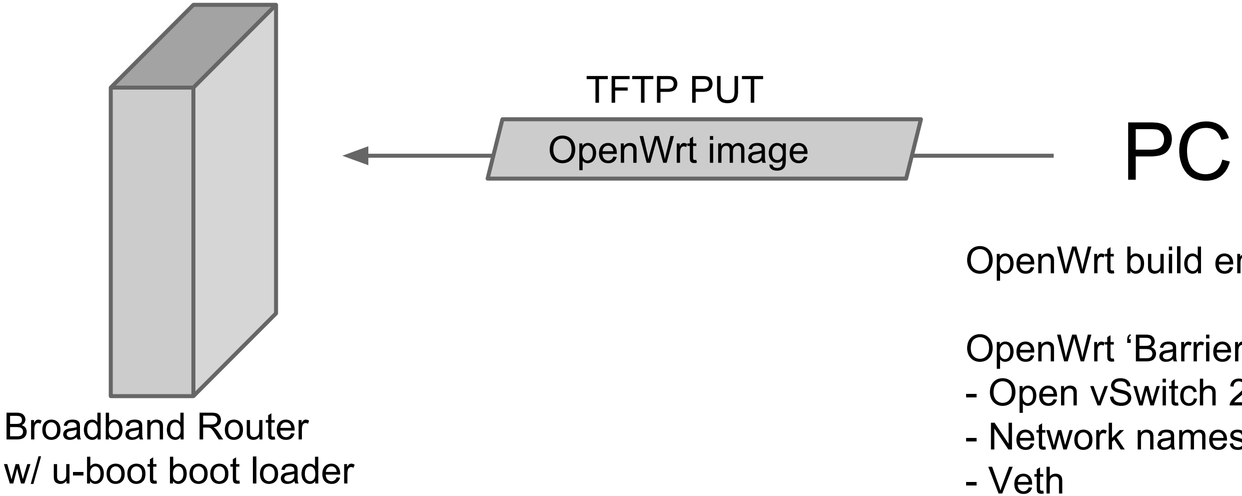
Strategy

- My budget is less than $200.
- Switches/routers I purchased in Akihabara, Tokyo
  - Three $40 broadband routers and one $40 Raspberry Pi
- And I develop all the SDN software from scratch
  - But reuse existing networking software as much as possible, such as Open vSwitch
- Base knowledge/skills
  - SDN in the past: SIP and IP-PBX
  - OpenFlow, OpenStack neutron and SaltStack
  - Java and Python
  - HTML5 and CSS (a little)
- Let’s develop neutron-like SDN for my home network ⇒ let’s call it ‘neutron-lan’

Project ‘neutron-lan’ characteristics

- Cheap routers as ‘Baremetal Switch’
  - OpenWrt, Raspberry Pi
  - u-boot for installing new firmware
- Home-made DevOps tool ‘NLAN’ from scratch
  - 100% Python implementation
  - YAML-based state rendering
  - Model-driven service abstraction
- VXLAN-based edge-overlay for network virtualization
- LXC for Network Functions Virtualization
- Open vSwitch as a programmable switch
- OVSDB as a general-purpose config database

Cheap routers as ‘Baremetal Switch’



OpenWrt build environment

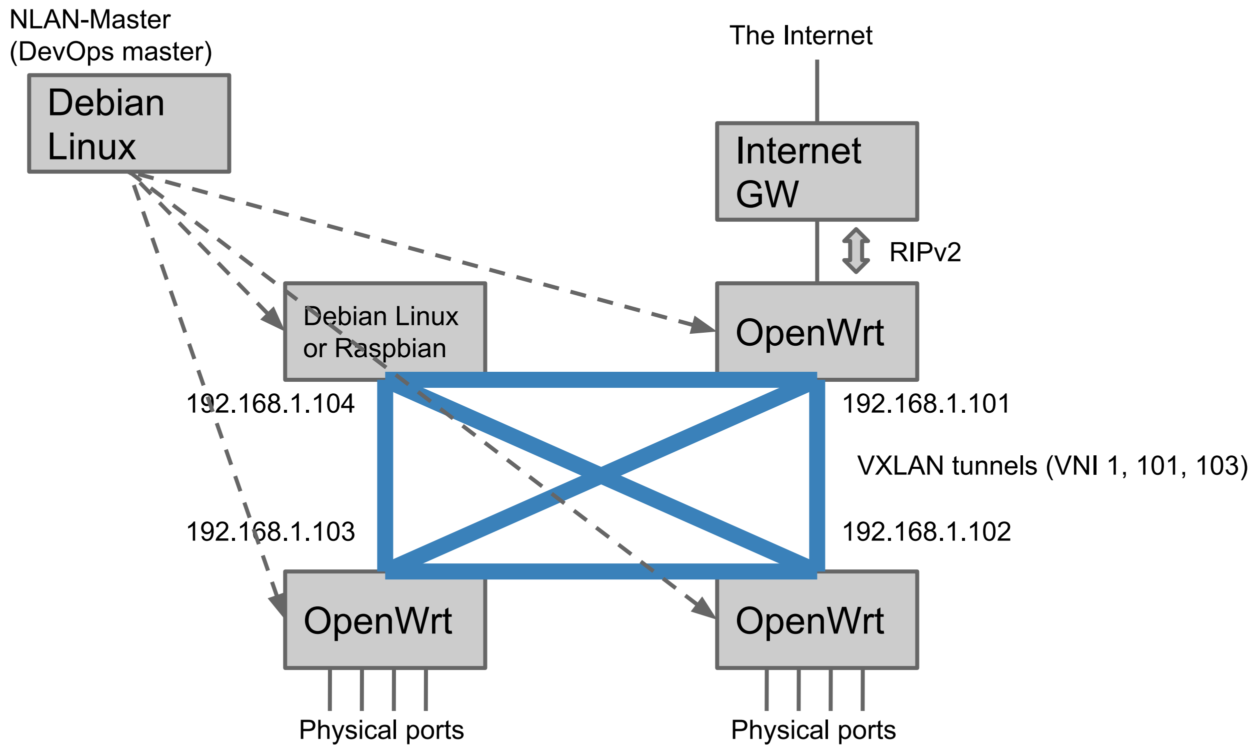
OpenWrt ‘BarrierBreaker’ image w/
- Open vSwitch 2.0.0
- Network namespaces
- Veth

Test bed (cont'd)

OpenWrt routers
(and Home Gateway)

