neutron-lan

SDN study environement @ home

github.com/alexanderplatz1999

Background and Motivation

- Too many SDN definitions
  - I have been confused a lot.
  - OpenFlow, OVSDB, Netconf, BGP extensions such as FlowSpec...
  - The latest addition: 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
- Let’s develop neutron-like SDN for my home network ⇒ let’s call it ***‘neutron-lan’***

OpenStack neutron br-tun

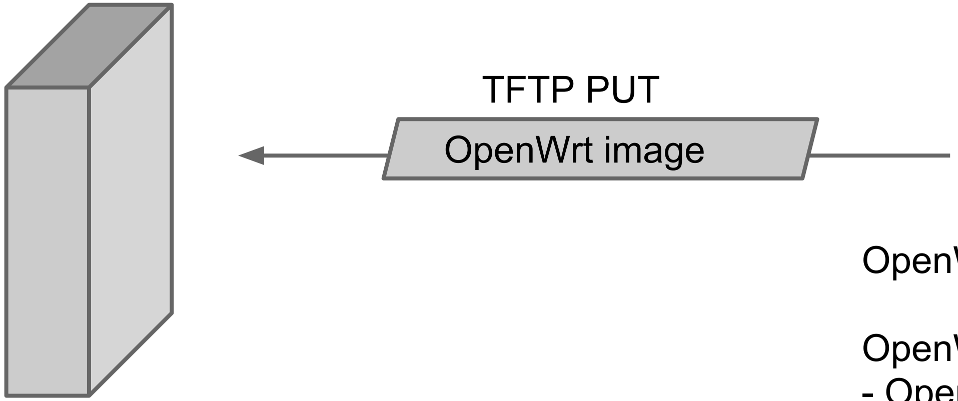
```
root@compute1:~# ovs-ofctl dump-flows br-tun
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=9638.539s, table=0, n_packets=0, n_bytes=0, idle_age=9638, priority=1,in_port=3 actions=resubmit(,2)
 cookie=0x0, duration=9642.632s, table=0, n_packets=502, n_bytes=51575, idle_age=1003, priority=1,in_port=1 actions=resubmit(,1)
 cookie=0x0, duration=9628.395s, table=0, n_packets=657, n_bytes=68175, idle_age=1003, priority=1,in_port=2 actions=resubmit(,2)
 cookie=0x0, duration=9642.472s, table=0, n_packets=2, n_bytes=140, idle_age=9635, priority=0 actions=drop
 cookie=0x0, duration=9642.102s, table=1, n_packets=10, n_bytes=1208, idle_age=1700, priority=0,dl_dst=01:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,21)
 cookie=0x0, duration=9642.278s, table=1, n_packets=492, n_bytes=50367, idle_age=1003, priority=0,dl_dst=00:00:00:00:00:00/01:00:00:00:00:00 actions=resubmit(,20)
 cookie=0x0, duration=9636.347s, table=2, n_packets=657, n_bytes=68175, idle_age=1003, priority=1,tun_id=0x3 actions=mod_vlan_vid:1,resubmit(,10)
 cookie=0x0, duration=9641.973s, table=2, n_packets=1, n_bytes=94, idle_age=9636, priority=0 actions=drop
 cookie=0x0, duration=9641.823s, table=3, n_packets=0, n_bytes=0, idle_age=9641, priority=0 actions=drop
 cookie=0x0, duration=9641.677s, table=10, n_packets=657, n_bytes=68175, idle_age=1003, priority=1 actions=learn(table=20,hard_timeout=300,priority=1,
NXM_OF_VLAN_TCI[0..11],NXM_OF_ETH_DST[]=NXM_OF_ETH_SRC[],load:0->NXM_OF_VLAN_TCI[],load:NXM_NX_TUN_ID[]->NXM_NX_TUN_ID[],output:
NXM_OF_IN_PORT[]),output:1
 cookie=0x0, duration=9641.545s, table=20, n_packets=0, n_bytes=0, idle_age=9641, priority=0 actions=resubmit(,21)
 cookie=0x0, duration=9636.651s, table=21, n_packets=10, n_bytes=1208, idle_age=1700, hard_age=9628, priority=1,dl_vlan=1 actions=strip_vlan,set_tunnel:0x3,output:3,
output:2
```

It’s like VPLS in a data center...

Project ‘neutron-lan’ characteristics

- Cheap routers as ‘Baremetal Switch’
  - OpenWrt routers and Raspberry Pi
  - u-boot for installing new firmware
- OpenWrt kernet/kernel modules rebuilt(compiled) for supporting additional features
  - Open vSwitch 2.0.0, network namespaces(netns), veth and LXC.
- Home-made DevOps tool ‘NLAN’ for small routers w/ small memory/storage footprints
  - 100% Python implementation
  - YAML-based state rendering (Python’s OrderedDict internally)
  - Model-Driven Service Abstraction Layer
    - Both imperative and declarative state definitions
  - OVSDB as a general-purpose config database
    - RFC7047 client library
    - With NLAN-specific OVSDB schema and schema extensions
  - MIME Mutipart messages to covery various kinds of info
    - CRUD/RPC Transaction outputs
    - Exceptions
    - Logging info
    - RFC7047 JSON-RPC transacions
- VXLAN-based edge-overlay for network virtualization
  - Broadcast tree per VNI
  - Distributed Virtual Routing topology as well as hub & spoke topology
- LXC for Network Functions Virtualization

Cheap routers as ‘Baremetal Switch’



PC

Broadband Router
w/ u-boot boot loader

OpenWrt build environment

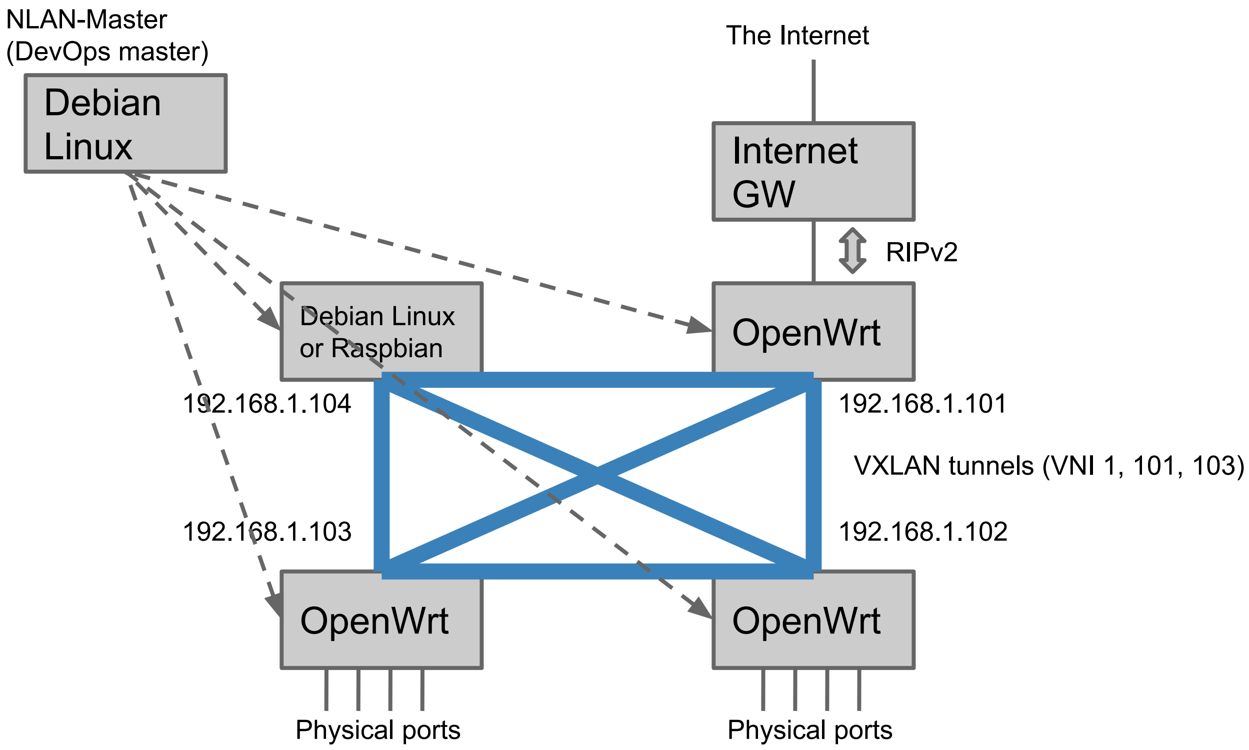
OpenWrt ‘BarrierBreaker’ image w/
- Open vSwitch 2.0.0
- Network namespaces
- Veth

Test bed (1/2)

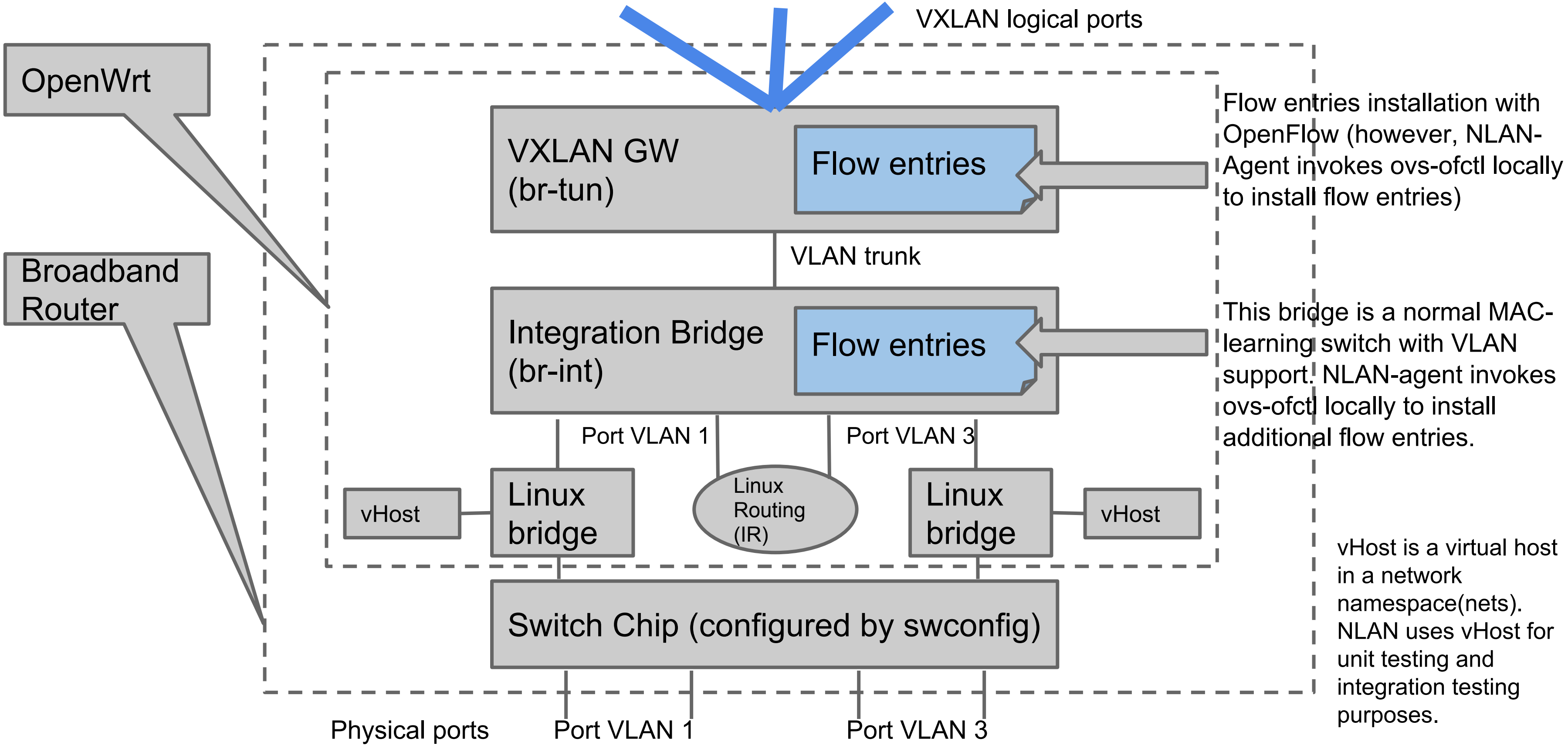
OpenWrt routers
(and Home Gateway)

