VLCP Documentation

Release 1.3.0

Hu Bo

Aug 10, 2018

Contents

VLCP is a modern SDN controller able to be integrated with OpenStack, Docker and other virtualization environments. It is designed to be highly scalable, highly available and have very low overhead for virtual networking. Currently it is ready for production, and has been verified, tested and in use in clusters with about 10 physical servers. Tests show that the controller stays stable for more than a week under high pressure as: 1000 endpoints per server; 16+ Gbps traffic; 200 endpoint changes (creations and deletions) per minute per server.

CHAPTER 1

Introduce

1.1 Why VLCP

Because VLCP is **exactly what you need for your modern SDN network**

1.1.1 Functions

VLCP provides the ability to create both L2 and L3 SDN networks. All the elements in the SDN network like physical networks (infrastructures), logical networks and endpoints can be modified at any time and take an immediate effect. Logical networks are fully isolated with each other, including the abilities to use the same MAC addresses or IP addresses. It is very easy to create multi-tenant networks with VLCP controllers. VLCP supports both VLAN and VXLAN for isolation; it is even possible to use them for different logical networks at the same time.

VLCP provides easy-to-use web APIs for configurations. The APIs can be used anywhere with a connection to the central database, and multiple instances can be deployed to provide load balancing or high-availabilities.

For VXLAN, VLCP supports software implementation on OpenvSwitch, and hardware implementation with hardware_vtep interface on physical switches (the same interface used by NSX). Software VXLAN implementation provides about 6+Gbps for each server. With physical switches supporting hardware_vtep, it is usually 20+Gbps.

1.1.2 Stabilities

VLCP has a modern software architecture. It is designed to be stable under the worst situations. Usually the controller is deployed on each server, working independently. Server failures only affects traffic to and from this server. As long as the servers containing the source endpoint and the destination endpoint stay alive, network traffic between these two endpoints is not affected.

VLCP uses a ZooKeeper cluster for configuration management. ZooKeeper provides consistency for all the nodes easily. All nodes are equal to each other when reading from and writing to the central storage. They use a transaction layer to provide ACID on multiple keys, so any change to the central storage either success or fail at once. Nodes use the Watch mechanism of ZooKeeper to subscribe and update informations related to the local endpoints. There is not any middle-states, any critical failures like power failures, system core dumps, network disconnections are recoverable.

The hardest recover operations of VLCP controllers are no more than restarting the controller. Usually it recovers as soon as the network/system problems are solved.

There are multiple guarantees for the SDN network connectivities:

1. Partial failures (less than half of the servers) on the ZooKeeper cluster do not affect any operations
2. A full failure on the ZooKeeper cluster makes it impossible to create/delete/modify endpoints, but the existed endpoints are not affected.
3. Controller crashes on one server makes it stop responding to network structure changes (endpoint creation/deletion etc.), but the existed endpoints are not affected.
4. OpenvSwitch crashes, server crashes disconnect the endpoints on this server with other endpoints, but connectivities between endpoints on other servers are not affected.
5. Failures are always recoverable. No components would stay in a middle-state.

With these guarantees, any disasters can be keep in the smallest scope to reduce the impact to the production environment.

1.1.3 Highly Extensible

VLCP is designed to be both a production-ready SDN controller and an extensible SDN framework. Most functions in the SDN networks are split into smaller modules, each provides an independent function. Every module can be loaded, unloaded or reloaded even without a restarting. You can rearrange the modules to add or remove functions. It is also possible to develop new functions with separated modules, and integrate them with the SDN controller.

1.2 More Figures, Less Text

These are figures used in other parts of this document. Is there any interesting topic for you?

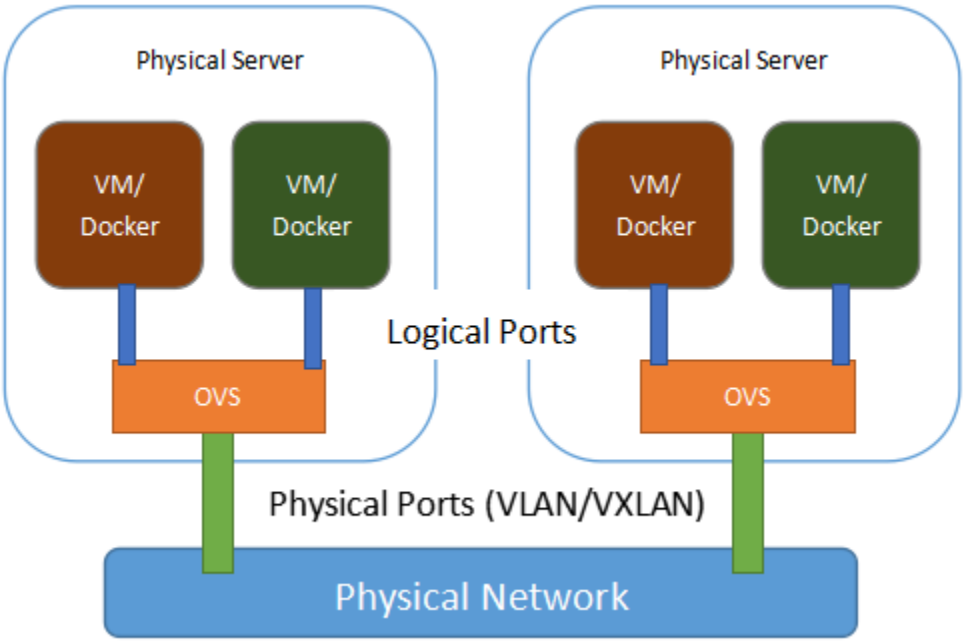

Fig. 1: Physical View of the Network

This is the physical view of the SDN network. VMs or containers are in different servers, but we want to organize them in a logical way. See *Physical View of the Network*.