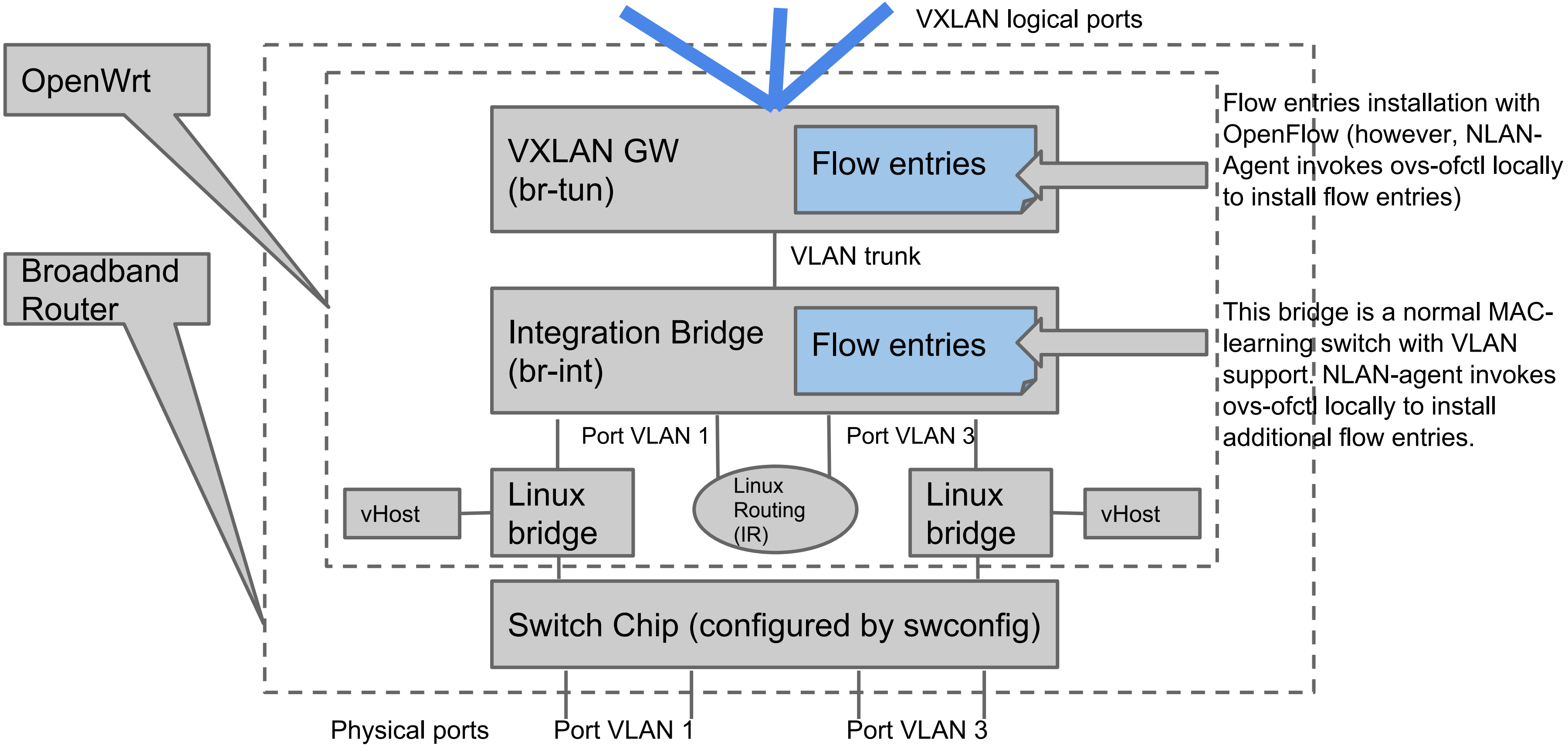
Raspberry Pi

Test bed

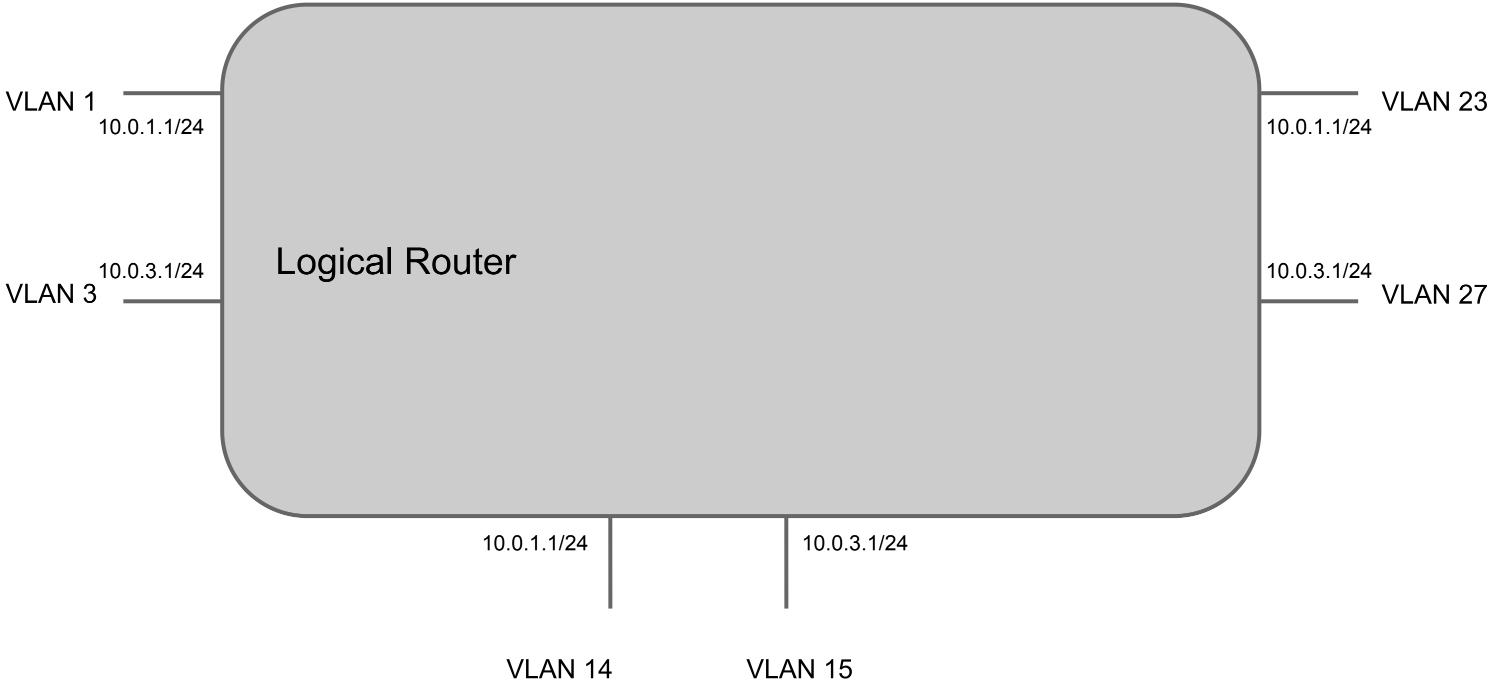


NLAN-Master
(DevOps master)
Debian Linux
The Internet
Internet GW
RIPv2
Debian Linux or Raspbian
OpenWrt
192.168.1.104
192.168.1.101
VXLAN tunnels (VNI 1, 101, 103)
192.168.1.103
192.168.1.102
OpenWrt
OpenWrt
Physical ports
Physical ports

OpenStack-neutron-like bridge configuration



Distributed Virtual Router (Logical view)



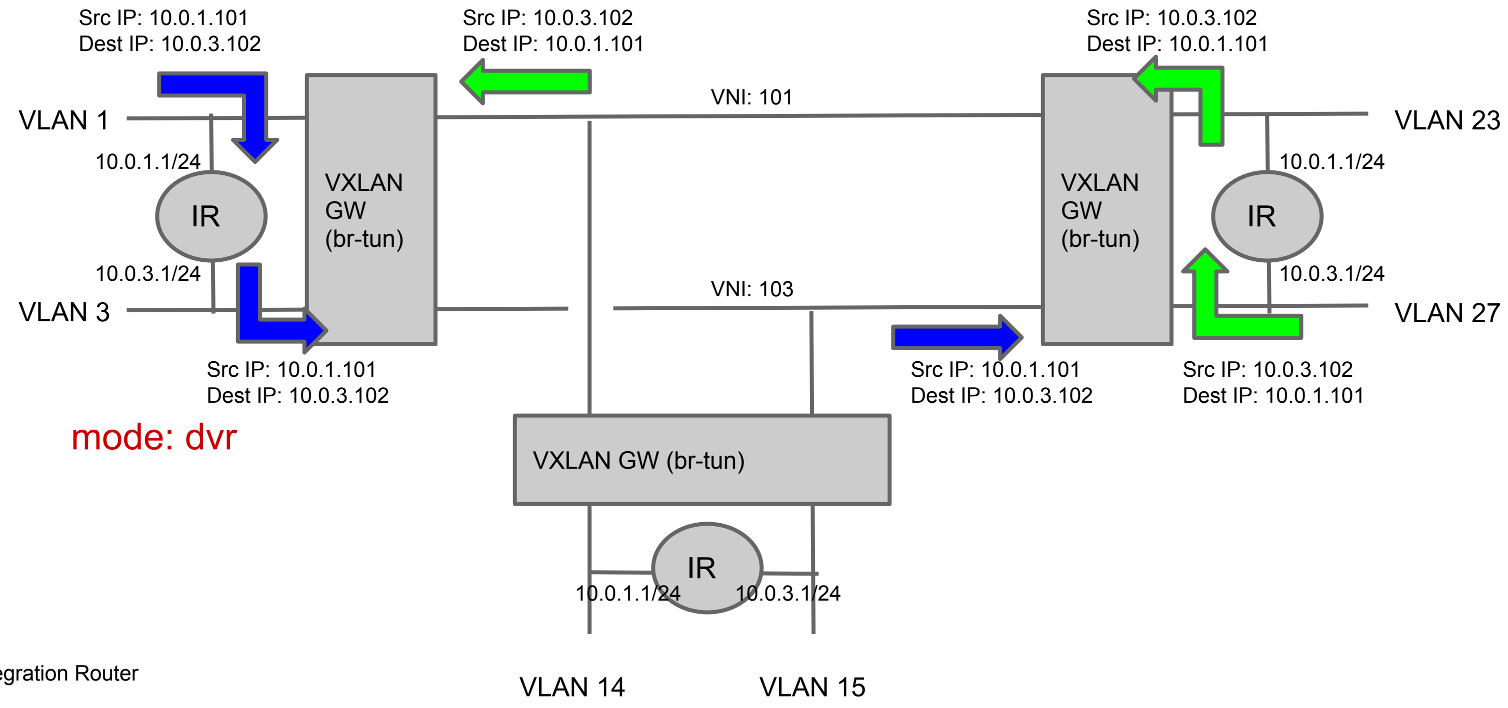
Virtual network topologies

NLAN node operation mode	Virtual Network Topology
dvr	Distributed Virtual Router
hub	Hub & Spoke
spoke	Hub & Spoke
spoke_dvr	Mixture of DVR and Hub & Spoke