Raspberry Pi

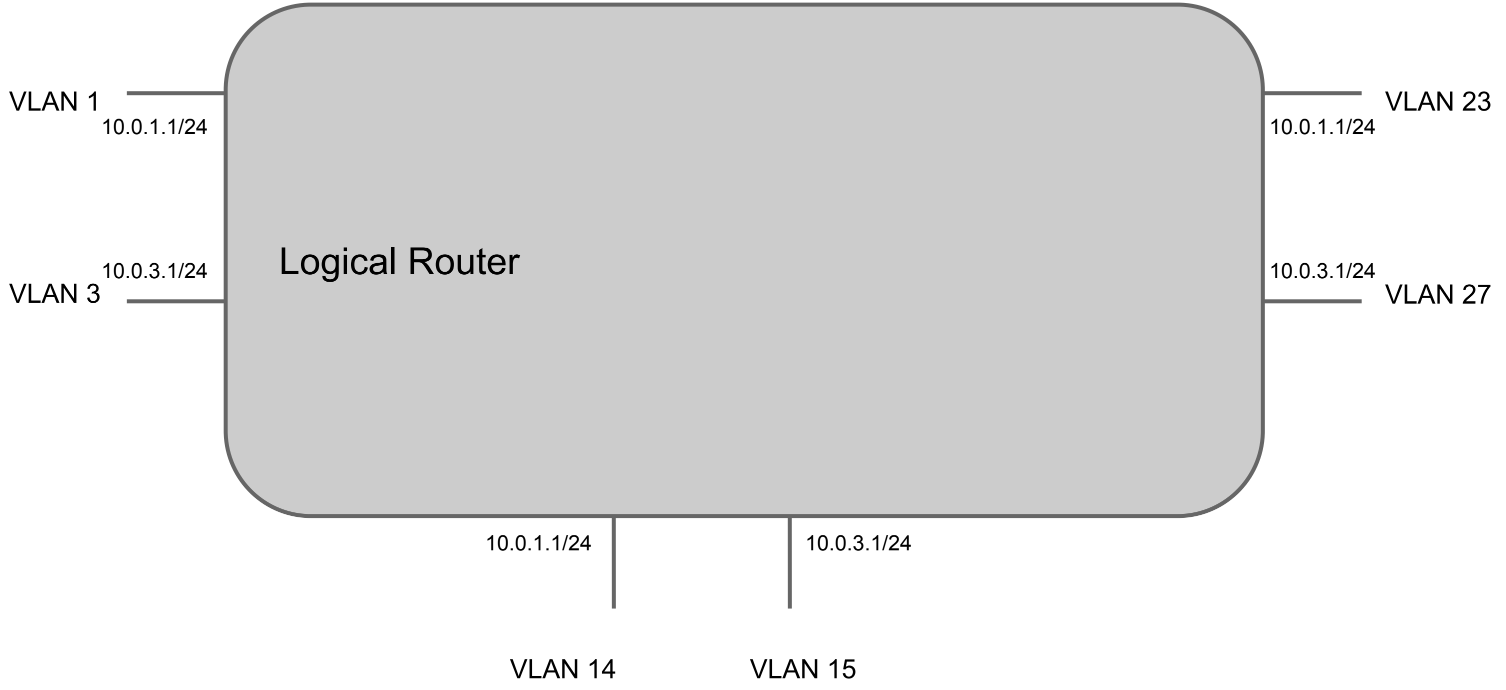
Test bed (2/2)


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

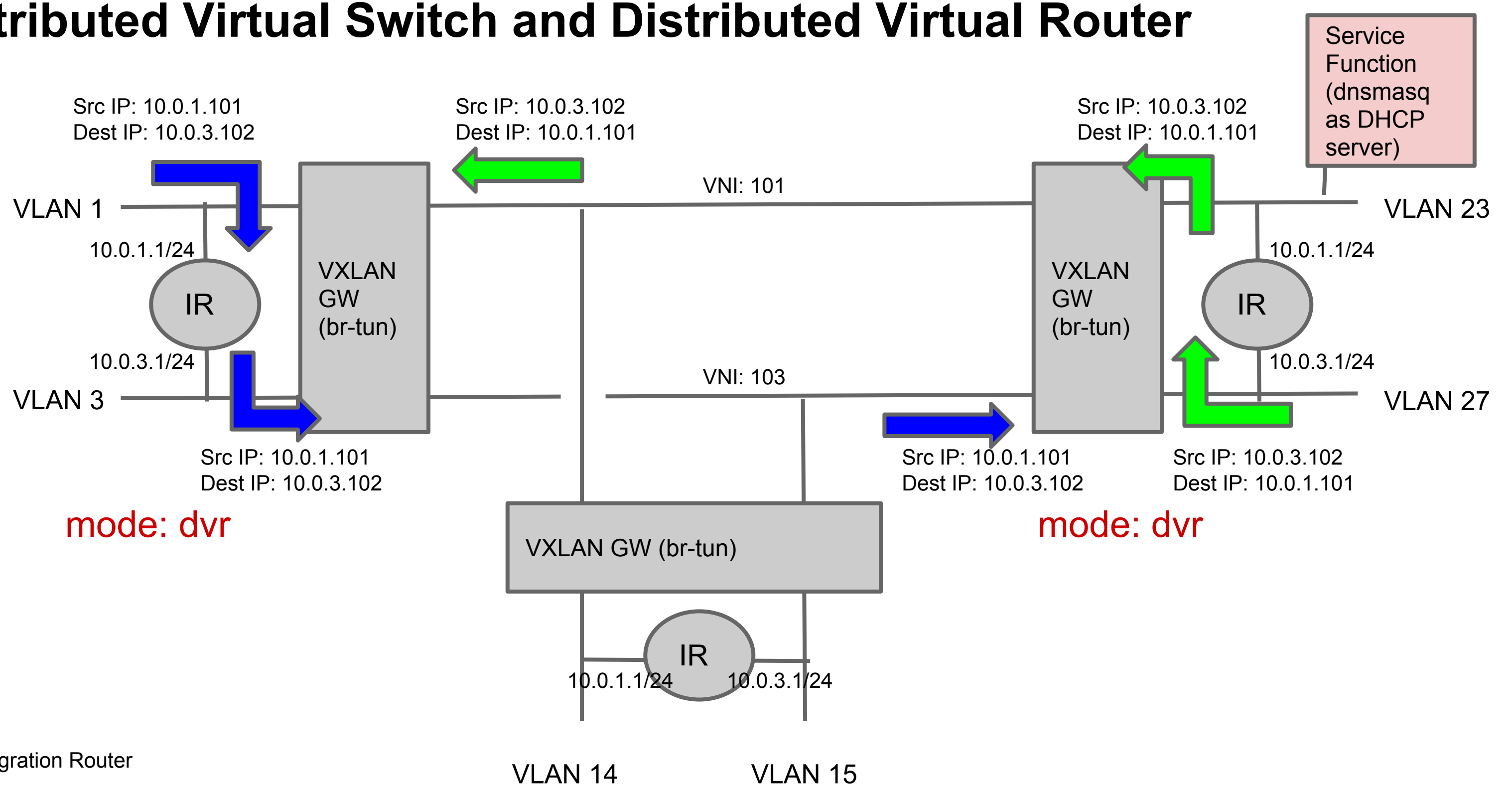
NLAN node operation mode	Virtual Network Topology
dvr	Distributed Virtual Router
hub	Hub & Spoke
spoke	Hub & Spoke
spoke_dvr	Mixture of DVR and Hub & Spoke

NLAN “subnets” state and its parmeters in YAML

```
subnets:
  - vid: 1
    vni: 1001
    ip_dvr:
      addr: '10.0.1.1/24'
      mode: hub
      dhcp: enabled
    ip_vhost: '10.0.1.101/24'
    ports: [eth0.1]
    peers: <peers>
```

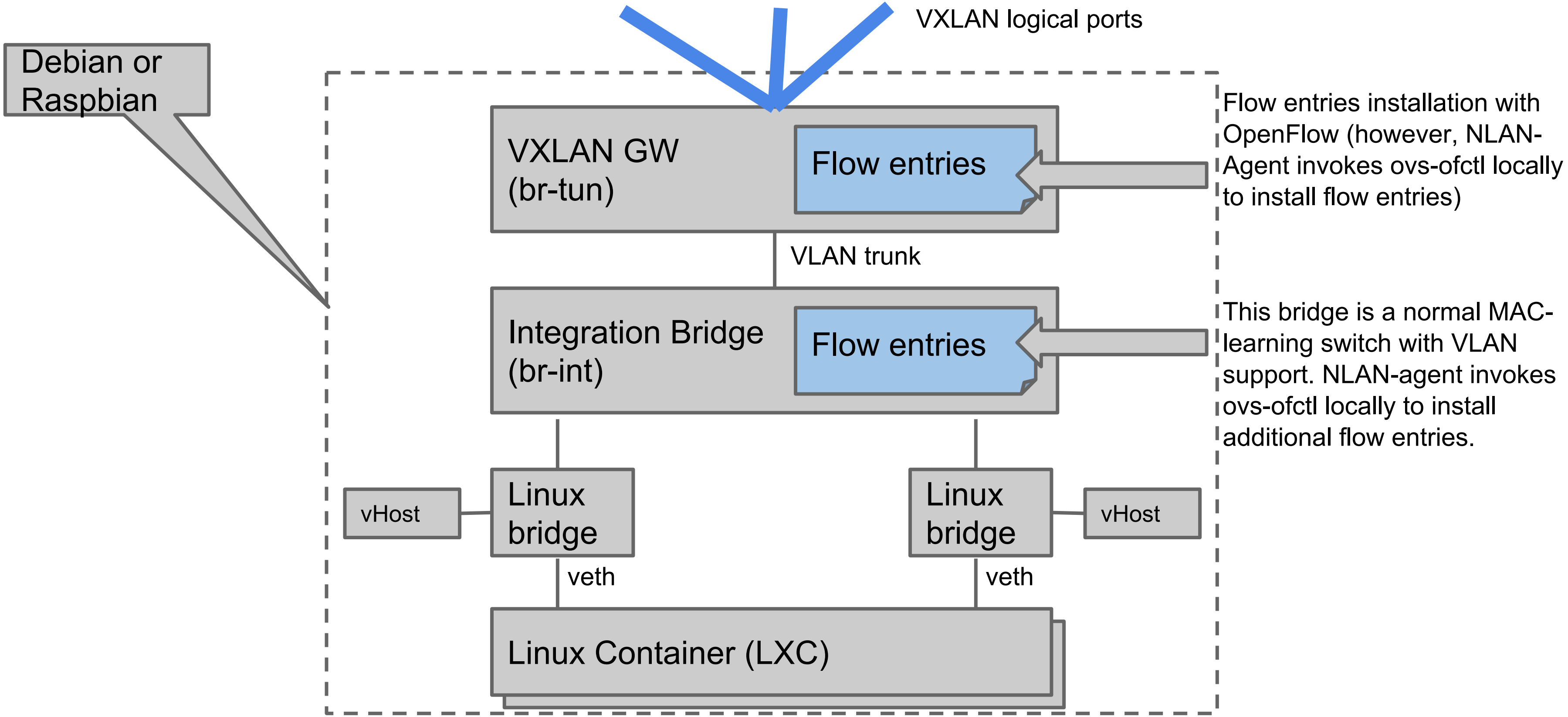
mode can be ‘dvr’, ‘hub’, ‘spoke’ or ‘spoke_dvr’