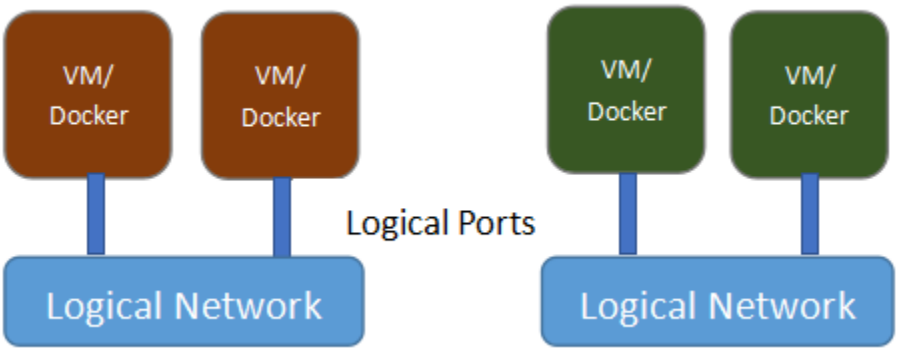

Fig. 2: Logical View of the Network

This is the logical view of the same network. VMs or containers are organized by their logical positions; their physical positions do not matter. See *Logical View of the Network*.

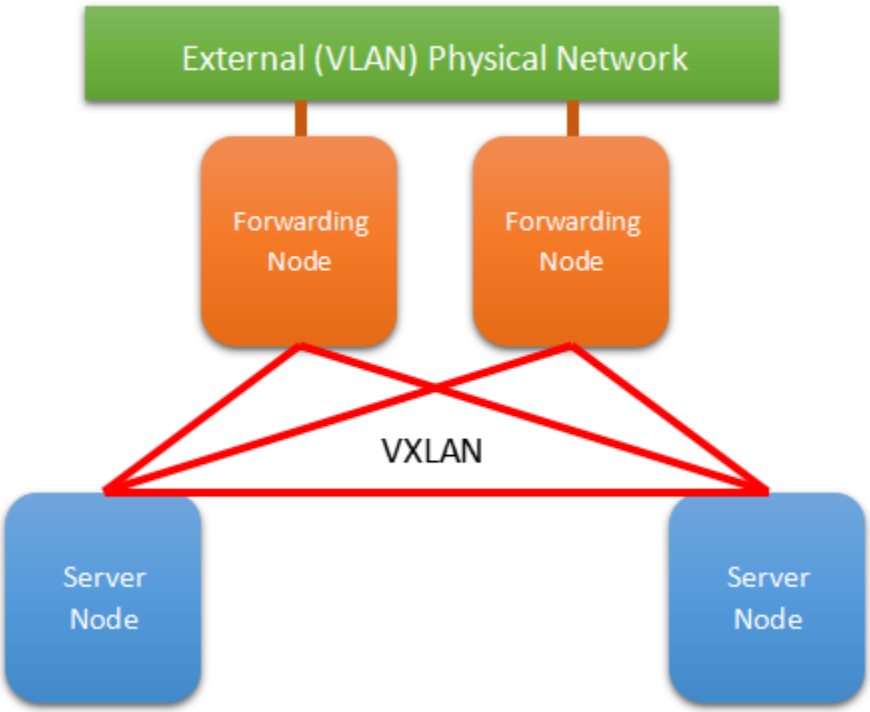

Fig. 3: L3 Network with Forwarding Nodes

This is a network with forwarding nodes. A forwarding node forwards traffic between SDN networks and traditional networks. In VLCP, the forwarding nodes are just normal SDN controllers with a few configurations turned on. See *L3 Network with Forwarding Nodes*.

This is how we configure our SDN network. See *Network Settings Structure*.

This is the technology stack of VLCP - how this software is built. See *Technology Stack of VLCP*.