NLAN state in YAML

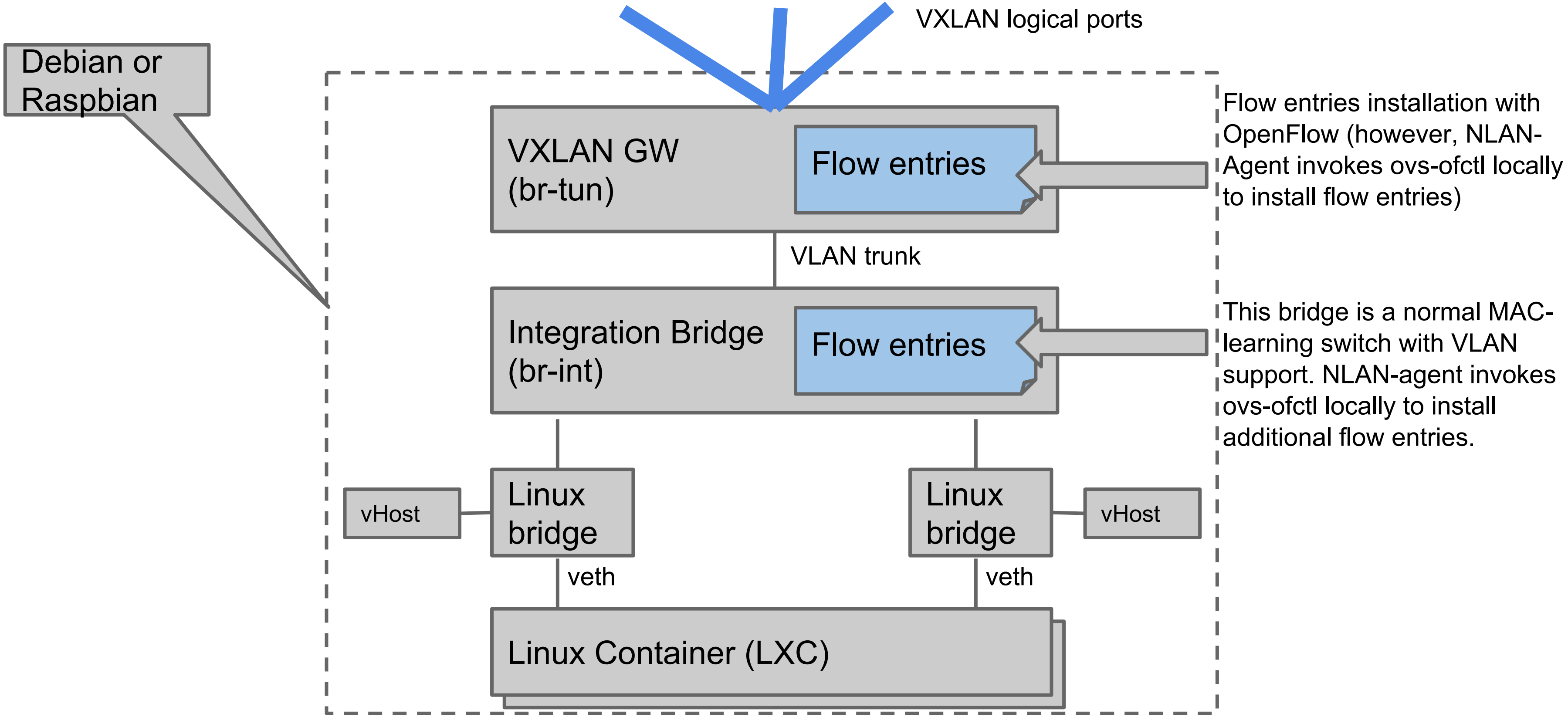
```
subnets:
  - vid: 1
  vni: 1001
  ip_dvr: '10.0.1.1/24'
  mode: hub          mode can be ‘dvr’, ‘hub’, ‘spoke’ or ‘spoke_dvr’
  ip_vhost: '10.0.1.101/24'
  ports:
          - eth0.1
  peers: <peers>
```

Distributed Virtual Switch and Distributed Virtual Router

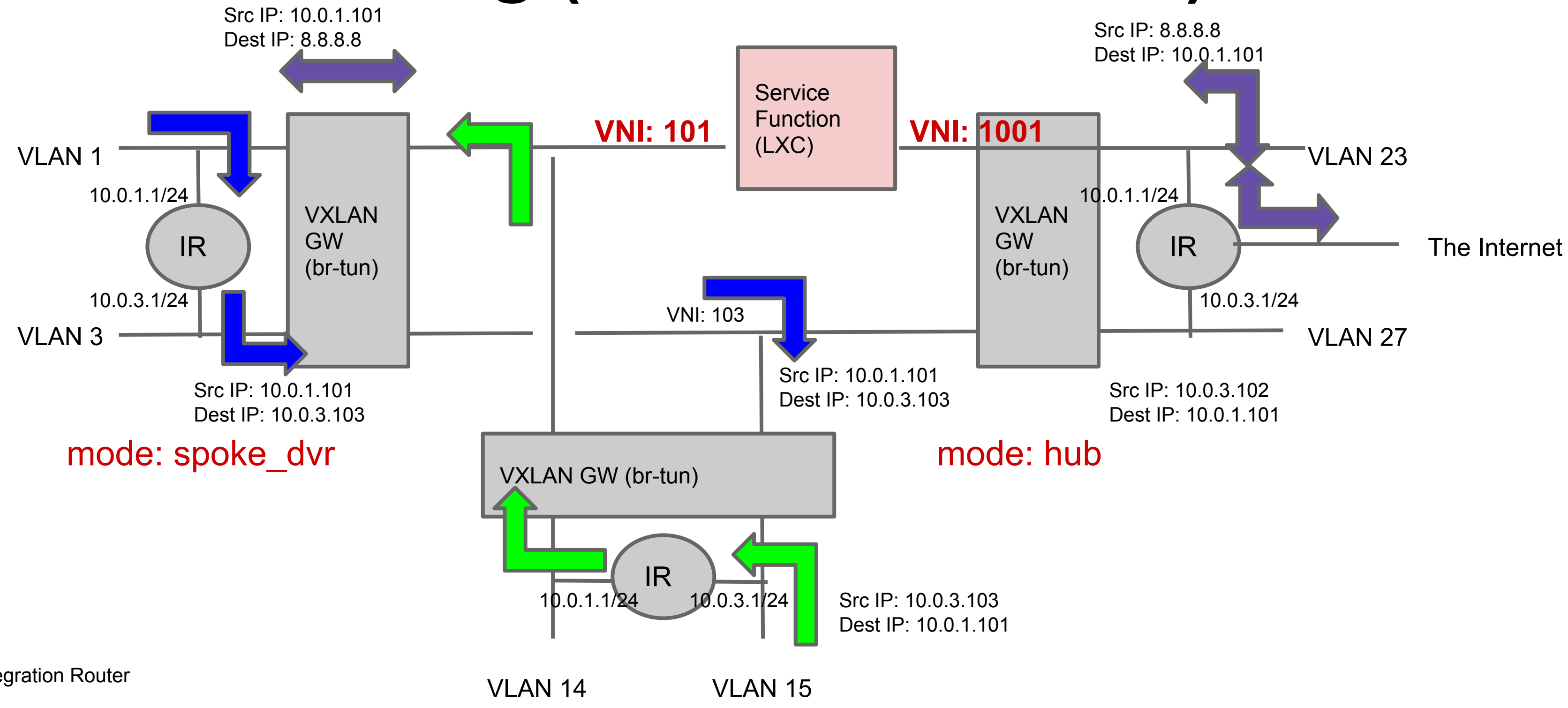


IR: Integration Router

Service Function in Linux Container

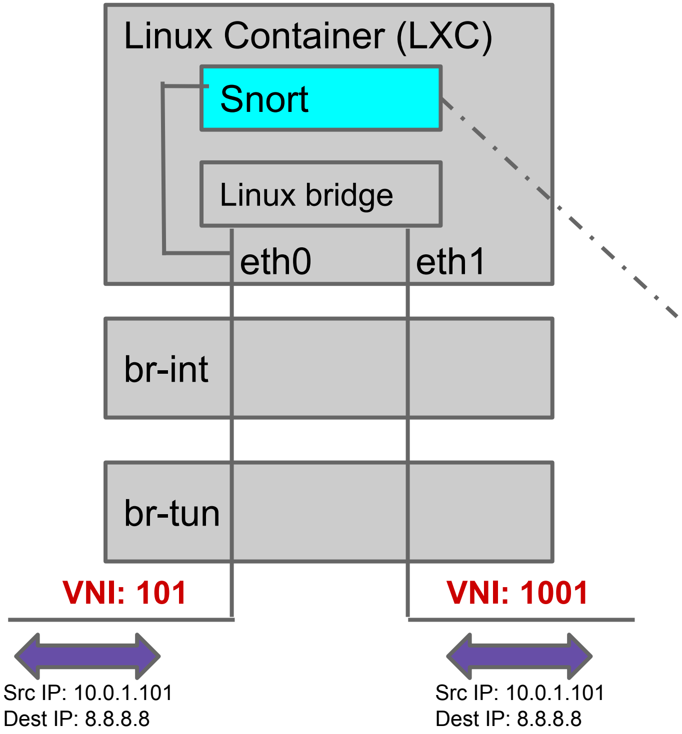


Service Chaining (Service Insertion)



IR: Integration Router

Example: Snort (in IPS mode) as Service Function



04/13-23:16:32.859385 10.0.1.102 -> 8.8.8.8
ICMP TTL:64 TOS:0x0 ID:11745 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:49496 Seq:25 ECHO
=+

04/13-23:16:32.861970 8.8.8.8 -> 10.0.1.102
ICMP TTL:63 TOS:0x0 ID:38077 IpLen:20 DgmLen:84
Type:0 Code:0 ID:49496 Seq:25 ECHO REPLY
=+

04/13-23:16:33.861151 10.0.1.102 -> 8.8.8.8
ICMP TTL:64 TOS:0x0 ID:11746 IpLen:20 DgmLen:84 DF
Type:8 Code:0 ID:49496 Seq:26 ECHO
=+