Distributed Virtual Switch and Distributed Virtual Router

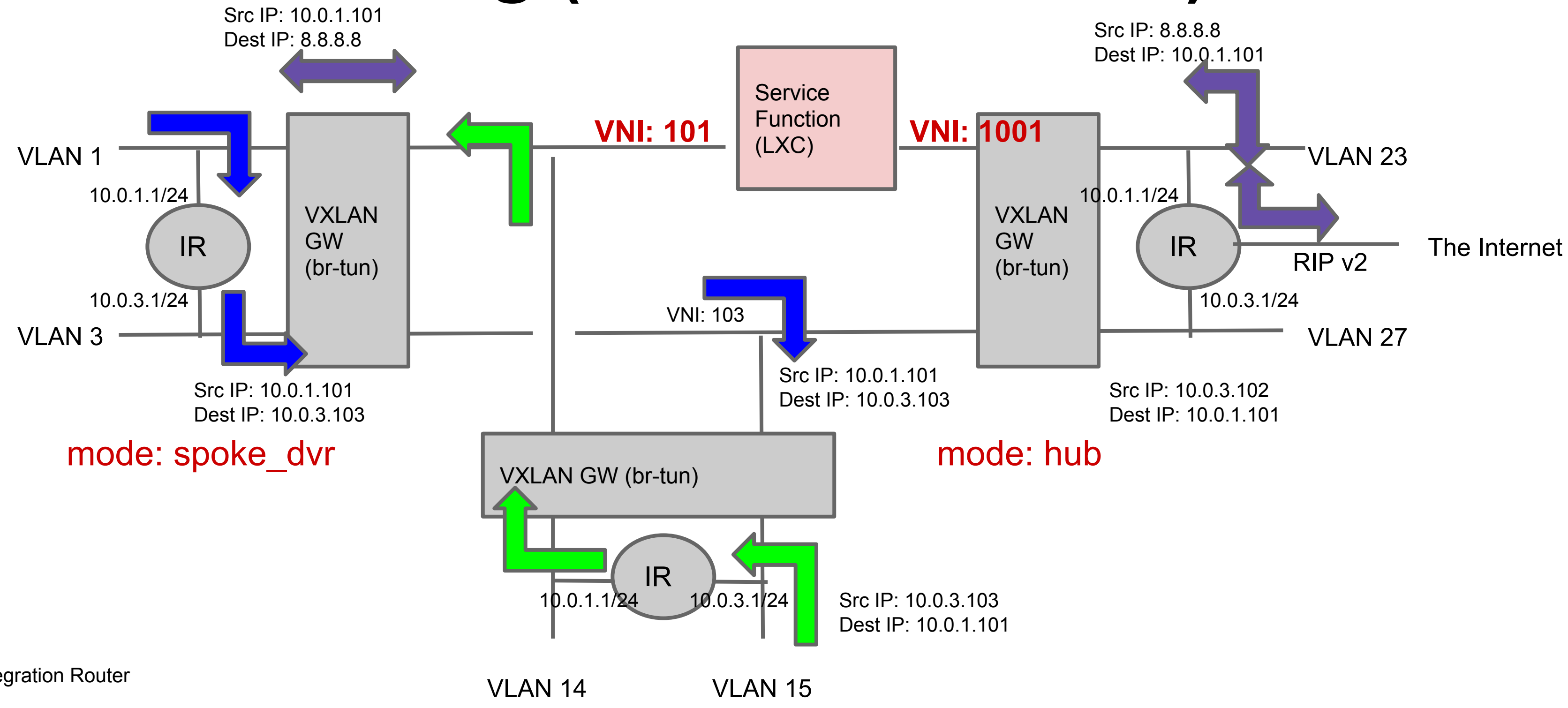


IR: Integration Router

Service Function in Linux Container

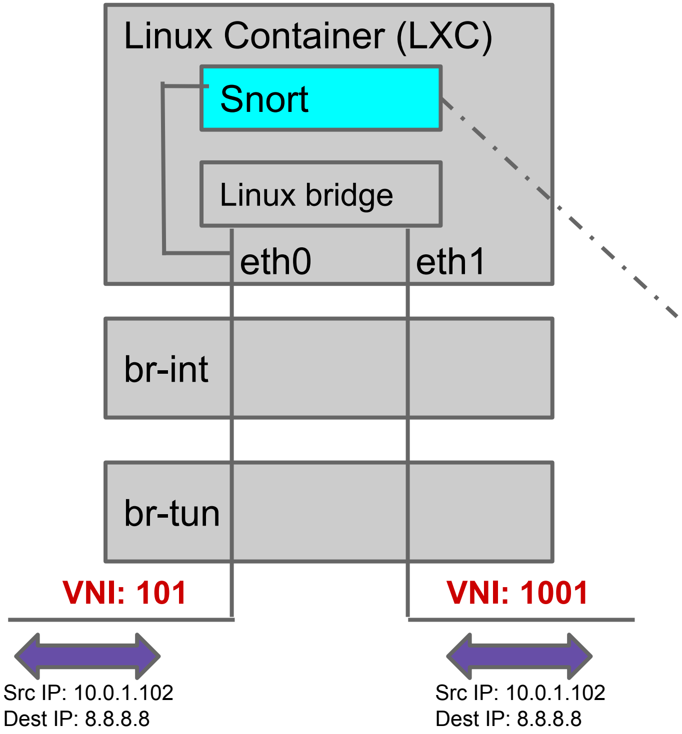


Service Chaining (Service Insertion)