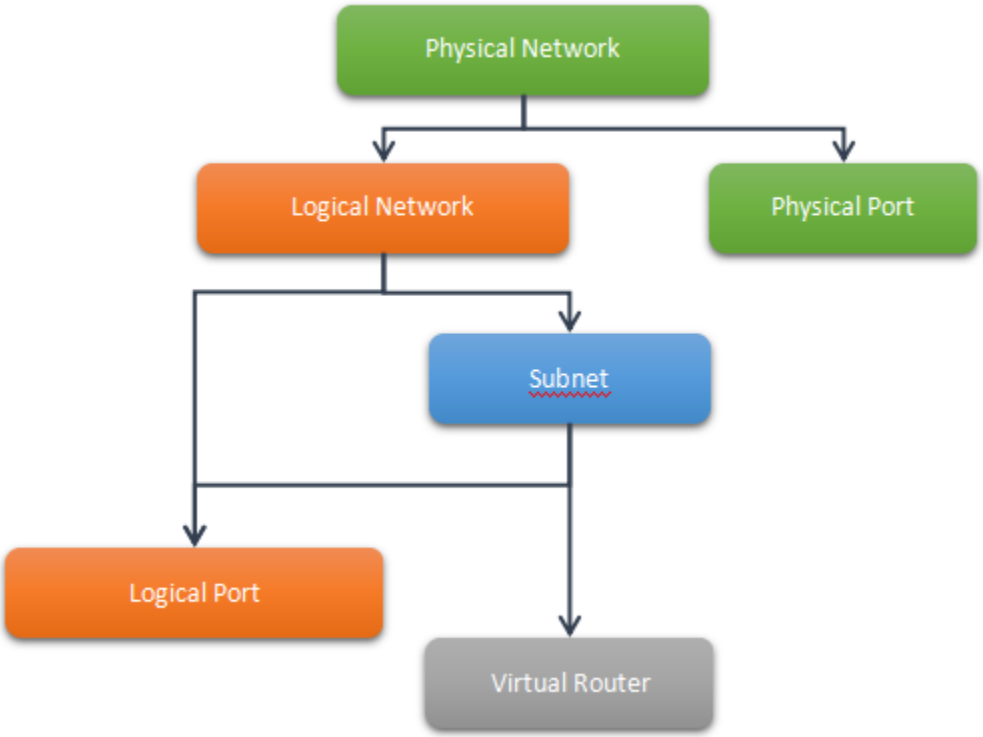

Fig. 4: Network Settings Structure

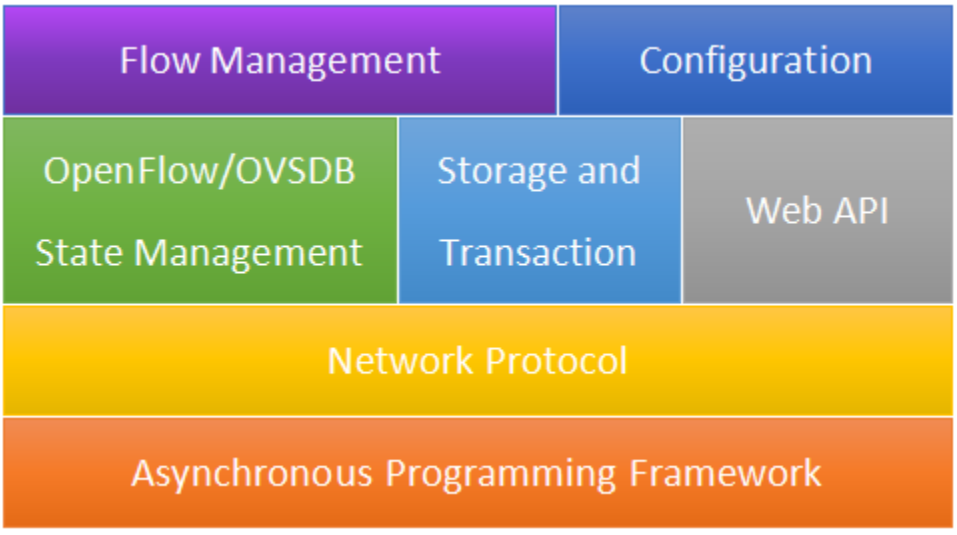

Fig. 5: Technology Stack of VLCP

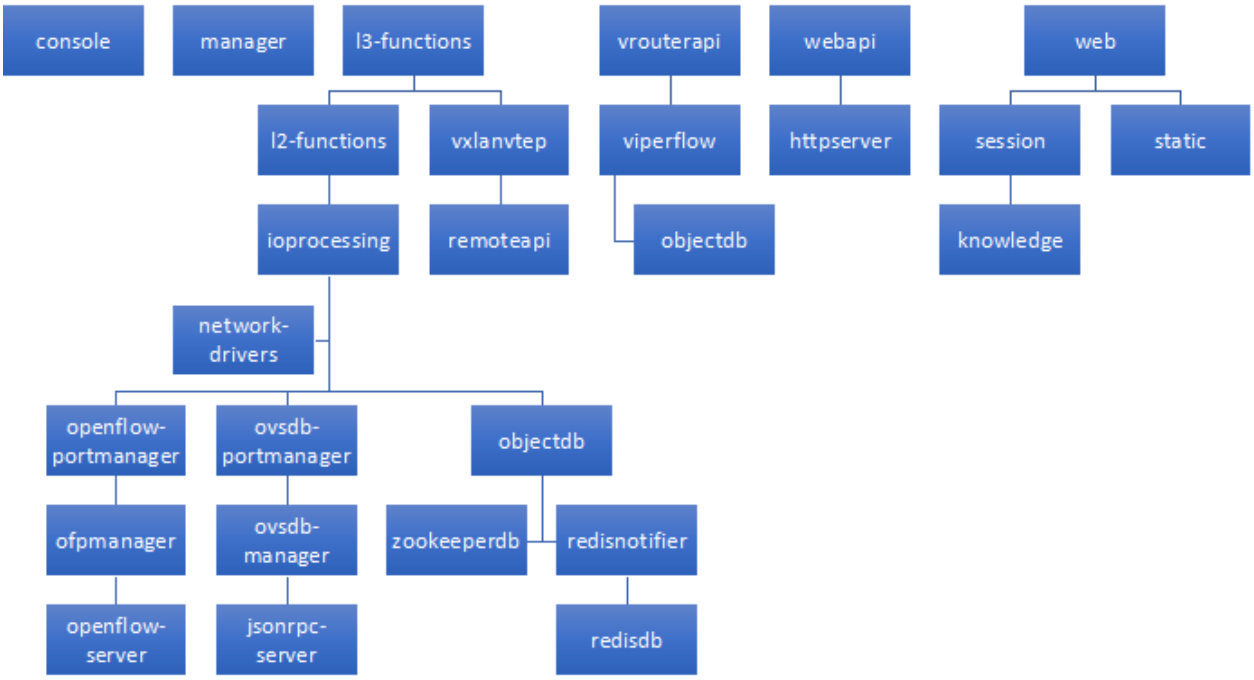

Fig. 6: Modules of VLCP

This is the current modules and relationships between them. See *Modules of VLCP*.

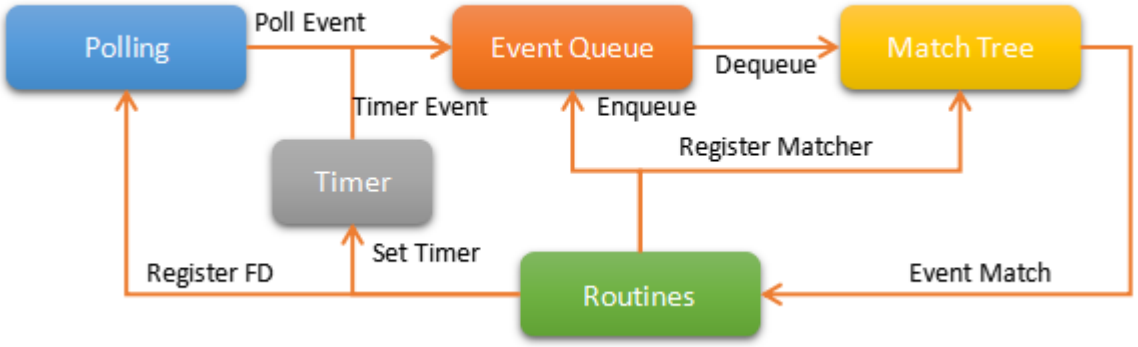

Fig. 7: Scheduler Work Flow

This is the design of the core scheduler of VLCP, it provides asynchronous programming interfaces for this software. See *Scheduler Work Flow*.

This is the architecture of the central storage system. VLCP uses an external KV-database for data storage, any nodes can read from or write to the central database independently. A transaction layer - ObjectDB is used to synchronize these operations. See *Central Storage*.

This is the design of ObjectDB. See *ObjectDB Basic Design*.