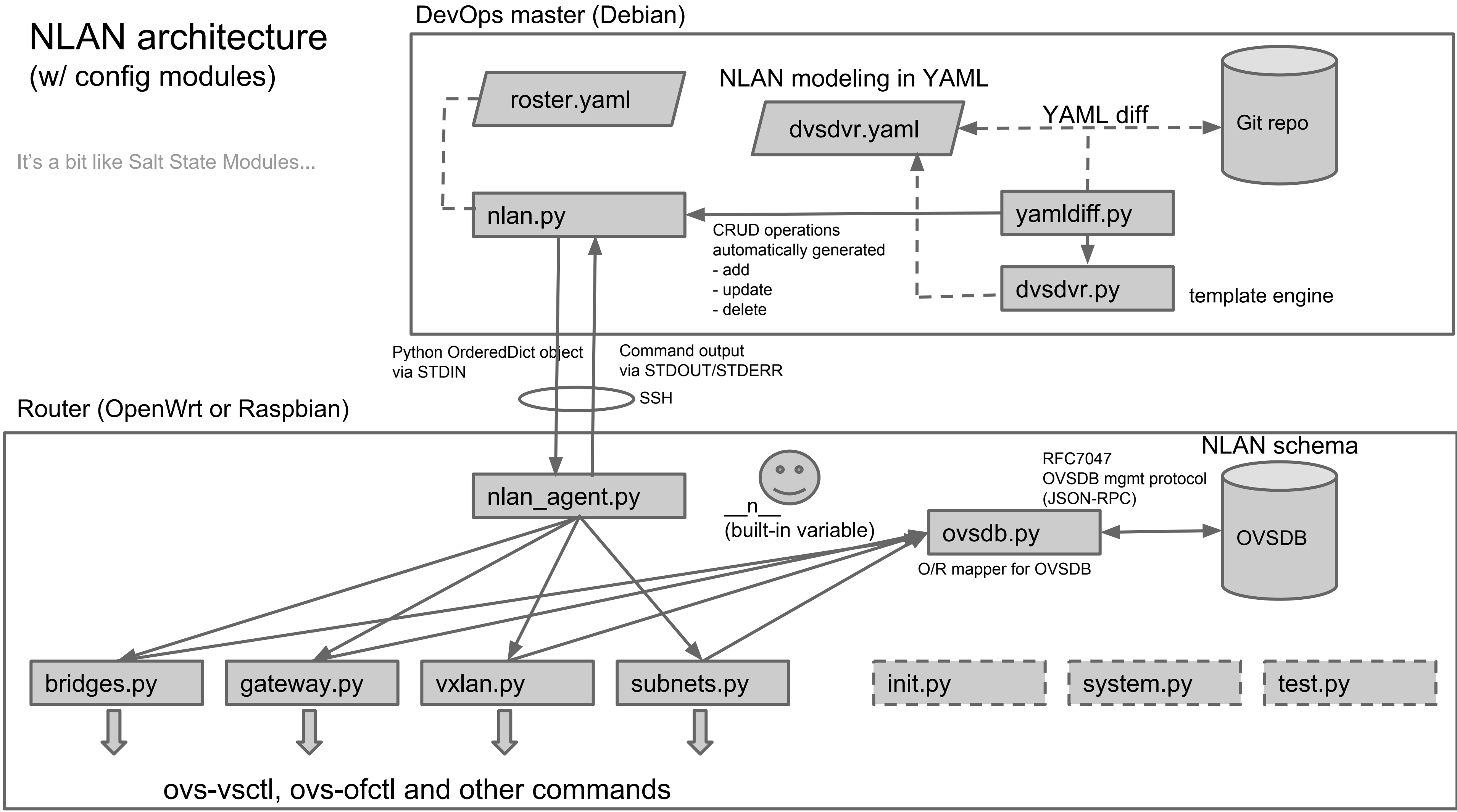
04/13-23:16:33.862906 8.8.8.8 -> 10.0.1.102
ICMP TTL:63 TOS:0x0 ID:38078 IpLen:20 DgmLen:84
Type:0 Code:0 ID:49496 Seq:26 ECHO REPLY
=+

Home-made DevOps tool ‘NLAN’

- 100% Python implementation
- Borrwed a lot of ideas from SaltStack
  - Model-driven procedure
  - YAML-based states w/ a simple template engine
  - Imperative/declarative state rendering
- Works with OpenWrt with minimal Python
  - opkg install python-mini
  - opkg install python-json
  - sshd only
- OVSDB as a local config mgmt database
- State schema defined in YAML
  - merged with OVSDB schema in JSON

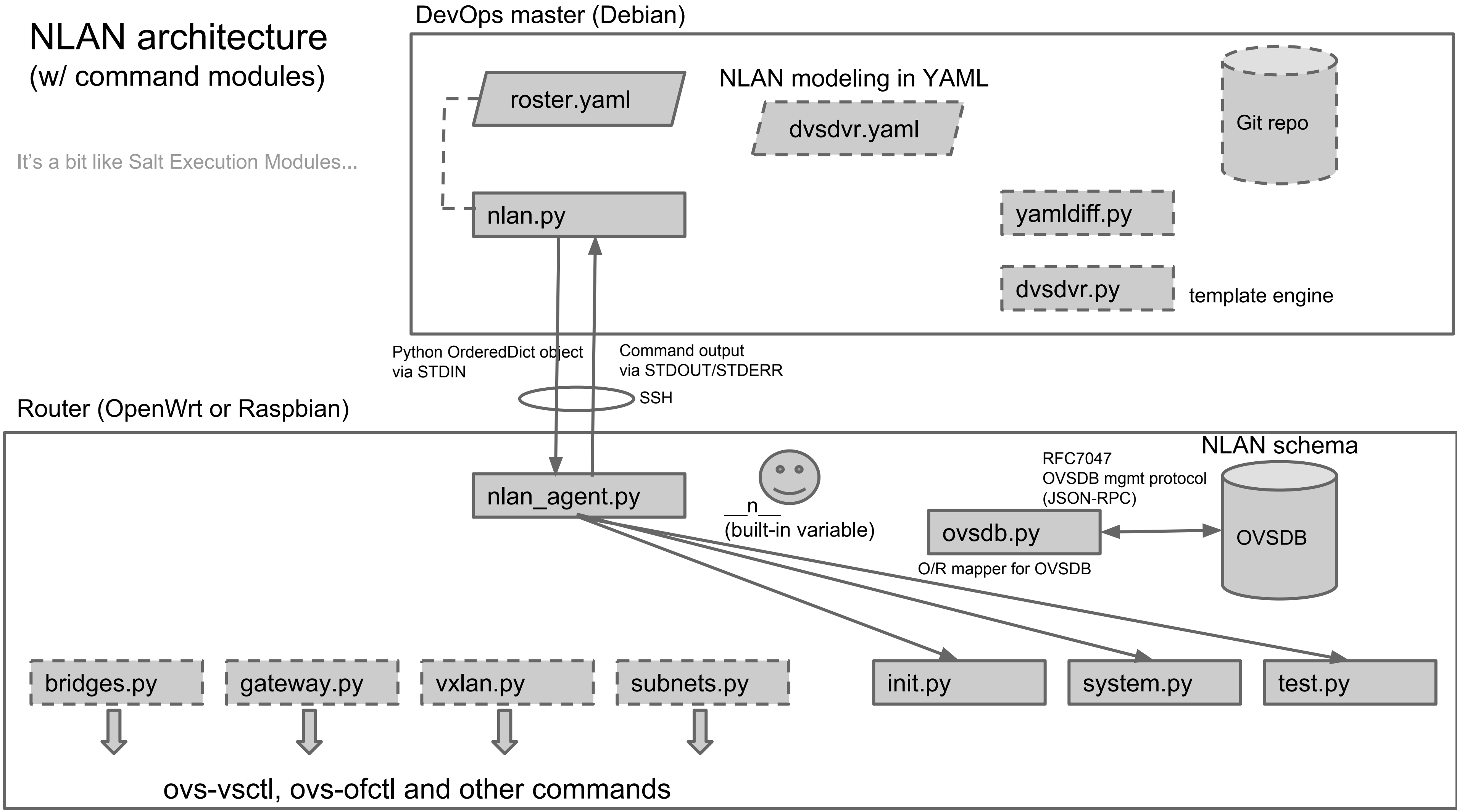
NLAN architecture
(w/ config modules)

It’s a bit like Salt State Modules...



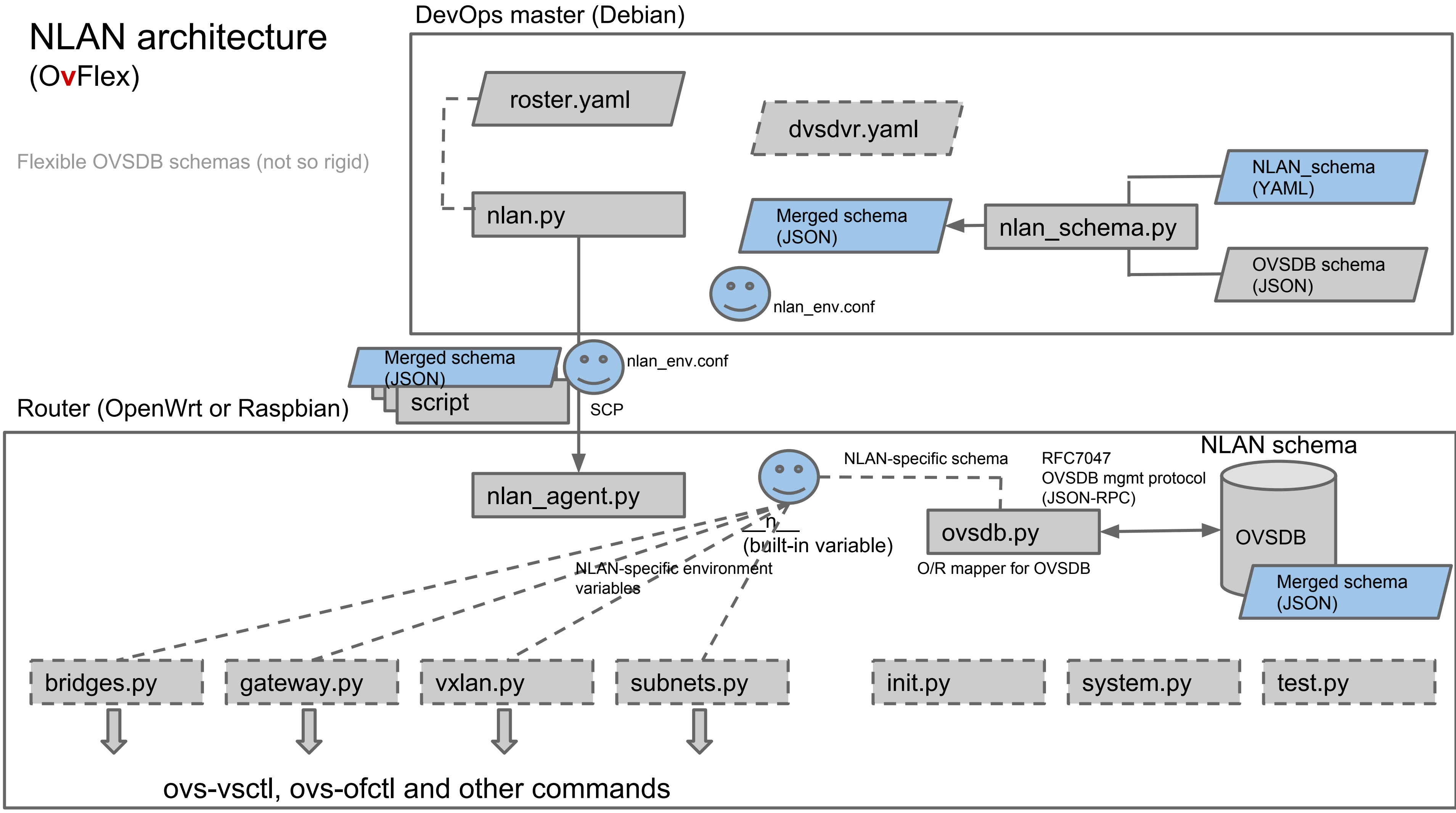
NLAN architecture
(w/ command modules)

It's a bit like Salt Execution Modules...