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
=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
```

Snort is a free and open source IPS/IDS software.

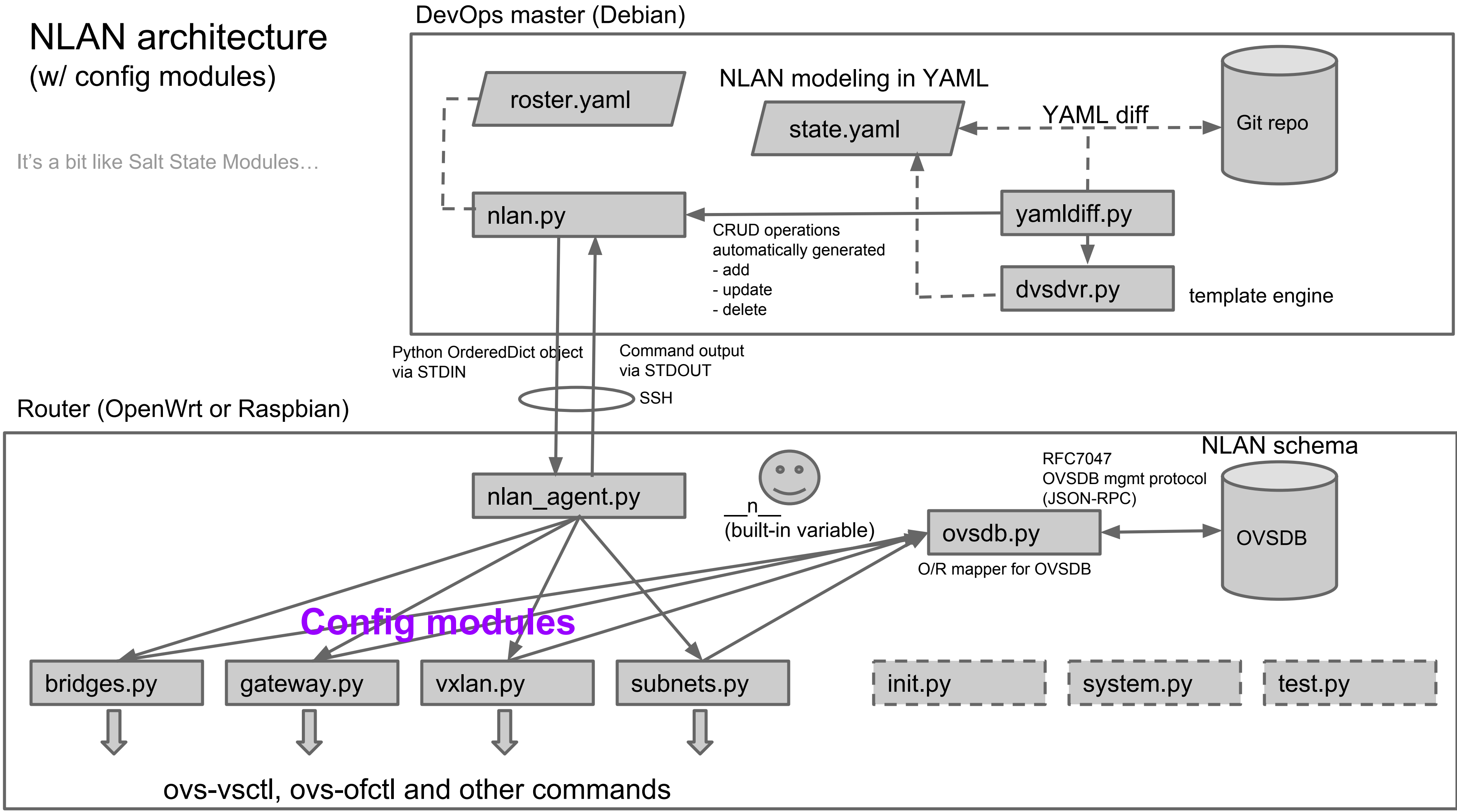
Home-made DevOps tool ‘NLAN’

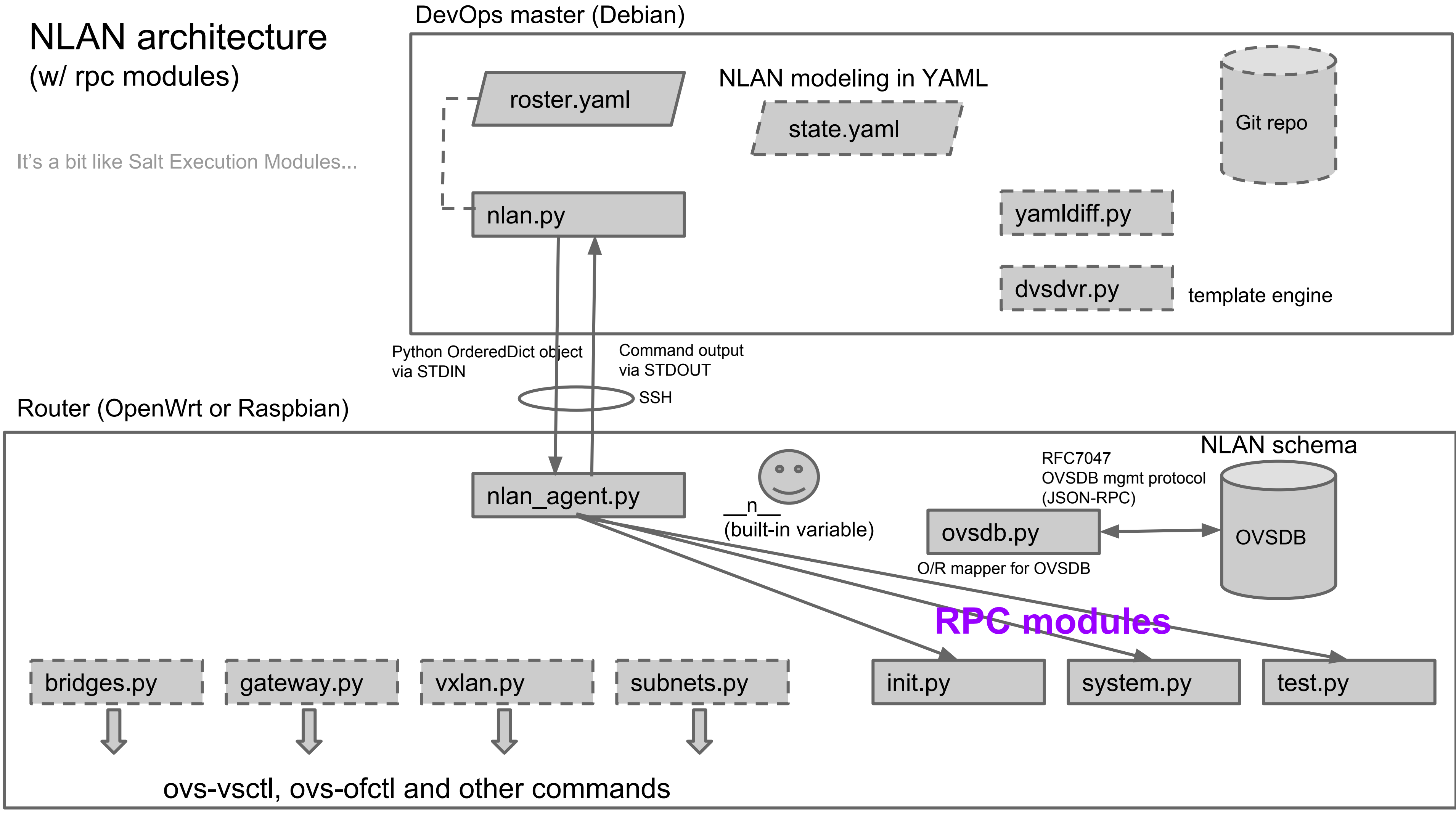
- 100% Python implementation
- Borrwed a lot of ideas from SaltStack (YAML-based state rendering) and OpenDaylight (MD-SAL)
  - Model-driven approach
  - YAML-based state rendering w/ a simple template engine
  - Imperative/declarative state rendering
- Works with OpenWrt with minimal Python
  - opkg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

Salt Stack is an open source DevOps framework and applications written in Python.
opkg is a light-weight package management software for OpenWrt.

NLAN architecture
(w/ config modules)

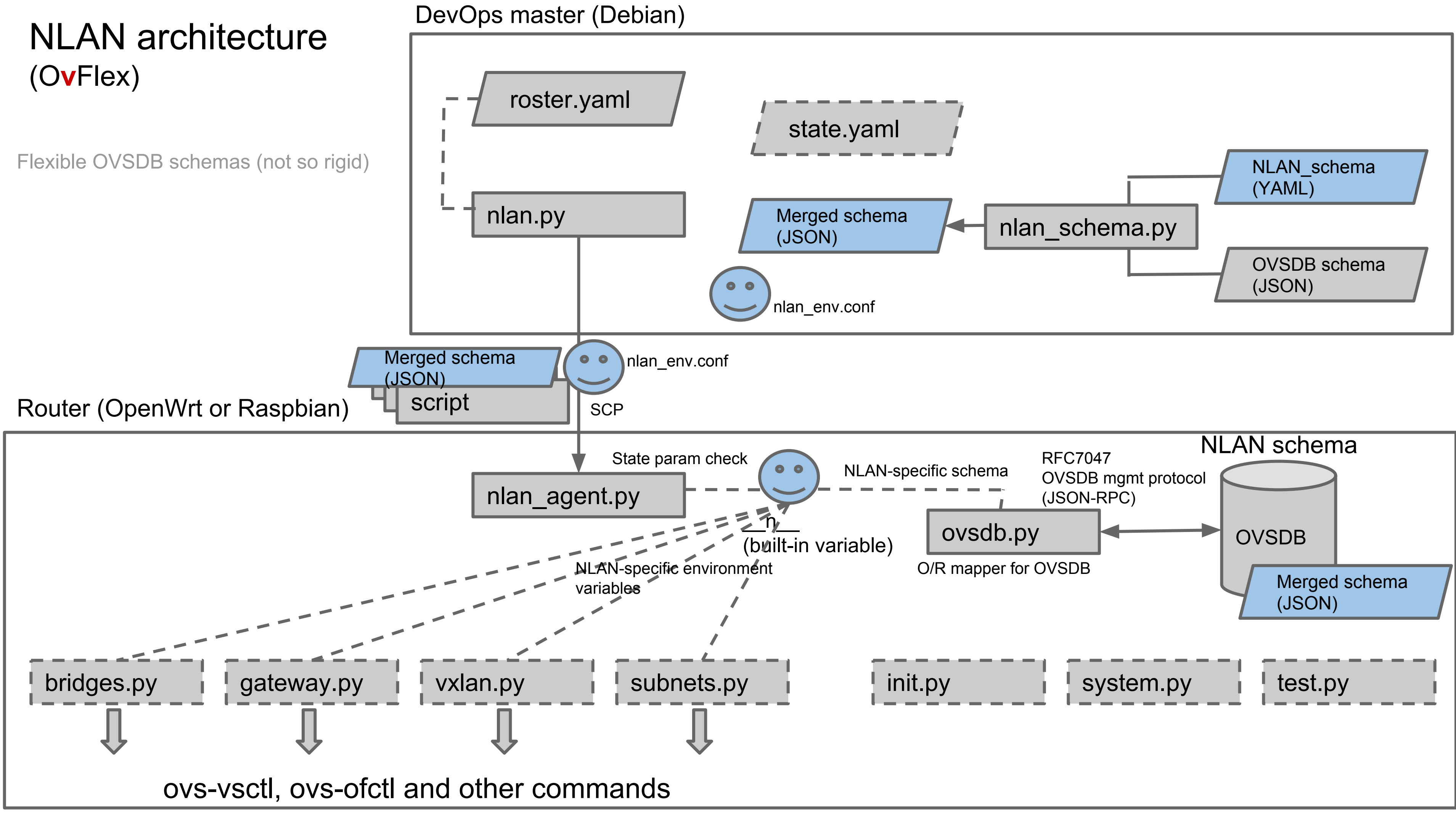
It's a bit like Salt State Modules…



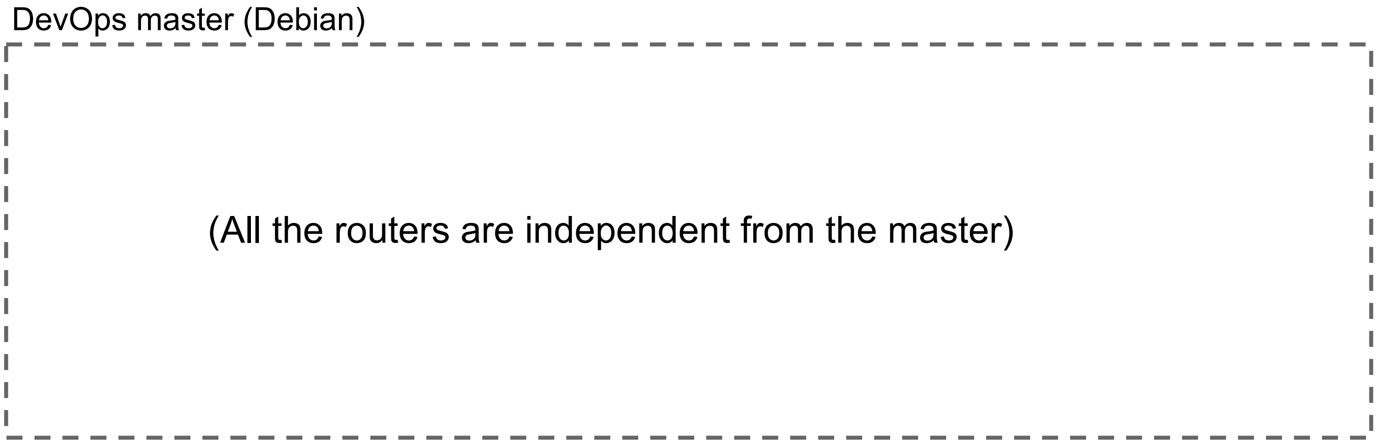

NLAN architecture
(w/ rpc modules)
It's a bit like Salt Execution Modules...
DevOps master (Debian)
roster.yaml
NLAN modeling in YAML
state.yaml
Git repo
nlan.py
yamldiff.py
dvsdvr.py
template engine
Python OrderedDict object
via STDIN
Command output
via STDOUT
SSH
Router (OpenWrt or Raspbian)
NLAN schema
RFC7047
OVSDB mgmt protocol
(JSON-RPC)
nlan_agent.py
__n__
(built-in variable)
ovsdb.py
OVSDB
O/R mapper for OVSDB
RPC modules
bridges.py
gateway.py
vxlan.py
subnets.py
init.py
system.py
test.py
ovs-vsctl, ovs-ofctl and other commands

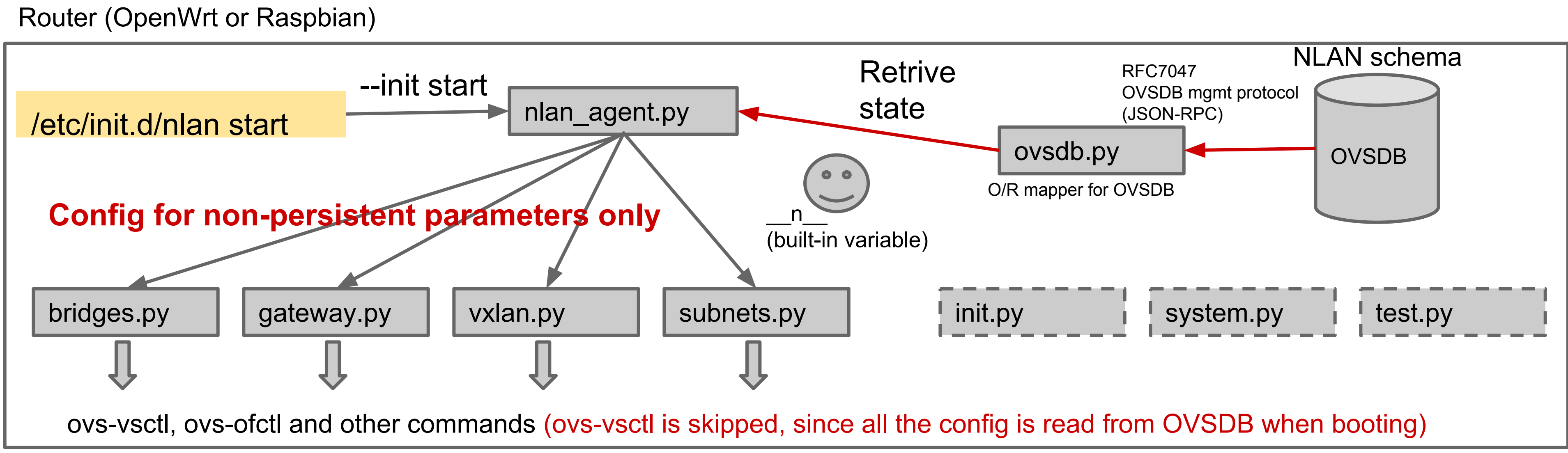
NLAN architecture
(OvFlex)

Flexible OVSDB schemas (not so rigid)



NLAN architecture
(System reboot)

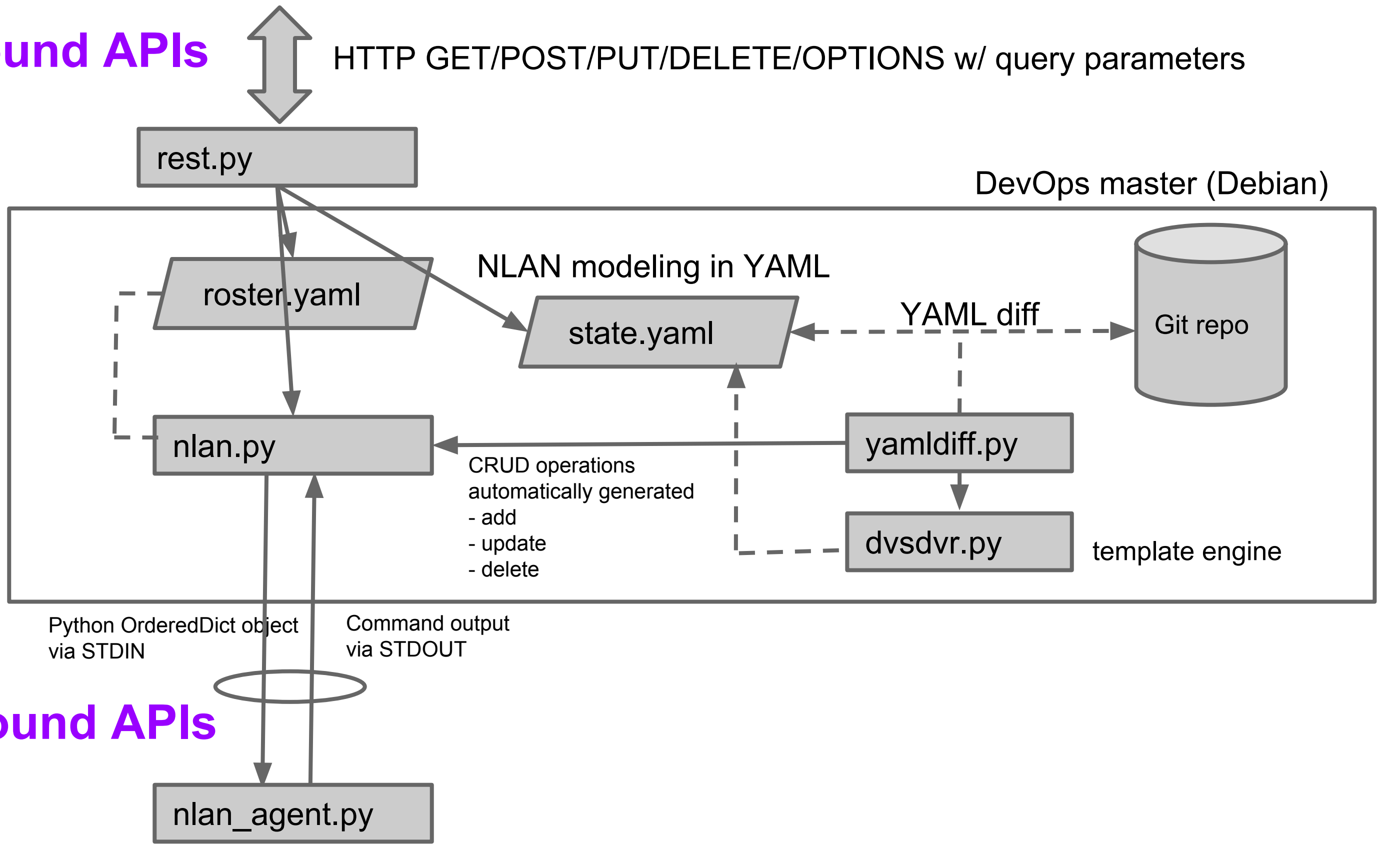




NLAN architecture
(REST APIs)

Northbound APIs

It's a bit like RESTCONF...



Southbound APIs

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)

NLAN schema in YAML