This is the description of isolations for walkers. ObjectDB uses a special way - *walker* function to describe a reading transaction, and uses *updater* function to describe a writing transaction. This provides a very flexable and easy way

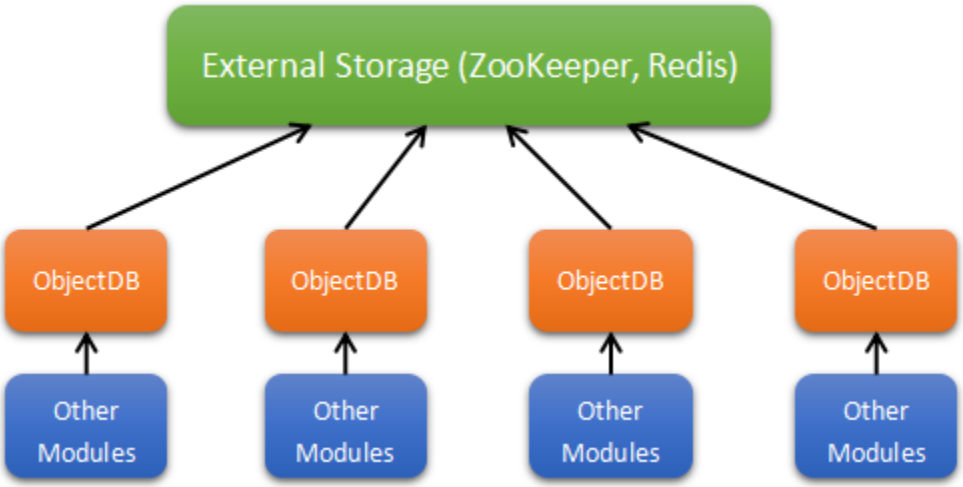

Fig. 8: Central Storage

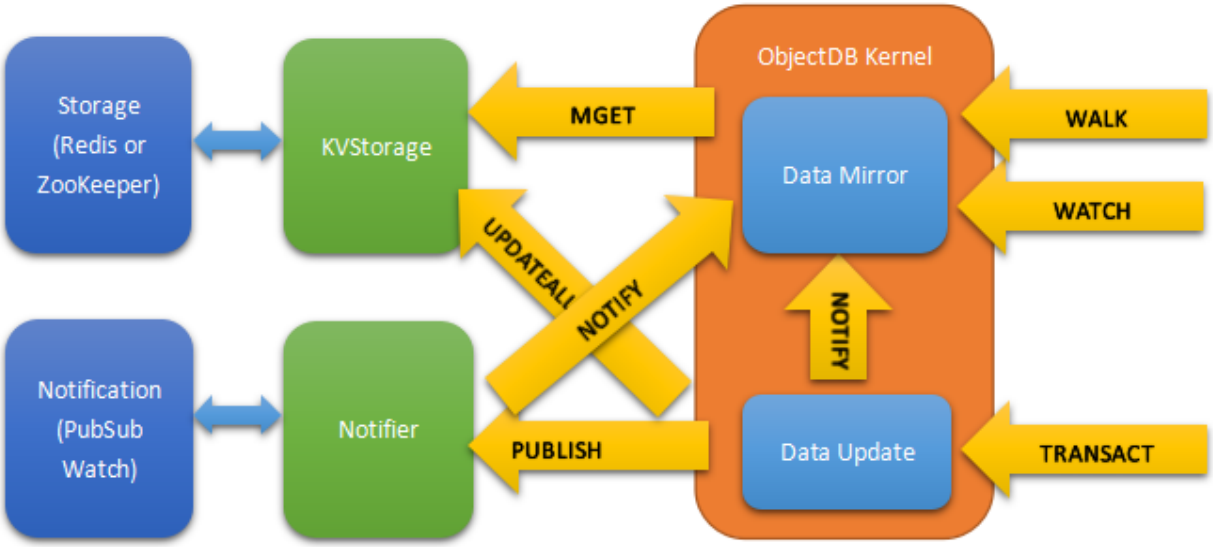

Fig. 9: ObjectDB Basic Design

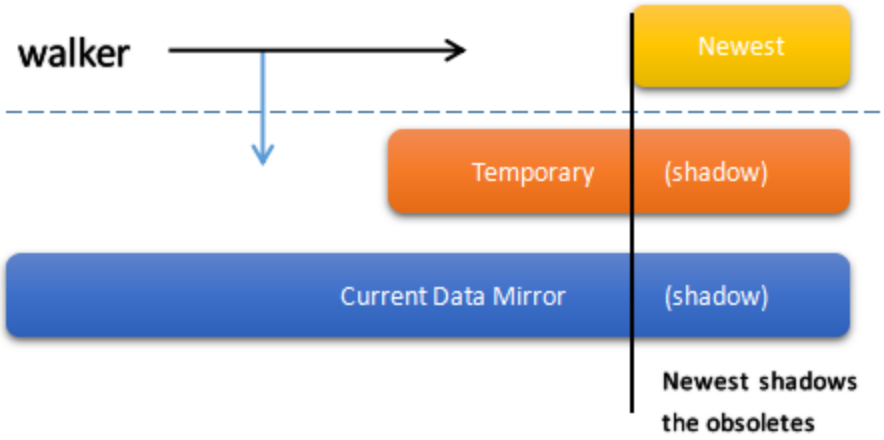

Fig. 10: Isolation of Data Space for walkers

to use the transaction. When executing a walker, VLCP uses this technique to provide consistent data for the walkers. See *Isolation of Data Space for walkers*.