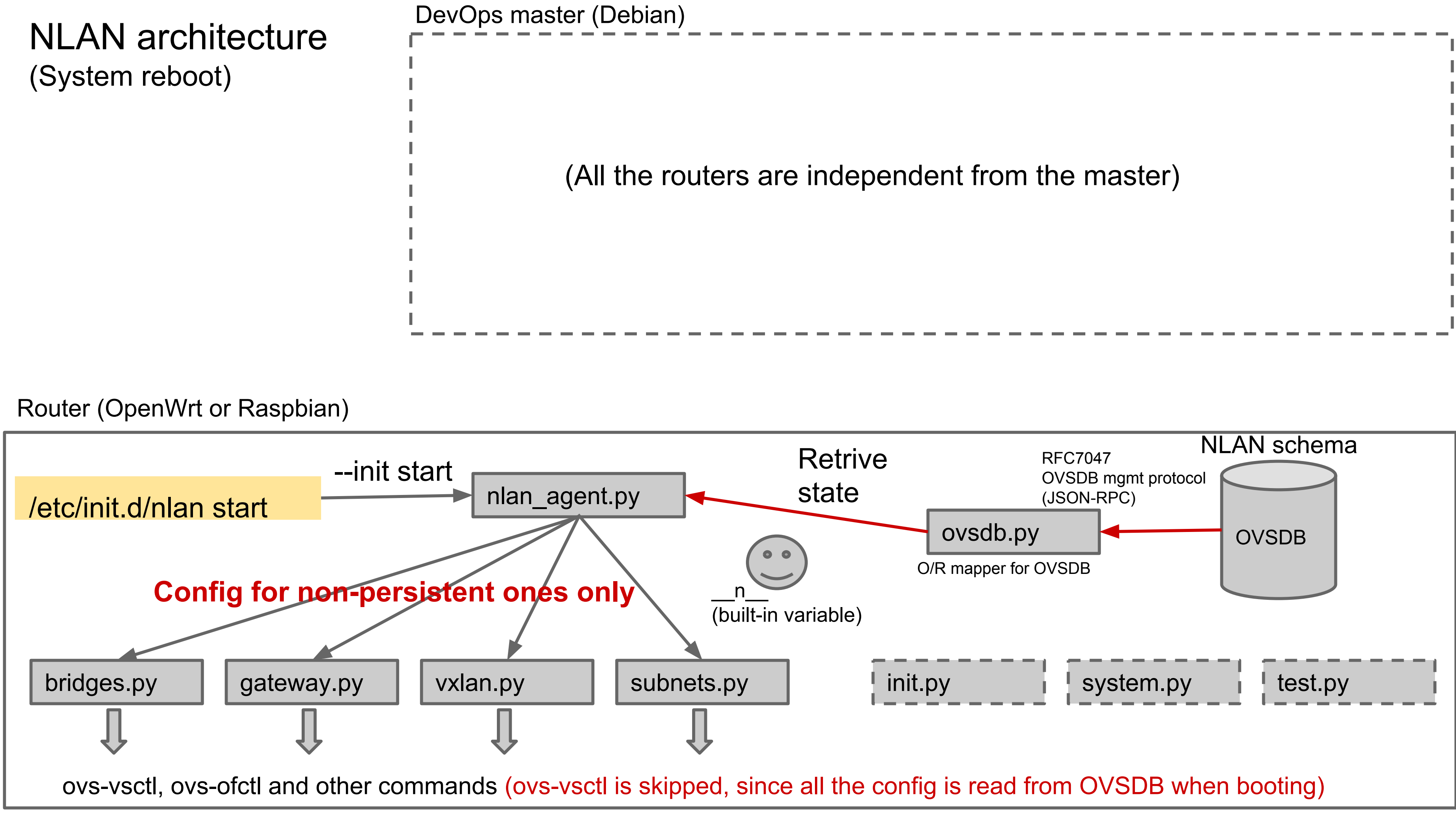


NLAN architecture
(OvFlex)

Flexible OVSDB schemas (not so rigid)



NLAN architecture
(System reboot)



OvFlex -- Flexible OVSDB schemas

OpFlex:

http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731302.html

OvFlex:

“It uses dynamic, flexible schemas for interaction with devices, effectively increasing the network to a higher common denominator feature set. The Open vSwitch Database (OVSDB) management protocol allows configuration of high-level abstract data models as well as basic primitives such as ports and bridges, and can support SDN geeks’ innovations”

OVSDB schema (Open_vSwitch database)

“Basic primitives”


```
{"name": "Open_vSwitch",
 "version": "7.4.1",
 "cksum": "951746691 20389",
 "tables": {
  "NLAN": {
   "columns": {
    "bridges": {
     "type": {"key": {"type": "uuid",
                 "refTable": "NLAN_Bridges"},
          "min": 0, "max": 1}},
    "services": {
     "type": {"key": {"type": "uuid",
                 "refTable": "NLAN_Service"},
          "min": 0, "max": "unlimited"}},
    "gateway": {
     "type": {"key": {"type": "uuid",
                 "refTable": "NLAN_Gateway"},
          "min": 0, "max": 1}},
    "vxlan": {
     "type": {"key": {"type": "uuid",
                 "refTable": "NLAN_VXLAN"},
          "min": 0, "max": 1}},
    "subnets": {
     "type": {"key": {"type": "uuid",
                 "refTable": "NLAN_Subnet"},
          "min": 0, "max": "unlimited"}}},
   "isRoot": true,
      :
```


Also possible to convert it into JSON format by using some libraries.


```
cksum: 951746691 20389
name: Open_vSwitch
tables:
  Bridge:
    columns:
      controller:
        type:
          key: {refTable: Controller, type: uuid}
          max: unlimited
          min: 0
      datapath_id:
        ephemeral: true
        type: {key: string, max: 1, min: 0}
      datapath_type: {type: string}
      external_ids:
        type: {key: string, max: unlimited, min: 0, value: string}
      fail_mode:
        type:
                    :
```

NLAN states in OVSDB

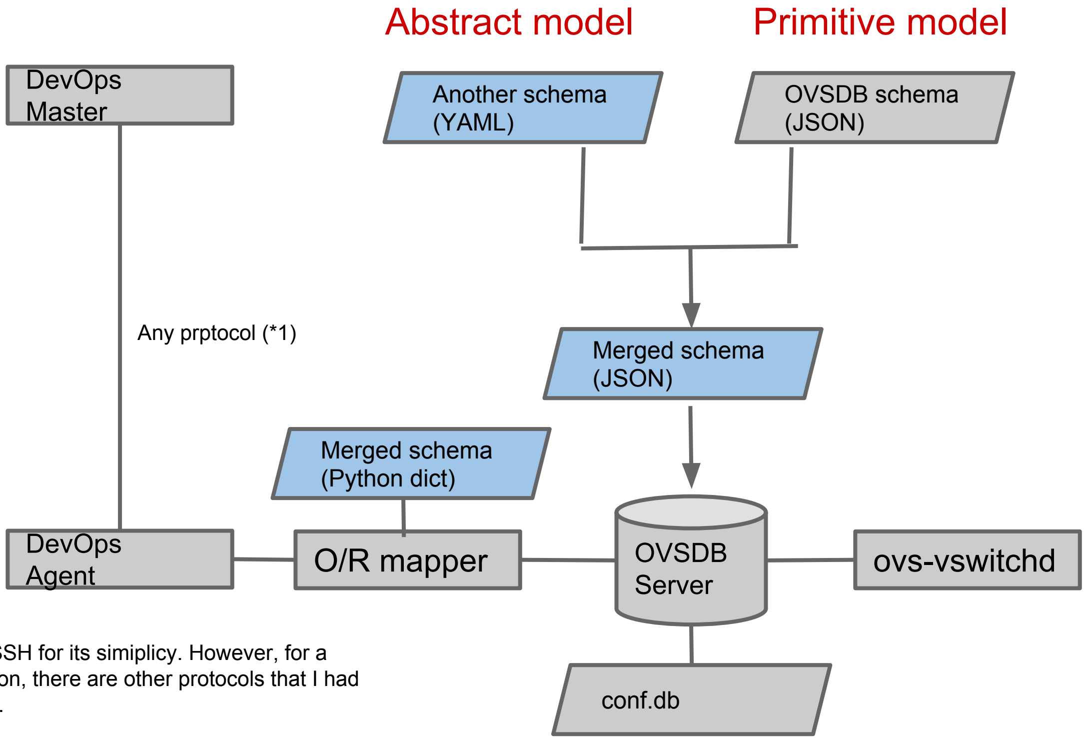
(ovsdb-client dump Open_vSwitch)

```
NLAN table
_uuid                                bridges                              gateway services                               subnets
vxlan
------------------------------------ ------------------------------------ ------- --------------------------------------
---------------------------------------------------------------------------- ------------------------------------
2928cfe4-1615-4e5a-ad56-5e10b881a9bb b96c47b6-1ace-4ea4-af7c-946d2623cacc []      [bf9038c4-5b02-45d0-adfd-51176b24ca49] [376c0b57-9108-44c1-8fcd-d28634953977, d11f17e3-1457-
4a24-b7fd-34710af48ca2] 42da0c85-688a-4231-9082-5e43a63756bd

NLAN_Bridges table
_uuid                                controller ovs_bridges
------------------------------------ ---------- -----------
b96c47b6-1ace-4ea4-af7c-946d2623cacc []         enabled

NLAN_Gateway table
_uuid network rip
----- ------- ---

NLAN_Service table
_uuid                                chain                  name
------------------------------------ ---------------------- --------
bf9038c4-5b02-45d0-adfd-51176b24ca49 ["dmz.1001", "mz.101"] "snort1"

NLAN_Subnet table
_uuid                                default_gw ip_dvr ip_vhost mode peers                              ports        vid vni
------------------------------------ ---------- ------ -------- ---- ---------------------------------- ------------ --- ----
376c0b57-9108-44c1-8fcd-d28634953977 []         []     []       []   ["192.168.1.101"]                  ["dmz.1001"] 111 1001
d11f17e3-1457-4a24-b7fd-34710af48ca2 []         []     []       []   ["192.168.1.102", "192.168.1.103"] ["mz.101"]   1   101

NLAN_VXLAN table
_uuid                                local_ip        remote_ips
------------------------------------ --------------- ---------------------------------------------------
42da0c85-688a-4231-9082-5e43a63756bd "192.168.1.104" ["192.168.1.101", "192.168.1.102", "192.168.1.103"]
```