```
                                         :
NLAN_Subnet:
          columns:
          vni:
          type:
          key: {type: integer, minInteger: 0, maxInteger: 16777215}
          min: 1
          max: 1
          _description: "Virtual network identifier"
          vid:
          type:
          key: {type: integer, minInteger: 0, maxInteger: 4095}
          min: 0
          max: 1
          _description: "VLAN ID"
          ip_dvr:
          type:
          key: {type: string, enum: [set, [addr, mode, dhcp]]}
          value: {type: string, _pattern: {addr: ipv4_prefix, mode: dvr_mode, dhcp:
string}}
          min: 0
          max: 3
          _description: "Distributed Virtual Router setting"
          ip_vhost:
          type:
          key: {type: string, _pattern: ipv4_prefix}
          min: 0
          max: 1
          _description: "Virtual host in a linux network namespace"
          default_gw:
          type:
          key: {type: string, _pattern: ipv4_address}
          min: 0
          max: 1
          _description: "Default GW address for this subnet"
                                         :
```

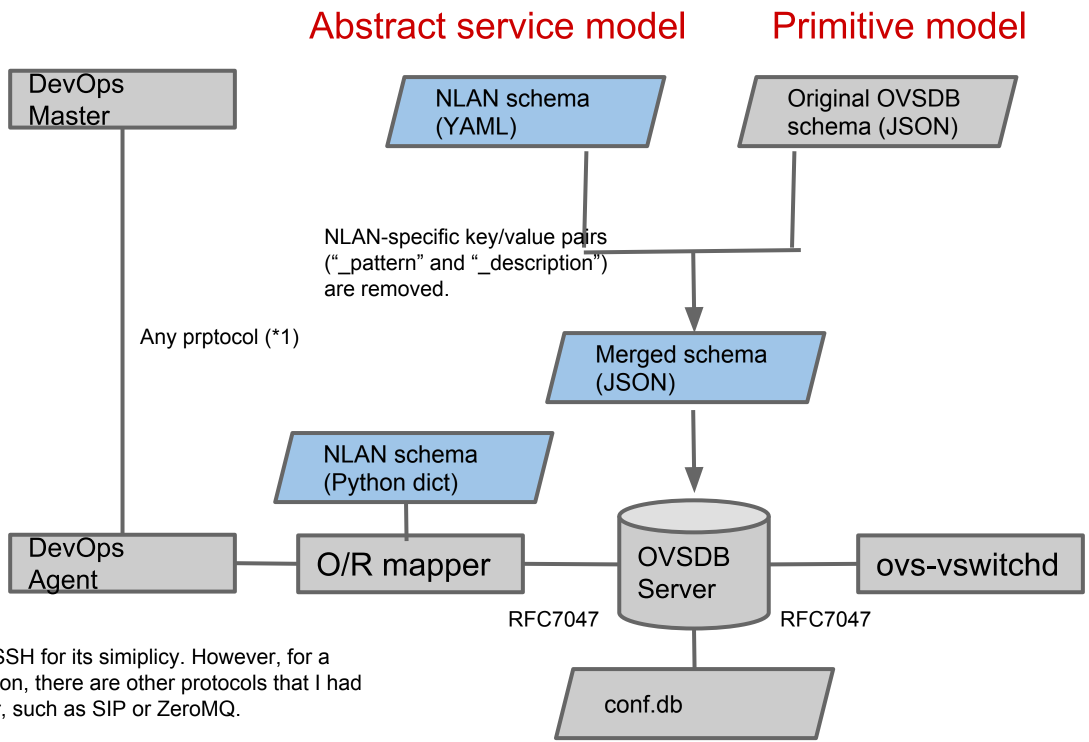
Original OVSDB schema in JSON

```
{"name": "Open_vSwitch",
 "version": "7.4.2",
 "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,
         "maxRows": 1},
                               :
```

Note: _pattern and _description are NLAN-specific OVSDB schema extensions

Merging schemas

OVSDB schema can also express more abstract data models.



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

NLAN states in OVSDB

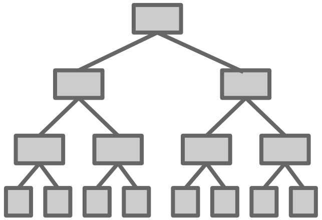
(ovsdb-client dump Open_vSwitch)

```
NLAN table
_uuid                                bridges                              gateway                              services  subnets
                                                                                                                    vxlan
------------------------------------ ------------------------------------ ------------------------------------ -------- ------------------------------------
------------------------------------------------------------------------------ ------------------------------------
0fc74b76-a3cb-40d7-9de5-26ea5d93e908 ae90946b-b00c-4e39-9ef0-2eda7ed89d66 dd3a47e2-b22c-45c6-a9f2-59f70f01eab7 []       [0ea1caea-57ab-4c29-913a-b916e903231
8, 6a607cd4-d0d0-4c7b-9b02-bf26c2acd343, abeeafb2-bedf-41f6-bf03-acc1ab773574] 4b0a1be2-1149-4665-943e-76088c20f271

NLAN_Bridges table
_uuid                                controller ovs_bridges
------------------------------------ ---------- -----------
ae90946b-b00c-4e39-9ef0-2eda7ed89d66 []         enabled

NLAN_Gateway table
_uuid                                network rip
------------------------------------ ------- -------
dd3a47e2-b22c-45c6-a9f2-59f70f01eab7 "eth2"  enabled

NLAN_Service table
_uuid chain name
----- ----- ----

NLAN_Subnet table
_uuid                                default_gw  ip_dvr                                        ip_vhost             peers                                ports v id vni
------------------------------------ ---------- -------------------------------------------- -------------------- ------------------------------------ ----- - -- ----
0ea1caea-57ab-4c29-913a-b916e9032318 []         {addr="192.168.100.1/24", mode=dvr}          "192.168.100.101/24" ["192.168.1.102", "192.168.1.103"] []    2   1
abeeafb2-bedf-41f6-bf03-acc1ab773574 []         {addr="10.0.1.1/24", dhcp=enabled, mode=hub} "10.0.1.101/24"      ["192.168.1.104"]                    []    1   1001
6a607cd4-d0d0-4c7b-9b02-bf26c2acd343 []         {addr="10.0.3.1/24", dhcp=enabled, mode=dvr} "10.0.3.101/24"      ["192.168.1.102", "192.168.1.103"] []    3   103

NLAN_VXLAN table
_uuid                                local_ip        remote_ips
------------------------------------ --------------- ---------------------------------------------------
4b0a1be2-1149-4665-943e-76088c20f271 "192.168.1.101" ["192.168.1.102", "192.168.1.103", "192.168.1.104"]
```

Model-Driven Service Abstraction (1/2)

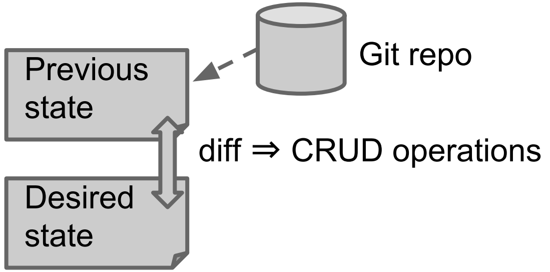
Option (b): Config in YAML

OVSDB schema



YAML-based config file

template engine



Step1: define network service model by using OVSDB schema language.

Option (a): use REST APIs and CLIs

Deploy the new schema to the network ⇒ REST APIs, CLIs and internal APIs for CRUD operations are automatically generated.

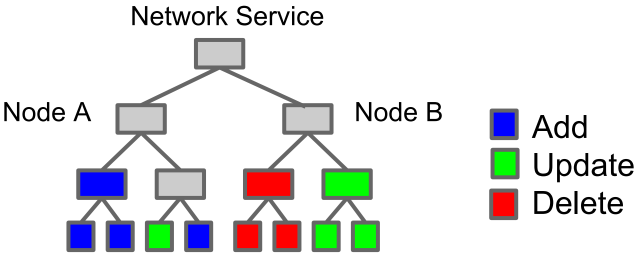
It's a Python-based Model-Driven Service Abstraction: you don't need to compile the code to generate APIs.

Step2: write the config as "desired state" in YAML format w/ some placeholders for a template engine

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

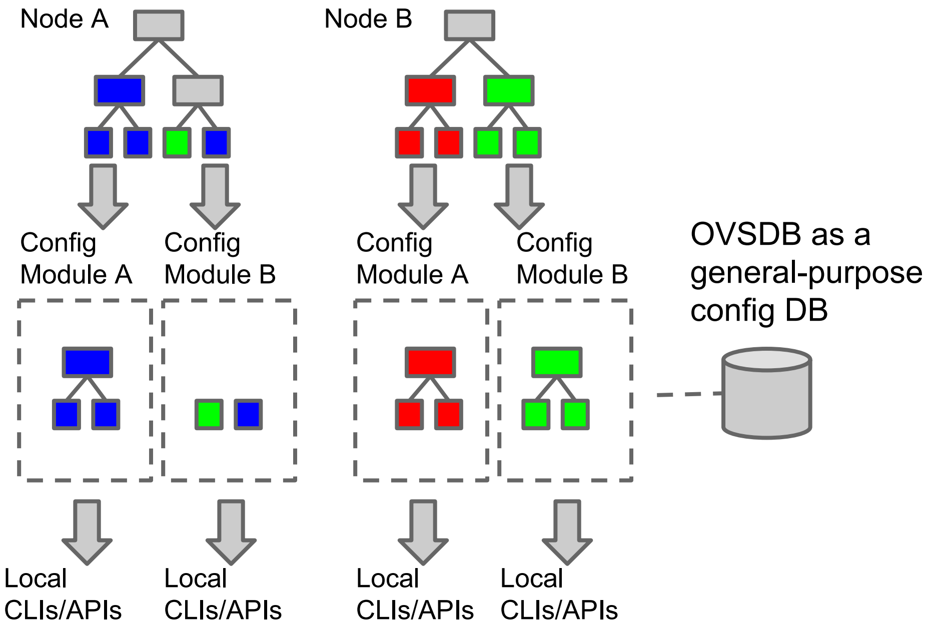
Step4: NLAN-Master (nlan.py and yamldiff.py) generates CRUD operations comparing the desired state with the previous state

Model-Driven Service Abstraction (2/2)



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

Step6: NLAN-Agent (nlan_agent.py) 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: {addr: '10.0.1.1/24'. mode: dvr}
      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