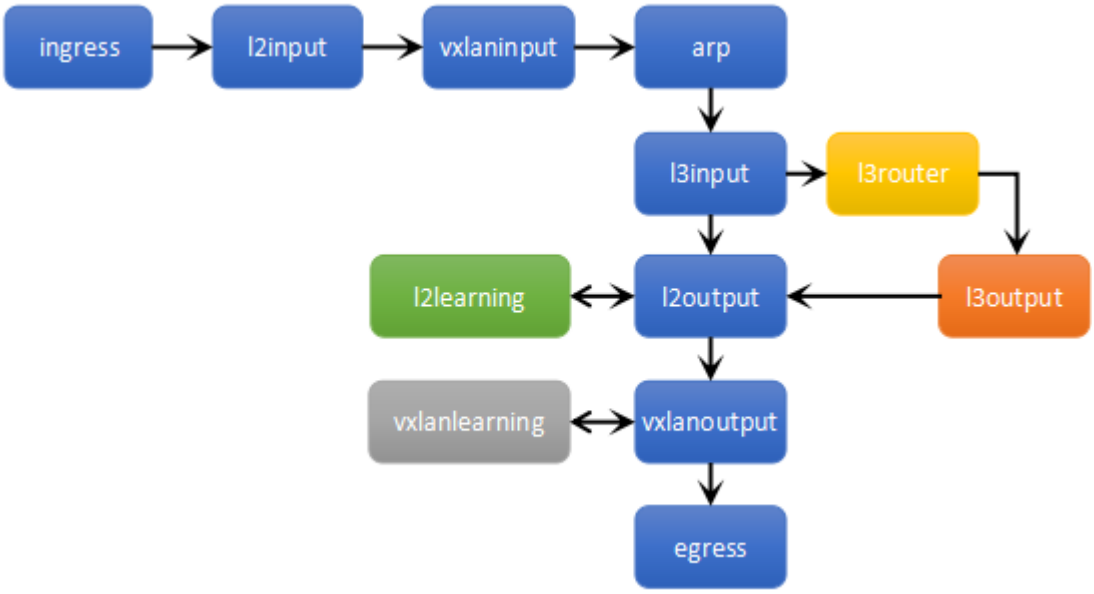

Fig. 11: OpenFlow Tables

This is the currently used OpenFlow tables in VLCP. See *OpenFlow Tables*.

CHAPTER 2

Quick Guide

These are the tutorials for a quick start and test. See *User Manual* for administration configurations, *Development Guide* for architecture explanations and developing guides, and *Reference* for module indexing.

2.1 Quick Understand on VLCP Network Concepts

This tutorial is a quick describe on VLCP network concepts. You should read this first to have better understanding on these concepts. They should be easy to understand and look familiar for OpenStack users or users with experience on other SDN products.

2.1.1 Layer-2 Network (Ethernet)

With the standard network model[1], VLCP assumes you connect your virtual machines or docker containers with OpenvSwitch (or OVS)[2].

The physical view of the network can be described with the figure :*Physical View of the Network*.

In this figure, two physical servers are connected with an external network. On each server, two endpoints (virtual machines or docker containers) are connected to an OVS bridge with a **Logical Port** for each endpoint. A **Physical**

[1] We should mention that VLCP is a full stack SDN controller which is:

1. An asynchronous network programming framework with dynamic module mangement functions
2. An OpenFlow controller framework friendly for extending
3. A product-ready controller for standard virtual network environment, supported by default modules

Loading different modules can give VLCP completely different functions. In this document, we assume users would like to use the standard network model (also named as "ViperFlow" model) to implement a quick SDN environment highly compatible with traditional networks. You may also want to develope your own SDN controller, possibly supporting OpenFlow 1.0, or targeting physical OpenFlow switches. In that situation, you do not need to be bounded to this model. Refer to *Development Guide* for more informations on developing your own controller with VLCP framework, or learn more about the implementation details of the standard model.

[2] OpenvSwitch is a software implementation of OpenFlow switch with customized extensions. It should be familiar to experienced SDN users. Most virtual network environments and tools (libvirt, for example) support using OpenvSwitch to connect network endpoints to external networks. You may learn more about OpenvSwitch from the official web site (http://openvswitch.org/)

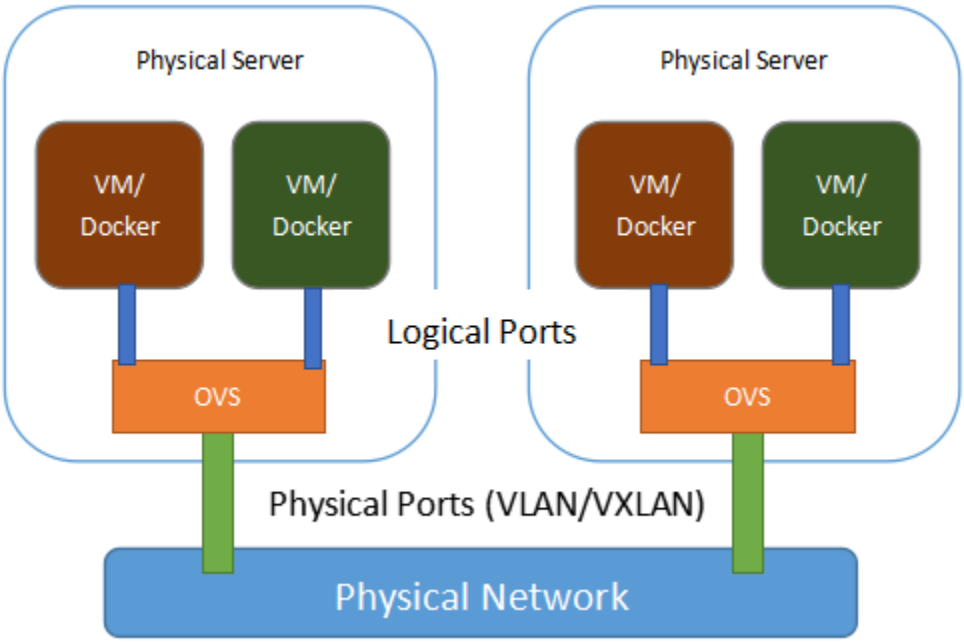

Fig. 1: Physical View of the Network

Port which is connected to the external network (or the **Physical Network**) is also connected to the OVS bridge, connecting endpoints on this physical server with endpoints on other physical servers.

Physical Network refers to the external connections between physical servers. It must be a L2 connection, means that it can transfer Ethernet packets from one physical server to another. Usually it can be separated with VLAN-tag/VNI to isolate different type of traffic, making it possible to create multiple **Logical Networks** on the same **Physical Network**. In most cases, the **Physical Network** should be:

- A physical network connecting the physical server with "trunk" port in the physical switches, so different **Logical Networks** can be isolated with *VLAN* tag
- A overlay network connecting the physical servers with VXLAN tunnels. Different **Logical Networks** can be isolated with *VXLAN* ID (or *VNI*)

If necessary, there could be more than one **Physical Networks** on the same physical server.