Merging schemas

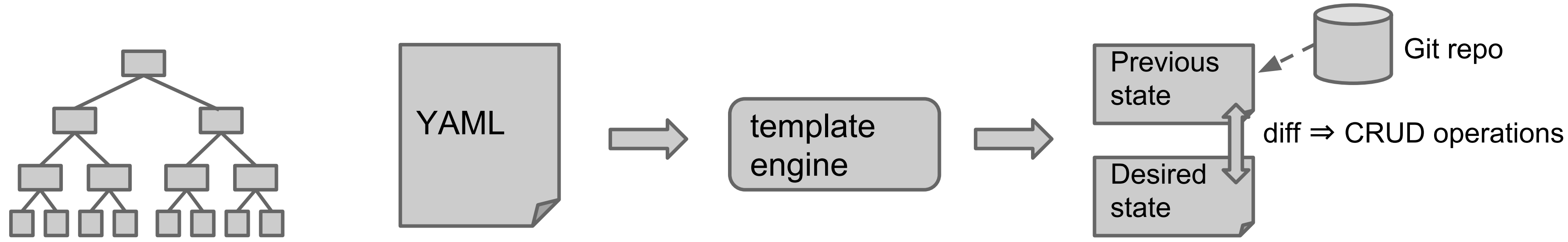
OVSDB schema can also express more abstract data models.

```
   :
NLAN_Service:
  columns:
    chain:
      type:
        key: {type: string}
        min: 0
        max: unlimited
    name:
      type:
        key: {type: string}
        min: 1
        max: 1
  indexes:
  - [name]
     :
```



(*1) I just use SSH for its simiplicy. However, for a scalability reason, there are other protocols that I had better consider.

Model-driven service abstraction (cont’d)



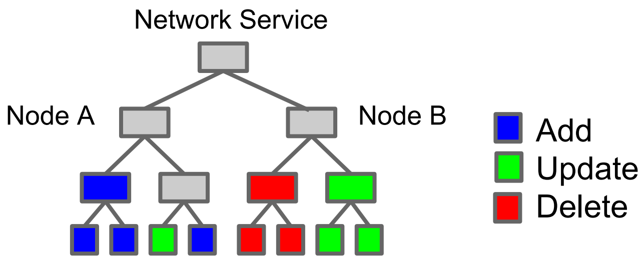
Step1: define network service model

Step2: write the mode as “desired state” in YAML format w/ some placeholders for a template engine

Step3: write a template engine to fill out the placeholders.

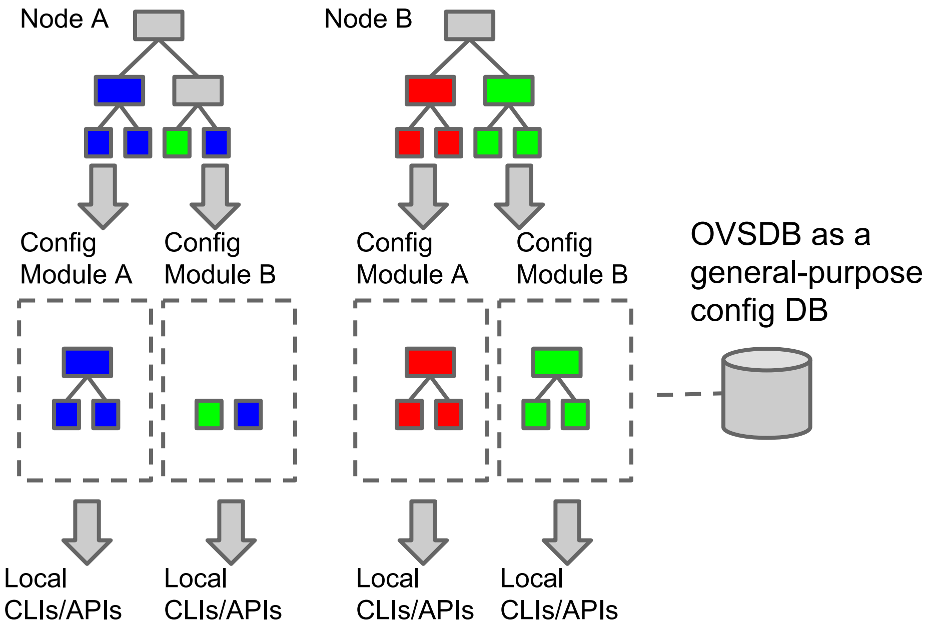
Step4: NLAN-Master generates CRUD operations comparing the desired state with the previous state

Model-driven service abstraction



Step5: Now CRUD-operations (= difff of the previous and the desired states) are in the form of Python OrderedDict object

Step6: NLAN-Agent routes the CRUD operations to corresponding nodes/modules



Template and placeholders example

```
#!template.dvsdvr
openwrt1:
    :
  vxlan:
    local_ip: <local_ip>
    remote_ips: <remote_ips>
  subnets:
    - vid: 1
    vni: 101
    ip_dvr: '10.0.1.1/24'
    ip_vhost: '10.0.1.101/24'
    ports:
    - eth0.1
    peers: <peers>
    :
```

template engine
“template.
dvsdvr”

- generates a local ip address
- generates VXLAN remote ip addresses
- generates broadcast tree per VNI
- automatically resolves dependencies among parameters

```
#!template.dvsdvr
openwrt1:
    :
  vxlan:
    local_ip: '192.168.1.101'
    remote_ips: [‘192.168.1.102’, ‘192.168.1.103’, ‘192.168.1.104’]]
  subnets:
    - vid: 1
    vni: 101
    ip_dvr: '10.0.1.1/24'
    ip_vhost: '10.0.1.101/24'
    ports:
    - eth0.1
    peers: [‘192.168.1.102’, ‘192.168.1.104’]
    :
```

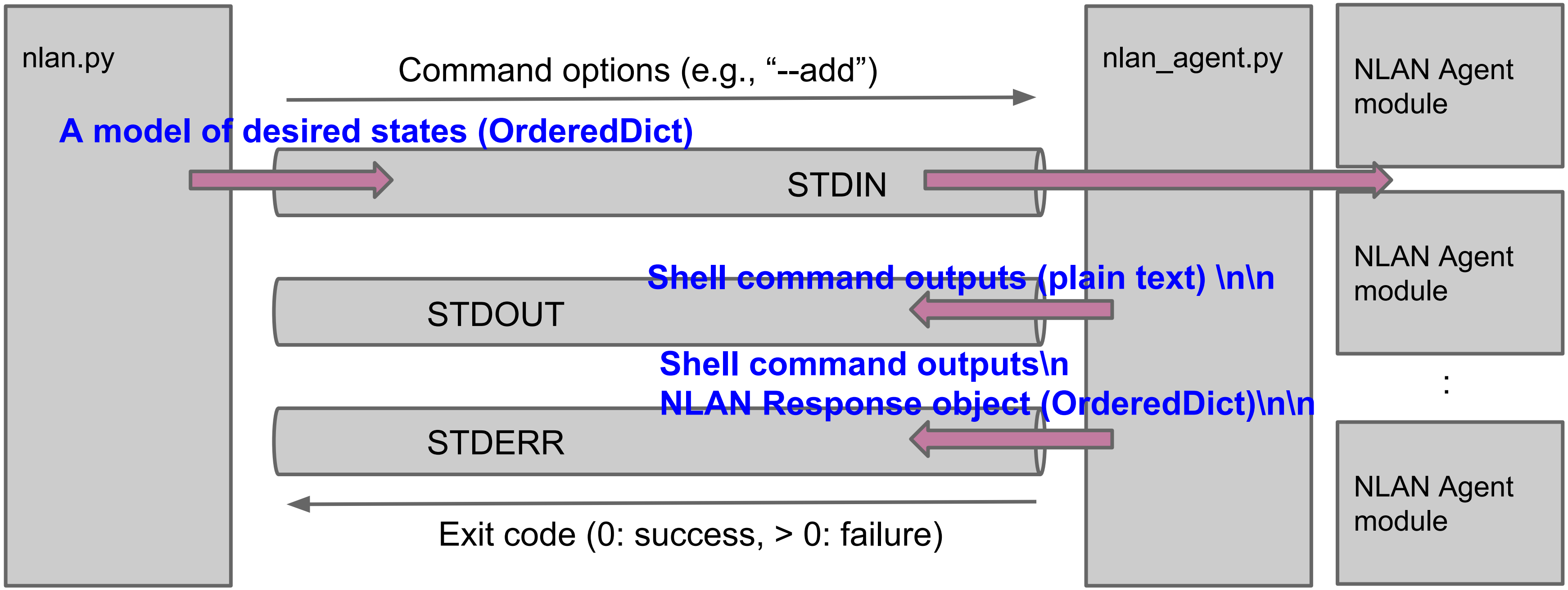
Model of desired state

- NLAN-Master sends Python OrderedDict to NLAN-Agent via ssh STDIN.
- To be exact, string form of an OrderedDict object