```
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: {addr: '10.0.1.1/24', mode: dvr}
      ip_vhost: '10.0.1.101/24'
      ports:
      - eth0.1
      peers: [‘192.168.1.102’, ‘192.168.1.104’]
      :
```

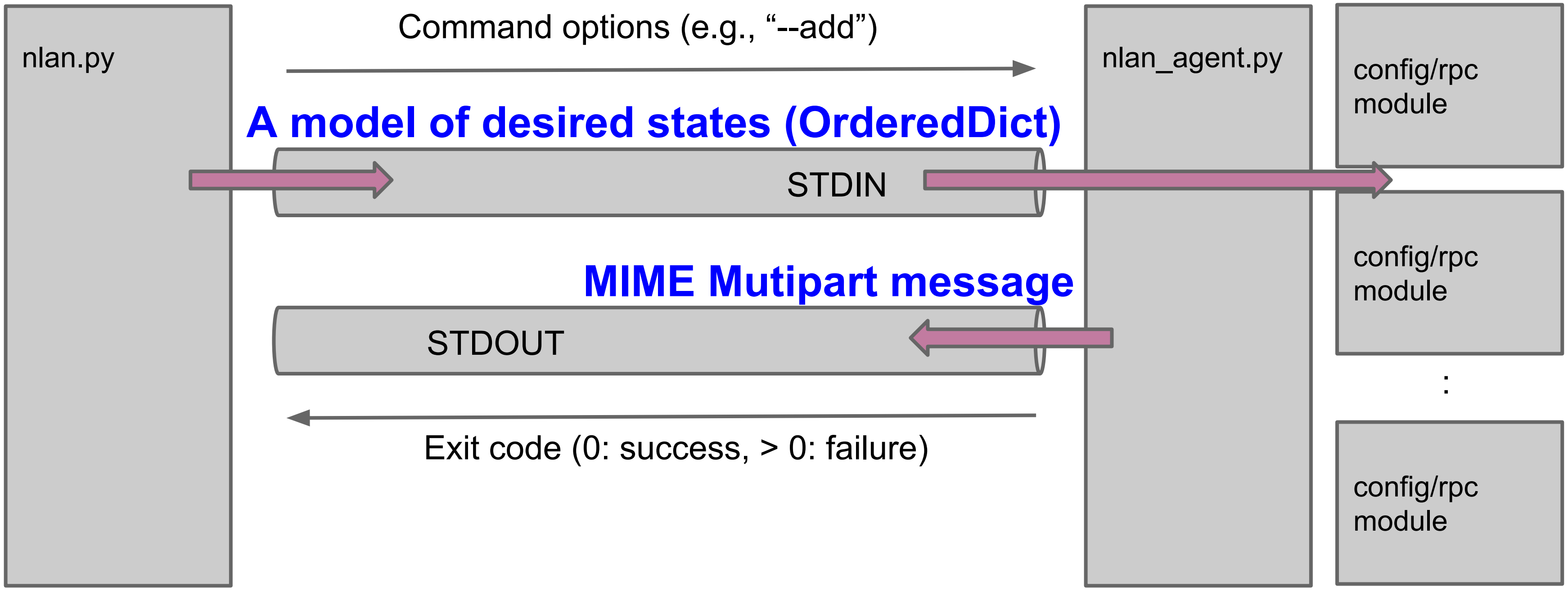
Desired state in Python OrderedDict

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

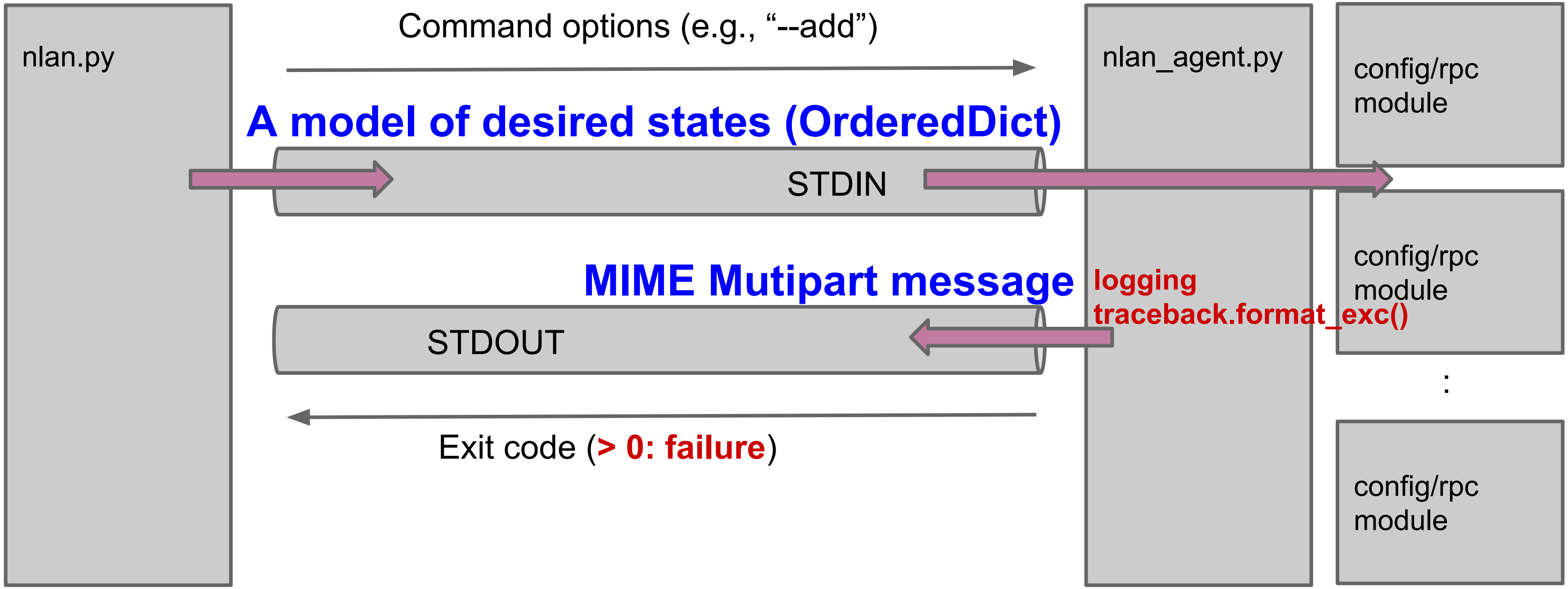
"OrderedDict([('bridges', {'ovs_bridges': 'enabled'}), ('gateway', {'network': 'eth2', 'rip': 'enabled'}), ('vxlan', {'remote_ips': ['192.168.1.103', '192.168.1.102', '192.168.1.104'], 'local_ip': '192.168.1.101'}), ('subnets', [{'peers': ['192.168.1.102', '192.168.1.103'], 'vid': 2, '_index': ['vni', 1], 'ip_vhost': '192.168.100.101/24', 'vni': 1, 'ip_dvr': OrderedDict([('addr', '192.168.100.1/24'), ('mode', 'dvr')])}, {'peers': ['192.168.1.102', '192.168.1.103'], 'vid': 3, '_index': ['vni', 103], 'ip_vhost': '10.0.3.101/24', 'vni': 103, 'ip_dvr': OrderedDict([('addr', '10.0.3.1/24'), ('mode', 'dvr')])}, {'peers': ['192.168.1.104'], 'vid': 1, '_index': ['vni', 1001], 'ip_vhost': '10.0.1.101/24', 'vni': 1001, 'ip_dvr': OrderedDict([('addr', '10.0.1.1/24'), ('mode', 'hub')])}])])"

- Imperative/declarative state representation.
- I don't use JSON, since NLAN is 100% Python implementation.

NLAN Request/Response over SSH



Exception handling and debugging



Trouble shooting becomes much easier by sending traceback.format_exc() and raw command outputs (test/plain) to nlan.py.

MIME Multipart example

```
root@debian:~# nlan -t openwrt1 --get bridges ovs_bridges -M
*** Response from router:openwrt1,platform:openwrt
From nobody Thu May 29 03:16:42 2014
MIME-Version: 1.0
Content-Type: multipart/mixed;boundary="NKwDX7NAzowkZDZymawAVVwY3TcnvS1D"

--NKwDX7NAzowkZDZymawAVVwY3TcnvS1D
Content-Type: test/plain
Content-Description: Default out
X-NLAN-Type: default_out


--NKwDX7NAzowkZDZymawAVVwY3TcnvS1D
Content-Type: application/x-nlan
Content-Description: CRUD get
X-NLAN-Type: crud_response

OrderedDict([('bridges', {'ovs_bridges': 'enabled'})])
--NKwDX7NAzowkZDZymawAVVwY3TcnvS1D
Content-Type: application/x-nlan
Content-Description: NLAN Response
X-NLAN-Type: nlan_response

OrderedDict([('message', 'Execution completed'), ('exit', 0)])
--NKwDX7NAzowkZDZymawAVVwY3TcnvS1D--
```

Just use Python’s “email” package for parsing the message.

Showing exception from remote routers

```
root@debian:~# nlan -t openwrt1 --add bridges ovs_bridges=enabled
*** Response from router:openwrt1,platform:openwrt
exception: CmdError
message: Command execution error
traceback: Traceback (most recent call last):
  File "/opt/nlan/nlan_agent.py", line 158, in _route
        call()
  File "/opt/nlan/config/bridges.py", line 35, in add
        cmdp('ovs-vsctl add-br br-int')
  File "/opt/nlan/cmdutil.py", line 115, in check_cmdp
        return _cmd('check_call', True, *args)
  File "/opt/nlan/cmdutil.py", line 20, in _cmd
        return _cmd2(check=check, persist=persist, args=cmd_args)
  File "/opt/nlan/cmdutil.py", line 62, in _cmd2
        raise CmdError(argstring, e.returncode, out)
CmdError: Command execution error

command: ovs-vsctl add-br br-int
stdout: None
exit: 1
operation: add
progress: [('bridges', False)]
```

traceback.format_exc()

Showing debug information from remote routers

```
--37axII0rq4QKTUh6Ll2jpcwxt3HC3Vm2
Content-Type: text/plain
Content-Description: [DEBUG] 2014-05-29 10:23:50, 057 module:cmdutil,
 function:log, router:openwrt3
X-NLAN-Type: logger

cmd: ip netns exec ns1 ip route add default via 192.168.100.3 dev eth0
--37axII0rq4QKTUh6Ll2jpcwxt3HC3Vm2
Content-Type: text/plain
Content-Description: [DEBUG] 2014-05-29 10:23:50, 060 module:cmdutil,
 function:log, router:openwrt3
X-NLAN-Type: logger

cmd: ip addr add dev int_dvr1 192.168.100.3/24
--37axII0rq4QKTUh6Ll2jpcwxt3HC3Vm2
Content-Type: text/plain
Content-Description: [DEBUG] 2014-05-29 10:23:50, 065 module:cmdutil,
 function:log, router:openwrt3
X-NLAN-Type: logger

cmd: route
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref  Use Iface
10.0.2.0         *                255.255.255.0   U        0       0        0 eth0
192.168.1.0      *                255.255.255.0   U        0       0        0 eth2
192.168.56.0     *                255.255.255.0   U        0       0        0 eth1
192.168.100.0     *               255.255.255.0   U        0       0        0 int_dvr1