Physical Port refers the OVS port connecting the OVS bridge with the **Physical Network**.

- For *VLAN* networks, the port should be a physical NIC or bonding of NICs which directly connects to physical switches, and configured to "trunk" on physical switches
- For *VXLAN* networks, the port should be a tunnel port created in OVS. The method to create this type of ports will be introduced in other documents.

Logical Port refers the OVS ports connecting the endpoints to the OVS bridge. It is usually a virtual device automatically created by external tools like *libvirt* or *vlcp-docker-plugin*. They act like access ports in traditional network and switches.

Each **Logical Port** has a corresponding **Logical Network**. A **Logical Network** is a virtual L2 network connecting endpoints from different physical servers with each other. Endpoints (or **Logical Ports**) in the same **Logical Network** can send Ethernet packets to and receive Ethernet packets from each other, while they can not send or receive Ethernet packets with endpoints in other **Logical Network**. You may consider a **Logical Network** as a standard Ethernet for endpoints.

With the isolation function of the **Logcial Network**, endpoints can be grouped into virtual networks, no matter what their physical locations are. Endpoints on different physical servers can access each other as if they are connected to the same L2 switch. At the same time, endpoints on the same physical server can be isolated as if they are connected to physical isolated networks. The logical view of the network looks like the figure :*Logical View of the Network*:

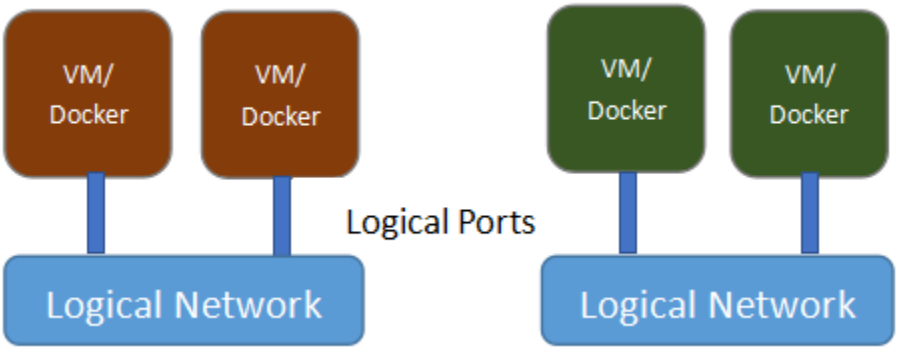

Fig. 2: Logical View of the Network

2.1.2 Logical Network and Subnet

VLCP network is functional even in pure Ethernet mode, in which case only MAC addresses are necessary to be configured on each **Logical Port**. But usually the network is used to transfer IP traffics, it is helpful to set IP and subnet configurations on the **Logical Port**. L2 connectivity is always available with or without subnet configurations.

IP configurations on a **Logical Network** is stored in a **Subnet**. Usually it contains CIDR and gateway of the network, but more advanced configurations are also available like static routes and DHCP options. Currently only IPv4 is supported.

A **Logical Port** is associated with both the **Logical Network** and the **Subnet**. The **Logical Network** association creates the L2 connectivity while the **Subnet** association affects functions related to IP addresses. Functions that need a **Subnet** configuration includes ARP proxy and embedded DHCP service.

2.1.3 Layer-3 (IP) Network and Routing

VLCP standard network model supports connecting different **Logical Networks** through L3 routing.

There are two types of networks in L3 routing: *internal networks* and *external networks*. *External networks* are **Subnets** with external gateways, the gateway set in **Subnet** of an *external network* should be IP address configured on external devices like physical routers. It should be accessible from the **Logical Network** VLAN or VXLAN. “Internal networks* are **Subnets** with internal gateways, the gateway address should not be any existed IP address in the network. VLCP SDN network will create a virtual router service for the networks as the default gateway of the *internal networks*.

Virtual router service with both *internal networks* and *external networks* forwards traffic to outer networks (Internet for example) through *external networks*. Correct routes for internal CIDRs should be configured in external gateways with static routes or technologies like OSPF, or use NAT (not yet supported).

Sometimes *internal networks* and *external networks* are in different **Physical Networks**, and not all server nodes have **Physical Ports** for **Physical Network** of the *external network*. For example, *internal networks* are connected through VXLAN, while *external networks* are using VLAN for isolation. In this situation, some *forwarding nodes* should be created to forward the traffic from *internal networks* to *external networks*. They are almost the same as other server nodes except a few configurations are modified. The figure :*L3 Network with Forwarding Nodes* shows the structure.

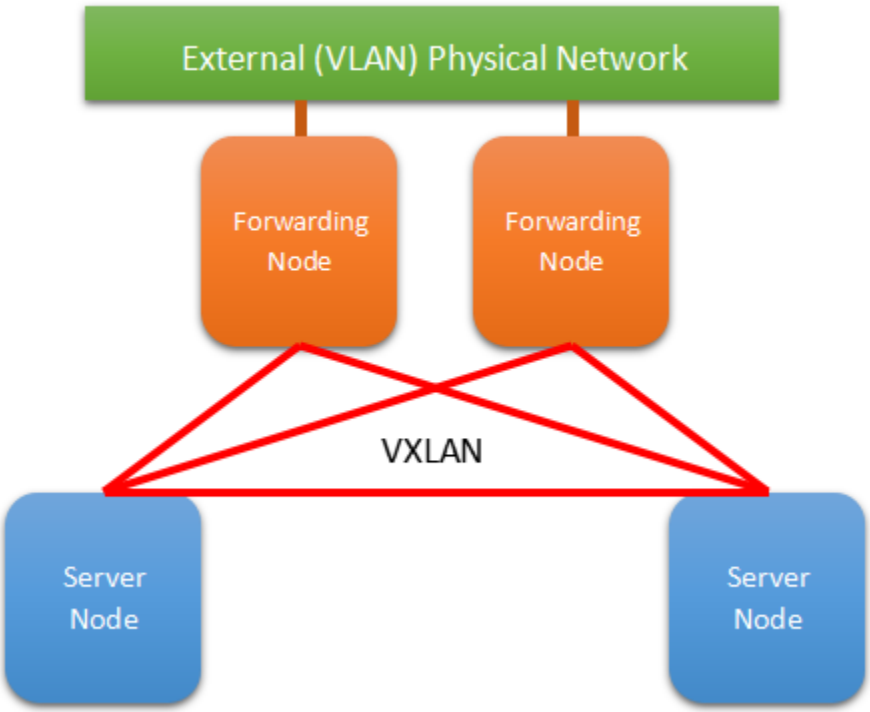

Fig. 3: L3 Network with Forwarding Nodes

2.1.4 Network Settings Structure

To setup a new network, elements in the figure :*Network Settings Structure* should be created in order from top to bottom.