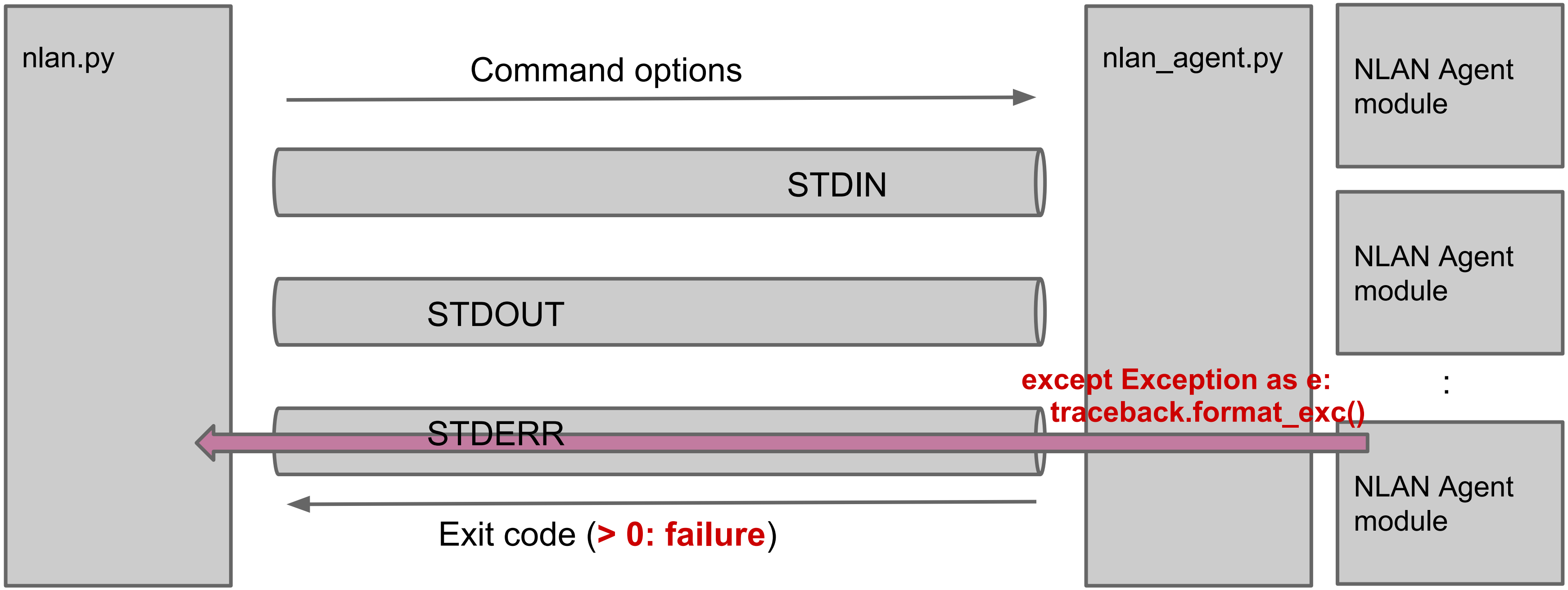
"OrderedDict([('bridges', {'ovs_bridges': 'enabled'}), ('gateway', {'network': 'eth0.2', 'rip': 'enabled'}), ('vxlan', {'remote_ips': ['192.168.1.103', '192.168.1.102', '192.168.1.104'], 'local_ip': '192.168.1.101'}), ('subnets', {('vni', 1): {'ip_vhost': '192.168.100.101/24', 'ip_dvr': '192.168.100.1/24', 'peers': ['192.168.1.102', '192.168.1.103'], 'vid': 2, 'vni': 1}, ('vni', 103): {'peers': ['192.168.1.102', '192.168.1.103', '192.168.1.104'], 'vid': 3, 'ip_vhost': '10.0.3.101/24', 'vni': 103, 'ip_dvr': '10.0.3.1/24', 'ports': ['eth0.3']}, ('vni', 101): {'peers': ['192.168.1.102', '192.168.1.103', '192.168.1.104'], 'vid': 1, 'ip_vhost': '10.0.1.101/24', 'vni': 101, 'ip_dvr': '10.0.1.1/24', 'ports': ['eth0.1']}})])"

- I don’t use JSON, since NLAN is 100% Python implementation.