--37axII0rq4QKTUh6Ll2jpcwxt3HC3Vm2
```

NLAN’s logger can print every command executed at remote routers.

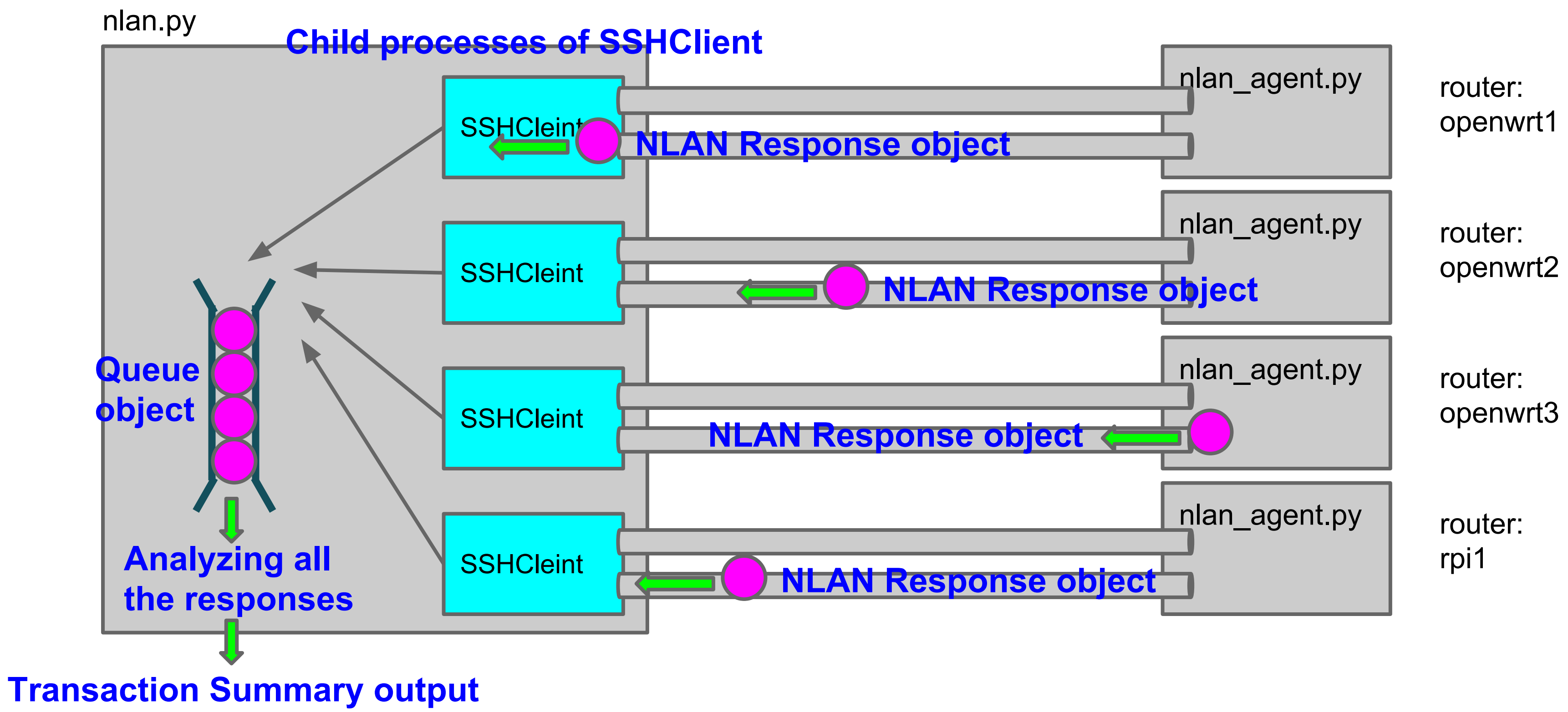
Showing transactions with OVSDB at remote routers

```
[DEBUG] 2014-05-29 03:45:53,322 module:ovsdb,function:crud,router:rpi1
CRUD operation (add): {'peers': ['192.168.1.102', '192.168.1.103'], 'ports': ['mz.101'], 'vni': 101, 'vid': 1}
[DEBUG] 2014-05-29 03:45:53,323 module:ovsdb,function:_send,router:rpi1
```


```
request:
  id: 89846
  method: transact
  params:
  - Open_vSwitch
  - op: insert
          row:
          peers:
          - set
          - - 192.168.1.102
          - 192.168.1.103
          ports:
          - set
          - - mz.101
          vid: 1
          vni: 101
          table: NLAN_Subnet
          uuid-name: temp_515801
  - mutations:
          - - subnets
          - insert
          - - set
          - - - named-uuid
          - temp_515801
          op: mutate
          table: NLAN
          where: []
response:
  error: null
  id: 89846
  result:
  - uuid:
          - uuid
          - cb6bc9a6-0a86-415f-a82c-19f6a5e841d0

  - count: 1
```


NLAN can show RFC7047 JSON-RPC transactions in the form of YAML.

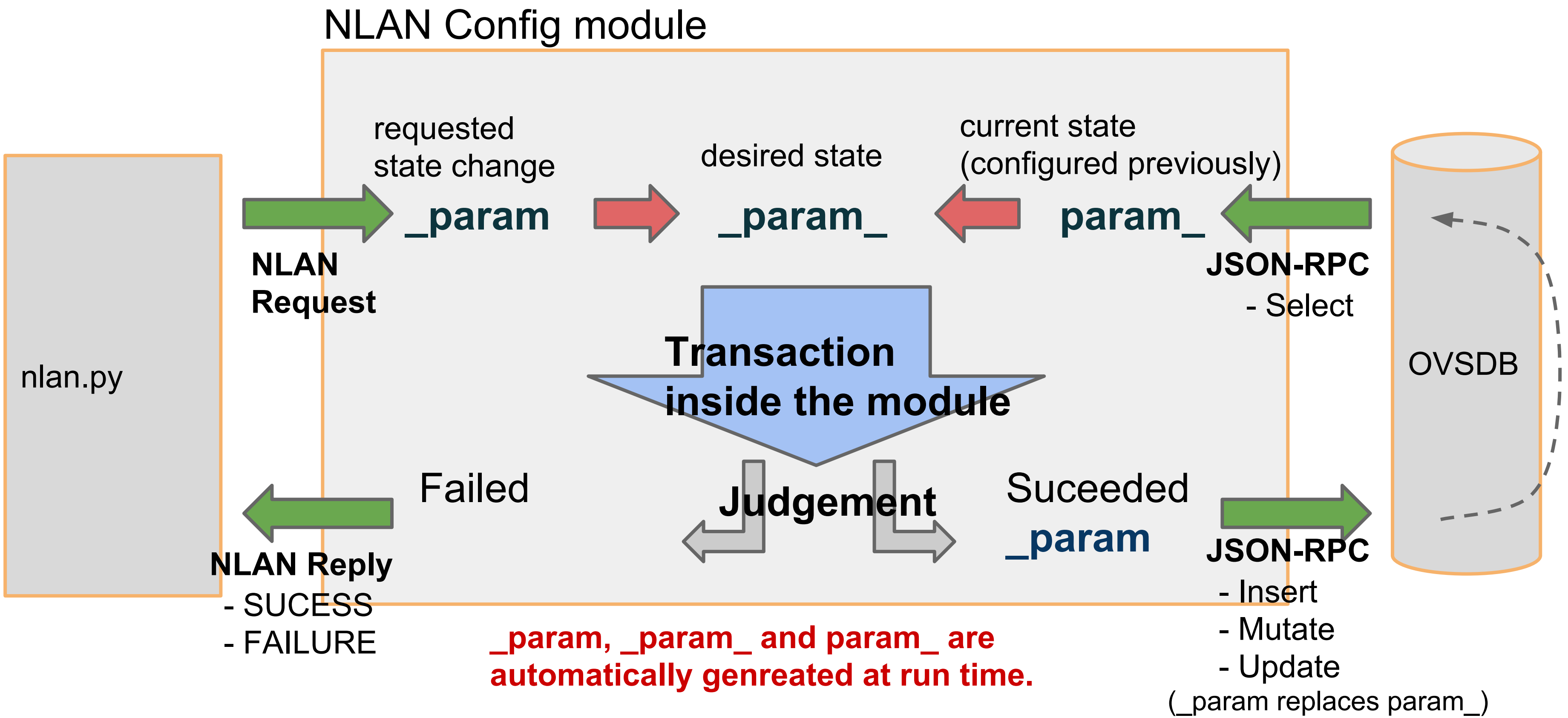
Parallel SSH sessions



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)
```

CRUD operations inside NLAN config modules



_param, _param_ and param_ are automatically genreated at run time.

Global variables (__dict__) generated by CRUD.params()

model

```
state -+- param a
       |
       +- param b
       |
       +- param c
       |
       +- param d
```

State parameters (for add/update operations)

			Global variables generated by Model.params()			
State params	Requested by Master	OVSDB	_param (Requested change)	_param_ (Desired state)	param_ (Current state in OVSDB)	Operation
a	1	None	_a=1	_a_=1	a_=None	add
b	2	1	_b=2	_b_=2	b_=1	update
c	None	1	_c=None	_c_=1	c_=1	
d	None	None	_d=None	_d_=None	d_=None	

State parameters (for delete operations)

			Global variables generated by Model.params()			
State params	Requested by Master	OVSDB	_param (Requested change)	_param_ (Desired state)	param_ (Current state in OVSDB)	Operation
a	1	None	_a=1	_a_=***	a_=None	(Never exists)
b	1	1	_b=1	_b_=None	b_=1	delete
c	None	1	_c=None	_c_=1	c_=1	
d	None	None	_d=None	_d_=None	d_=None	

Python conding in NLAN Config modules (1/2)

nlan_agent.py

```
with oputil.CRUD(...):
        module.add()
```

Python context manager

- to generate _param, _param_ and param_ at run time
- to save the state modification to OVSDB at the end of execution

```
with oputil.CRUD(...):
        module.delete()
```

```
with oputil.CRUD(...):
        module.update()
```

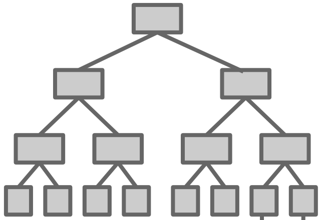
module.py

```
def add():
        if _param1:
                (execute local commands w/ _param1 and other
        _param_(s) as arguments)
        if _param2:
                (execute local commands w/ _param2 and other
        _param_(s) as arguments)
        if _param3:
                (execute local commands w/ _param3 and other
        _param_(s) as arguments)

def delete():
                :

def update():
                :
```

Every NLAN config module MUST implement add(), delete() and update() functions.

Python conding in NLAN Config modules (2/2)

Model defined in OVSDB schema
(converted into Python dictionary)

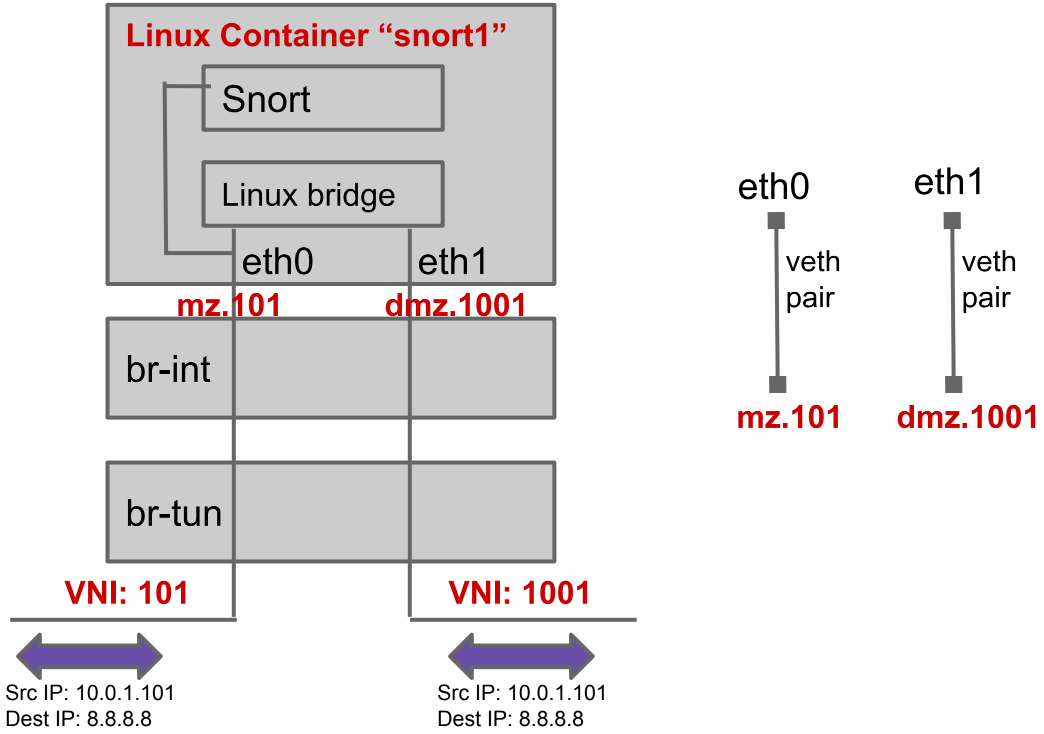


```
def add():
    if _param1:
        (execute local commands w/ _param1 and other _param_(s) as arguments)
    if _param2:
        (execute local commands w/ _param2 and other _param_(s) as arguments)
```

It’s a bit like OpenFlow flow entries in a table…: match ⇒ action

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 command usage (nlan.py)

Copy NLAN-agent-side modules to all the target routers (incl. NLAN/OVSDB schema and Linux init.d scripts):

$ nlan.py --scpmod

Initialize states at all the target routers:

$ nlan.py init.run

Ask all the target routers to transit to the desired states

$ nlan.py -G deploy

Rollback to the previous config

$ nlan.py init.run

$ nlan.py -R deploy

Command line CRUD (add/get/update/delete) operations

$ nlan.py -t openwrt1 --add subnets _index=101 vid=1 ip_dvr=mode:dvr,addr:10.0.1.1/24 ip_vhost=10.0.1.101/24

$ nlan.py -t openwrt1 --update subnets _index=101 ip_vhost=10.0.1.109/24

$ nlan.py -t openwrt1 --delete subnets _index=101 ip_vhost=10.0.1.109/24

$ nlan.py -t openwrt1 --get subnets _index=101 vid ip_dvr

CLIs for CRUD operations are automatically generated from the model at run time.