2.2 Basic Test with OpenvSwitch

This tutorial creates a simple SDN test environment with two physical servers (or virtual machines instead). If you are confused with some concepts, read *Quick Understand on VLCP Network Concepts* first.

You need two physcial servers with namespace support for this tutorial. This tutorial assumes you are using CentOS 7, but any Linux version in which "ip netns" command is available can be used. If you want to try VLAN networks, you may need to configure the physical switch port to "trunk mode". If that is not possible, you can still use VXLAN isolation, in that case, you may use virtual machines (even in public cloud) to replace physical servers.

This toturial assumes you are using root account, if you run into priviledge problems with a non-root account, you may need `sudo`.

Most of the setup steps should be done on both servers, or every server in the cluster, except :*Setup Central Database*, and all the API calls with *curl* which create global objects (physical networks, physical ports, logical networks, subnets, virtual routers).

2.2.1 Install OpenvSwitch

Download OpenvSwitch releases from http://openvswitch.org/download/. You may choose a version from 2.3x, 2.5x or higher. 2.5x is recommended.

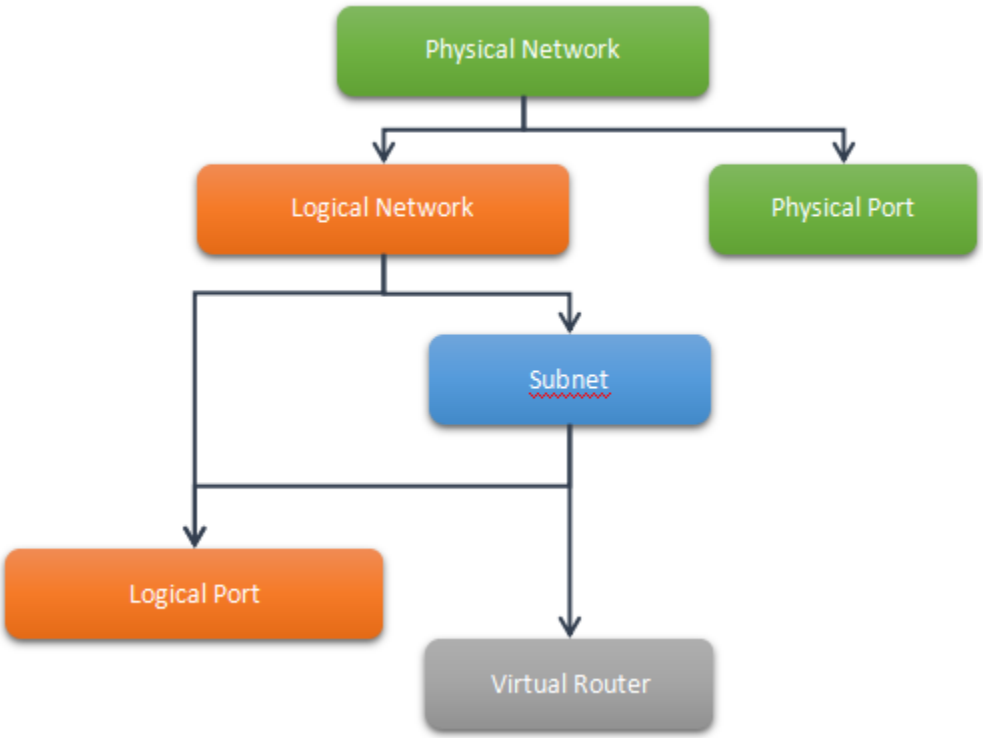

Fig. 4: Network Settings Structure

Build and install OpenvSwitch, following the steps in http://www.openvswitch.org/support/dist-docs-2.5/INSTALL.RHEL.md.html. Usually you may build a RPM package once and install it on each server nodes with *yum*.

Note: Other Linux distributions may have pre-built OpenvSwitch packages available, check the version if you want to use it instead.

2.2.2 Prepare Python Environment

VLCP v2.0 works Python 3.5+ and PyPy3.5. For production environment, PyPy3 is recommended which is about 5x faster than CPython. The most simple way to use PyPy is using the Portable PyPy distribution for Linux. You may also use a *virtualenv* environment if you want.

Note: VLCP v1.x supports Python 2.6, Python 2.7 and PyPy2, but in v2.0, only Python 3.5+ are supported.

Use pip to install VLCP. If pip is not ready, refer to https://pip.pypa.io/en/stable/installing/. If you are using old versions of pip, you may also want to upgrade pip, setuptools and wheel:

```
pip install --upgrade pip setuptools wheel
```

2.2.3 INSTALL VLCP

Install from pypy

Use pip tool auto install from pypy:

```
pip install vlcp
```

Install from source

Git clone from github:

```
git clone https://github.com/hubo1016/vlcp.git
```

Install use setup.py:

```
cd vlcp && python setup.py install
```

Some optional packages can be used by VLCP, install them as needed:

hiredis If you use Redis for the central database, install hiredis will improve the performance

python-daemon Support daemonize with "-d" option to create system service for VLCP. But it is usually more convient to use with *systemd* in CentOS 7.

2.2.4 Setup Central Database

VLCP uses external KV-storage as a central database for configuration storage and distributed coordination. Currently *Redis* and *ZooKeeper* is supported. *ZooKeeper* is recommended for its high availability with clustering.

Choose **one of the follwing sections** to proceed. You may install the central database on one of the physical servers or on another server.