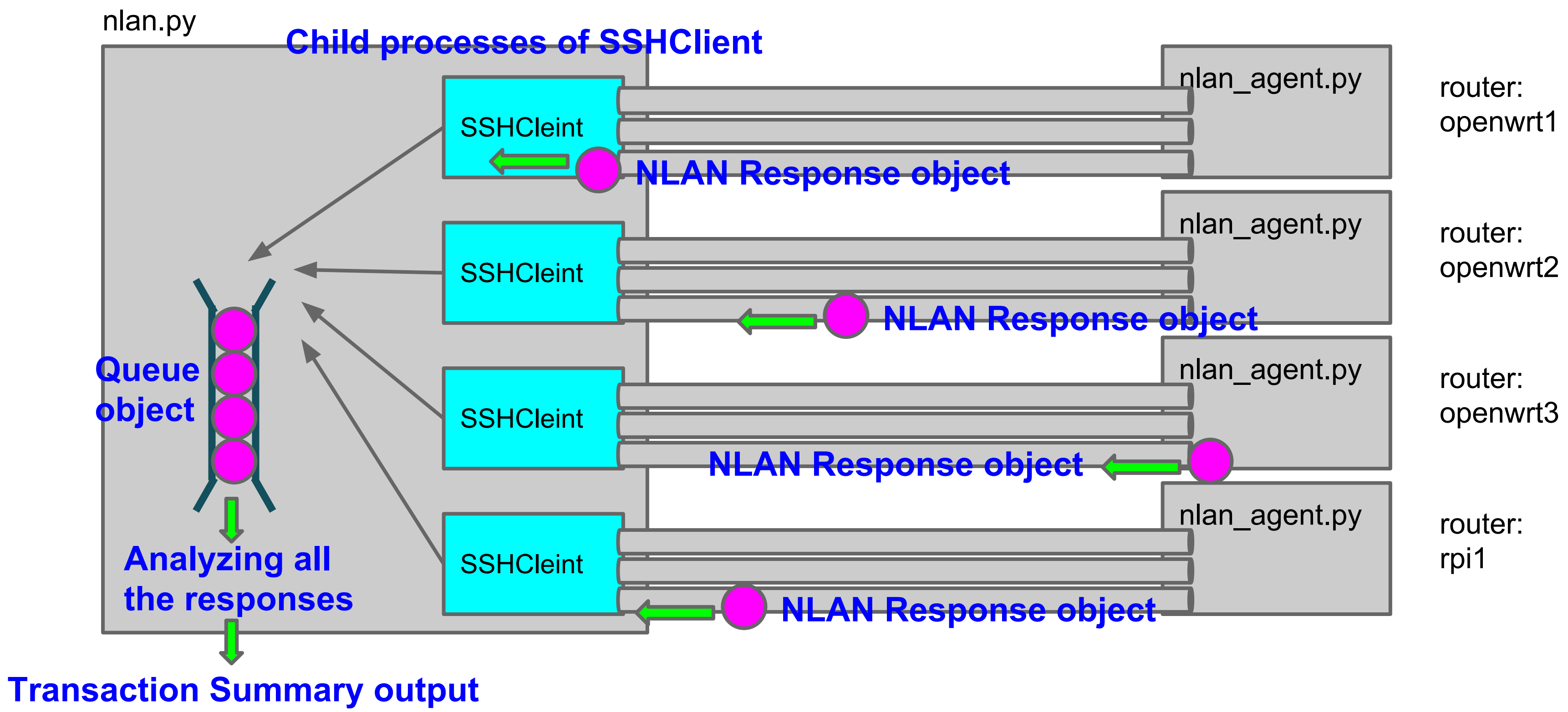
NLAN Request/Response over SSH



Exception handling



Parallel SSH sessions

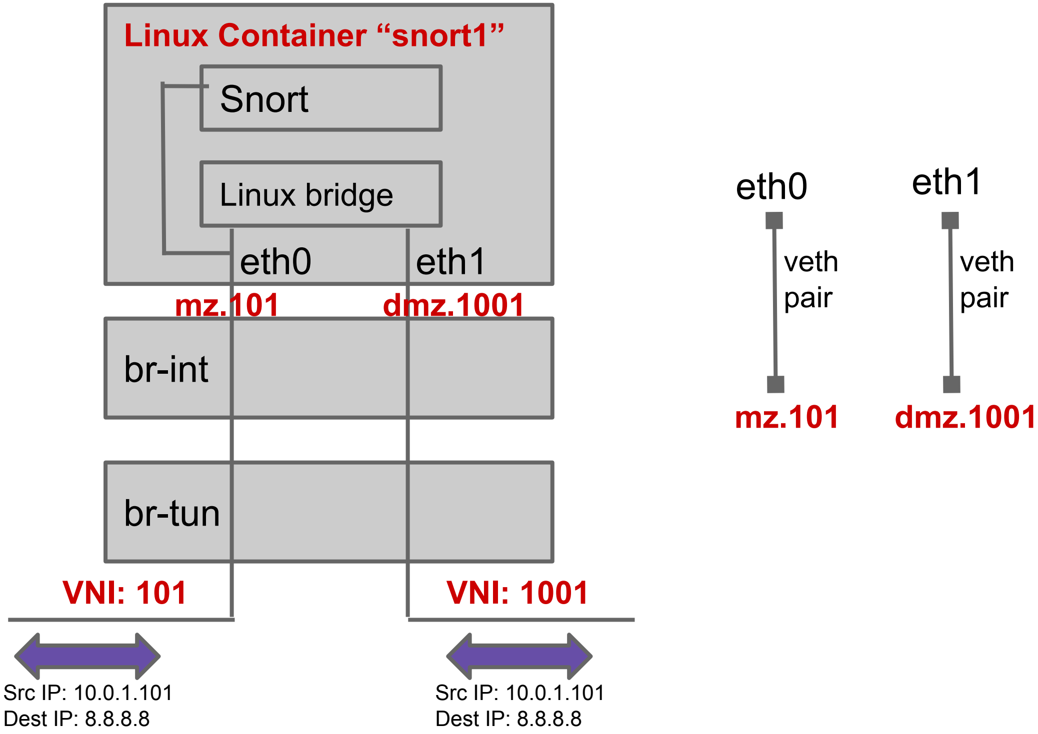


Transaction Summary output

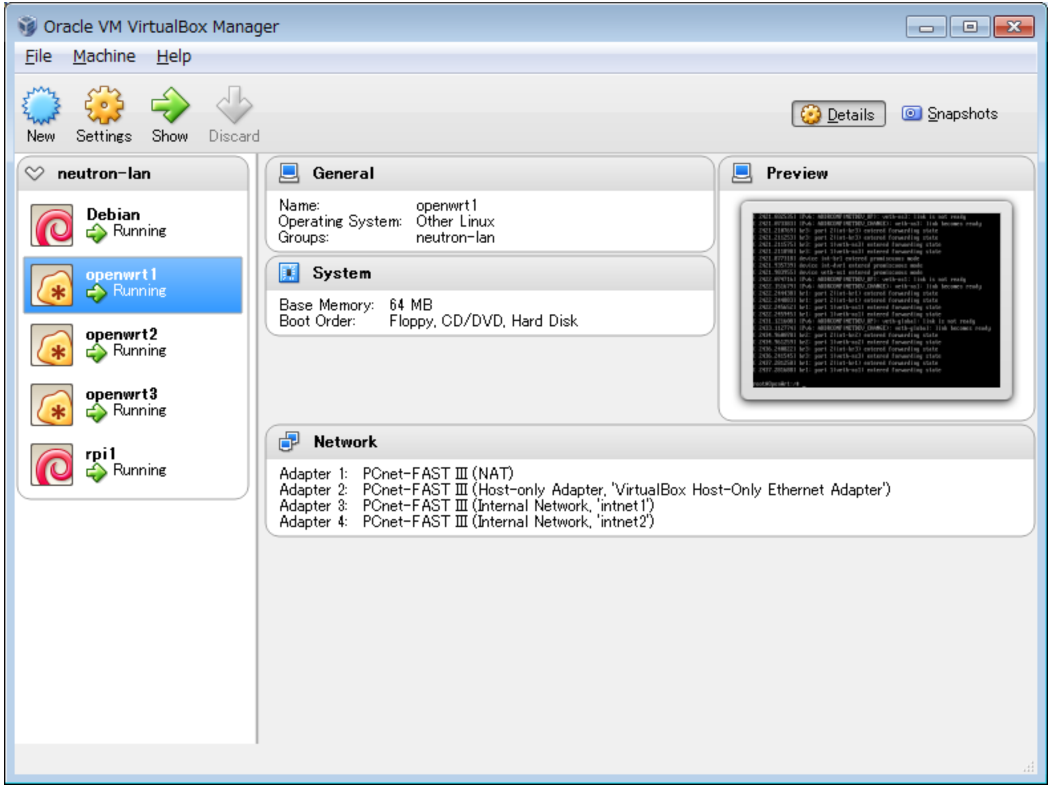
```
=+=+= Transaction Summary =+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+

Start Time: 2014-04-22 20:48:31.835552

Router         Result    Elapsed Time
---------------------------------------
openwrt1       :-)       2.88(sec)
openwrt3       :-)       2.99(sec)
openwrt2       :-)       3.00(sec)
rpi1           :-)       3.08(sec)
```

Service Function Chaining in YAML

```yaml
rpi1:
  bridges:
    ovs_bridges: enabled
  services: # Service Functions
    - name: snort1
    chain: [mz.101, dmz.1001]
  vxlan:
    local_ip: <local_ip>
    remote_ips: <remote_ips>
  subnets:
    - vid: 111
    vni: 1001
    peers: <peers>
    ports: <sfports>
    - vid: 1
    vni: 101
    peers: <peers>

    ports: <sfports>
```



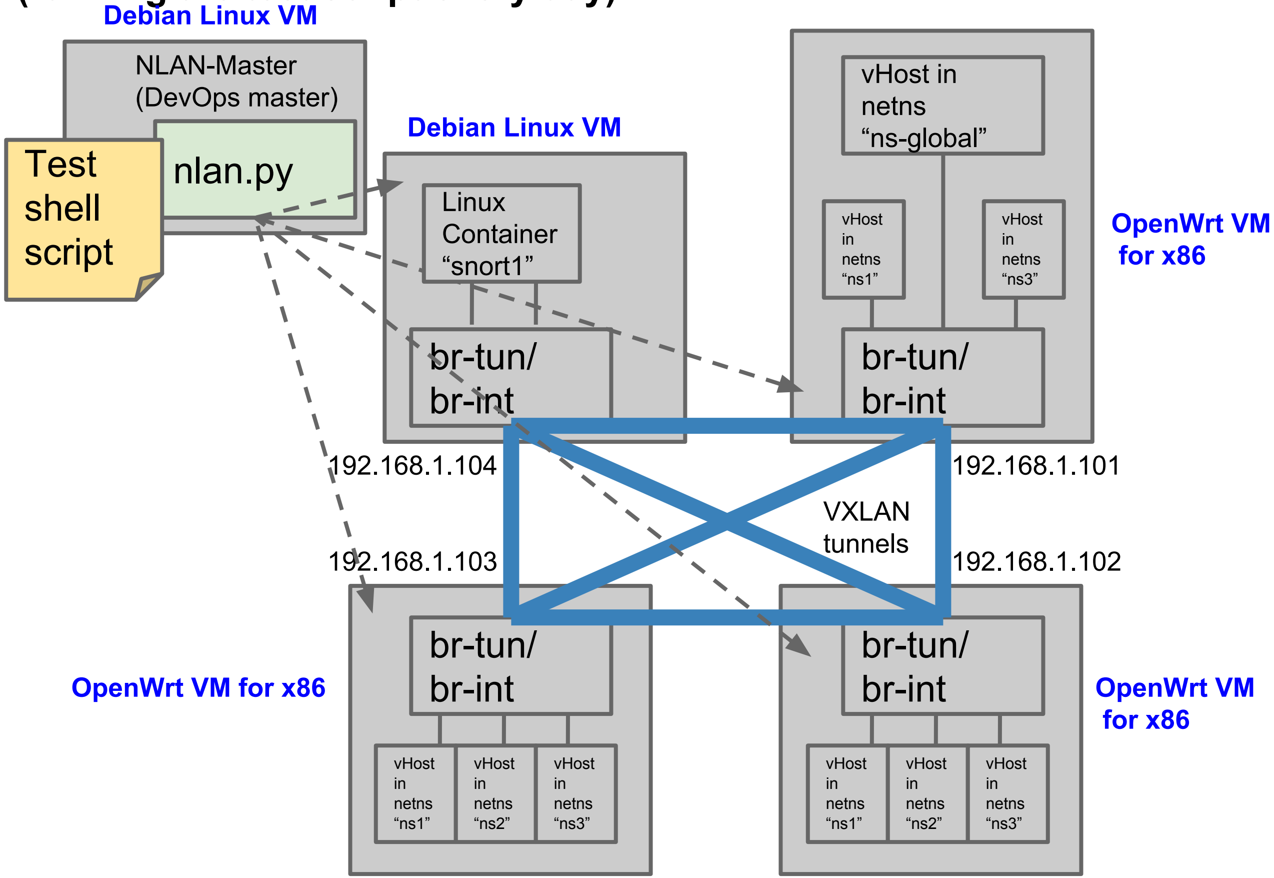
NLAN Software Development environment on VirtualBox



- Five VMs running on one Win7 PC.
  - Two Debian VMs
  - Three OpenWrt VMs
- OpenWrt image for x86
  - I built the kernel with Open vSwtich 2.0.0 and netns/veth/LXC support
  - Very light-weight Linux supporting Open vSwitch 2.0.0 ⇒ Alternative to mininet 2.0.0
- Network adapters setting
  - Internet access: “NAT”
  - Management: “Host-Only”
  - NLAN underlay: “Internal”

Integration Test environment on VirtualBox

(running the test script every day)



- Open vSwitch-based network more realisic than mininet 2.0
  - Every vSwitch with full-fledged Linux
  - netns-based virtual hosts
- Mimics "Beremetal Switch"
- Integration Test scipt running on Debian Linux VM
  - Makes use of "nlan. py"