Reboot all the target routers:

$ nlan.py system.reboot

NLAN command usage (nlan_agent.py)

Initialize states:

$ nlan_agent.py init.run

Command line CRUD (add/get/update/delete) operations

$ nlan_agent.py --add subnets _index=101 vid=1 ip_dvr=mode:dvr,addr:10.0.1.1/24 ip_vhost=10.0.1.101/24

$ nlan_agent.py --update subnets _index=101 ip_vhost=10.0.1.109/24

$ nlan_agent.py --delete subnets _index=101 ip_vhost=10.0.1.109/24

$ nlan_agent.py --get subnets _index=101 vid ip_dvr

CLIs for CRUD operations are automatically generated from the model at run time.

Reboot all the target routers:

$ nlan.py system.reboot

REST APIs
(automatically generated from the model at run time)

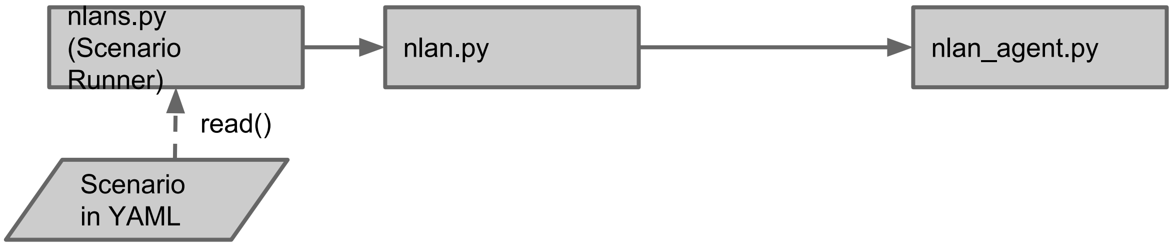
HTTP Method	NLAN CRUD operations	NLAN RPC operations
GET	CRUD: get URL: /<router>/config/<module>/<_index>	URL: /<router>/rpc/<module>/<func>
POST	CRUD: add URL: /<router>/config/<module>/<_index>	-
PUT	CRUD: update URL: /<router>/config/<module>/<_index>	-
DELETE	CRUD: delete URL: /<router>/config/<module>/<_index>	-
OPTIONS	Get NLAN schemas URL: none	-

REST APIs example

HTTP Method	URL	Query parameters
POST	/_ALL/rpc/test/echo	params=Hello!
OPTIONS	(none)	params=subnets
POST	/openwrt1/rpc/init/run	(none)
POST	/openwrt1/config/bridges	ovs_bridges=enabled
POST	/openwrt1/config/vxlan	local_ip=192.168.1.101&remote_ips=192.168.1.102,192.168.56.103
PUT	/openwrt1/config/vxlan	remote_ips=192.168.1.102,192.168.56.104
GET	/openwrt1/config/vxlan	params=remote_ips
POST	/openwrt1/config/subnets/101	vni=101&vid=1&ip_dvr=addr:10.0.1.1/24,mode:dvr
DELETE	/openwrt1/config/subnets/101	params=ip_dvr
POST	/openwrt1/rpc/db/state	(none)

Scenario Runner

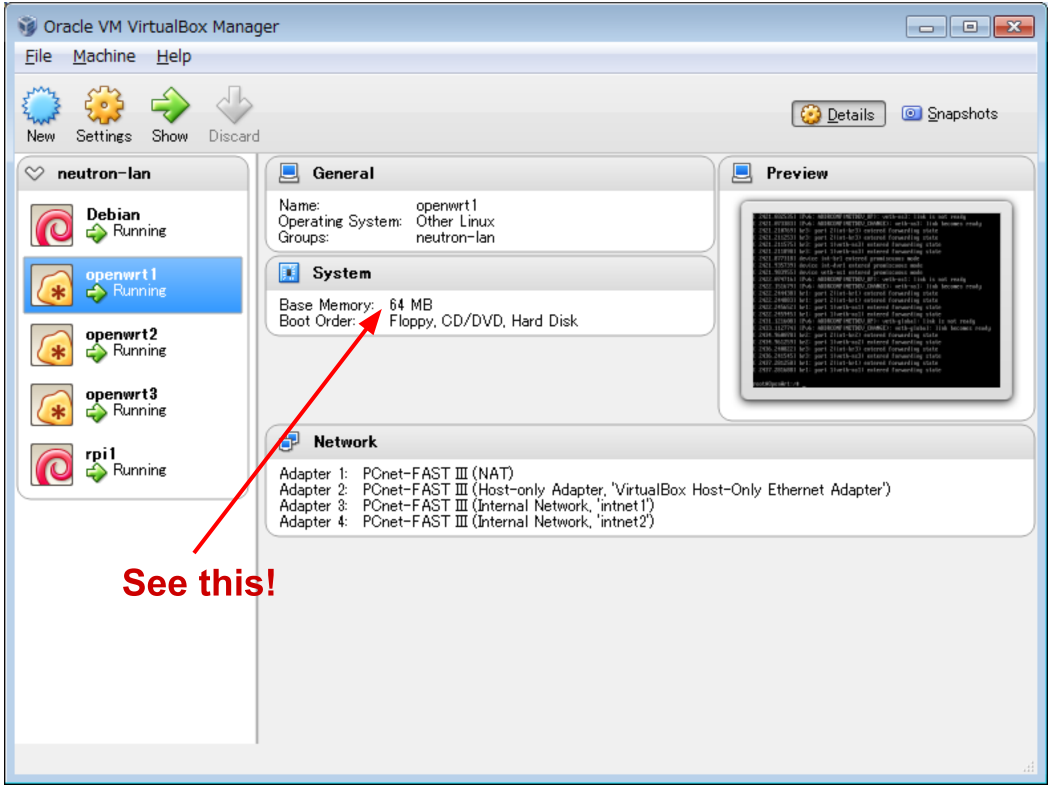
- nlans.py -- reads test scenarios and executes each test
- Test scenarios written in YAML
- Automatic test result confirmation
  - Inspired by Python's "unittest"
    - "assert"
    - "asserRaises"
    - ("assertOutputs" to be supported)



Simple RPC library

```
import rpc

# calls test.kwargs_test(...) at router ‘openwrt1’
rpc = rpc.RPC(module='test', func='kwargs_test', target='openwrt1')
result = rpc(1, b='Hello', d='World!')
print result

# calls test.echo(*args) at all routers on the roster
rpc = rpc.RPC(module='test', func='echo')
results = rpc('Hello World!')
for l in results:
    print l['router'], l['stdout']
```

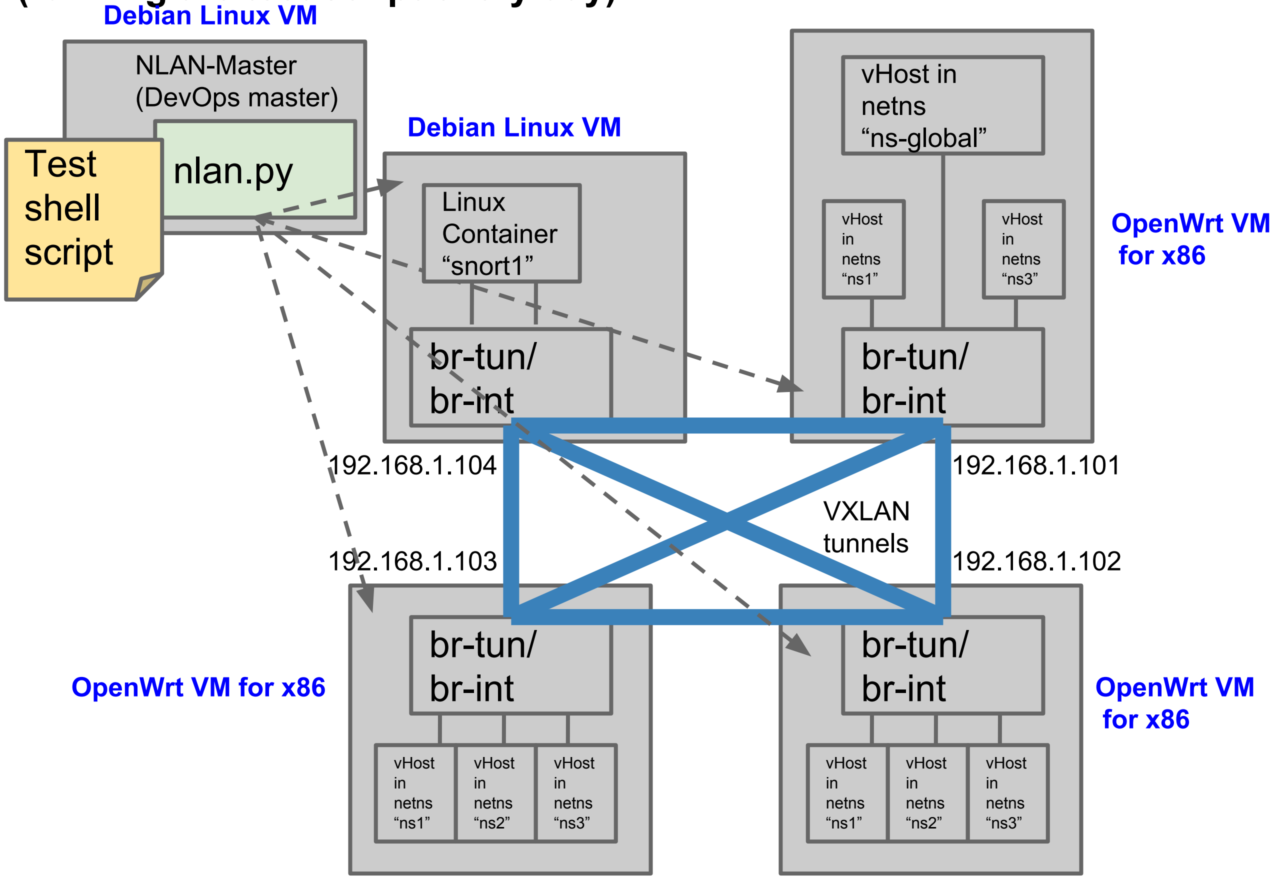
NLAN Software Development environment on VirtualBox



- 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 ⇒ An alternative to mininet 2.0.0
- Network adapters setting
  - Internet access: “NAT”
  - Management: “Host-Only”
  - NLAN underlay: “Internal”

Integration Test environment on VirtualBox

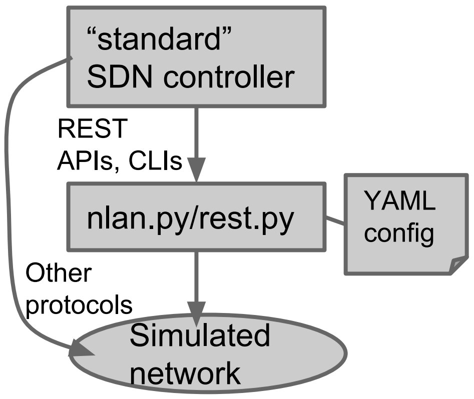
(running the test script every day)



- 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"

NLAN use cases

- SDN for the LAN
- Network simulator
    - mininet ⇒ OpenFlow-focused
    - NLAN ⇒ Can also simulate legacy networks (OpenWrt VM for x86 CPU)
        - Physical links (VXLAN)
        - Pseudo-wire, VPLS/VPWS (VXLAN+VNI/VID)
        - Nested networks: underlay (VXLAN) and overlay (VXLAN)
        - Routing protocols: quagga (RIP, OSPF, BGP etc)
        - OpenFlow: open vswitch
        - Config persistency: OVSDB
        - nlan as "proprietary SDN controller" supporing REST APIs and CLIs: more "standard" SDN controller controls nlan via those APIs.



Future work

- NLAN enhancements
    - NLAN Agent in a Linux container (lxc-excute)
    - Multi-generation config rollback
    - Rollback in an error condition
    - Full-automatic integration testing