Install ZooKeeper (Recommended)

Follwing steps in http://zookeeper.apache.org/doc/current/zookeeperStarted.html. For this tutorial, install a standalone node is enough, but you should setup a cluster for production environment.

Note: You should protect the server port 2181 with *iptables* to grant access only from your servers.

Install Redis (Optional)

Download, compile and install redis from https://redis.io/download. You should configure redis server to listen to 0.0.0.0 or a IP address available for all the servers. You may also want to enable AOF.

Caution: It is a huge security hole to open redis port on Internet without password protection: attackers can grant full controll of your server by overwriting sensitive files like ~/.ssh/authorized with administration commands. Make sure you use private IP addresses, or configure *iptables* or your firewall correctly to allow connecting only from your servers.

Newer versions of Redis also blocks this configuration with a "protect mode", you may need to disable it after you configure *iptables* correctly.

2.2.5 Create VLCP Configuration

Download example configuration file from GitHub and save it to `/etc/vlcp.conf` as a start. In this tutorial, we will use generic13.conf:

```
curl https://raw.githubusercontent.com/hubo1016/vlcp/master/example/config/generic13.
↪conf\
    > /etc/vlcp.conf
```

Modify the `module.zookeeperdb.url` line with your ZooKeeper server addresses, or if you are using Redis, following the comments in the configuration file.

Note: `module.jsonrpcserver.url='unix://var/run/openvswitch/db.sock'` special where the UNIX socket which communicate with ovs. if install ovs from source , the UNIX socket file mybe `unix://usr/local/var/run/openvswitch/db.sock`.

2.2.6 Start VLCP Service

It is usually a good idea to create a *system service* for VLCP to make sure it starts on server booting, but for convient we will start VLCP with *nohup* instead:

```
nohup vlcp-start &
```

or:

```
nohup python -m vlcp.start &
```

To stop the service, run command `fg` and press Ctrl+C, or use *kill* command on the process.

Note: You should start the controller with root priviledge (`sudo` if necessary).

2.2.7 Configure OpenvSwitch

Create a test bridge in OpenvSwitch for virtual networking. The name of the bridge usually does not matter. In this tutorial we use `testbr0`. For docker integration, the bridge name `dockerbr0` is usually used. Run following commands on each server:

```
ovs-vsctl add-br testbr0
ovs-vsctl set-fail-mode testbr0 secure
ovs-vsctl set-controller testbr0 tcp:127.0.0.1
```

This creates the test bridge and the OpenFlow connection to the VLCP controller.

Note: VLCP communicates with OpenvSwitch in two protocols: OpenFlow and OVSDB (a specialized JSON-RPC protocol). Usually the SDN controller is deployed on the same server with OpenvSwitch, in that case the default OVSDB UNIX socket is used, so we do not need to configure OVSDB connections with `ovs-vsctl set-manager`.

ovs fail-mode secure means ovs disconnect with controller, ovs will not set up flows on its own another fail-mode standalone ovs will set up flows cause the datapath to act link an ordinary MAC-learning switch.

From now on, if you run into some problems, or you want to retry this toturial, you can delete the whole bridge:

```
ovs-vsctl del-br testbr0
```

And cleanup or re-install the central database.

2.2.8 Create VXLAN Physical Network

There is only one step to create a physical network. The example configuration open a management API port at `http://localhost/8081`. We will call the management API with curl:

```
curl -g -d 'type=vxlan&vnirange=`[[10000,20000]]`&id=vxlan' \
           'http://localhost:8081/viperflow/createphysicalnetwork?'
```

You may run this command on any of your server nodes. All server nodes share the same data storage, so you create the network configuration once and they can be used anywhere.

The id of newly created physical network is "vxlan", this is a convient name for further calls, but you can replace it with any name you like. If you do not specify an id, VLCP creates a UUID for you. `vnirange` specify a list of VNI ranges, notice that different from *range* in Python, these ranges include **both** begin and end. For example, `[10000,20000]` is 10000-20000, which has 10001 VNIs enable. Network engineers are usually more familar with this type of ranges.

Note: By default, the management API supports HTTP GET (with query string), HTTP POST (with standard form data), and HTTP POST with JSON-format POST data. Though use the HTTP GET/POST format is usually the easiest

way to call the API in Shell command-line, when integrating with other systems JSON-format POST may be more convient.

‘‘ quoted expression is a VLCP-specified extension. Some APIs need data types other than strings for its parameters. When a string parameter is quoted by ‘‘, VLCP recognizes it as a literal expression in Python. You may use numbers, string, tuples, list, dictionary, sets and any combinations of them in a quoted expression.

‘[]’ have special meanings in *curl*, that is way we use `-g` option to turn it off.

2.2.9 Create Physical Port

Every physical network need one physical port for each server to provide external connectivity. There are two steps to create a physical port:

1. Create the port on each server and plug the port to the bridge
2. Create the physical port configuration in VLCP

Note: These two steps can be done in any order. When you extend a cluster, you only need to do Step 1. on new servers since the second step is already done.

First create a vxlan tunnel port in each server:

```
ovs-vsctl add-port testbr0 vxlan0 -- set interface vxlan0 type=vxlan \
          options:local_ip=10.0.1.2 options:remote_ip=flow options:key=flow
```

Replace the IP address `10.0.1.2` to an external IP address on this server, it should be different for each server. VLCP will use this configuration to discover other nodes in the same cluster.

Note: `options:remote_ip=flow` means vxlan dst server ip , will set use flow dynamic `options:key=flow` means vxlan tunnel id , will set use flow dynamic.

The port name `vxlan0` can be replaced to other names, but you should use the same name for each server.

Note: VXLAN uses UDP port 4789 for overlay tunneling. You must configure your *iptables* or firewall to allow UDP traffic on this port. If there are other VXLAN services on this server (for example, overlay network driver in docker uses this port for its own networking), you may specify another port by appending `option:dst_port=4999` to the commandline. Make sure all your servers are using the same UDP port.

Then create the physical port configuration (only once, on any server node):

```
curl -g -d 'physicalnetwork=vxlan&name=vxlan0'\
        'http://localhost:8081/viperflow/createphysicalport'
```

The `physicalnetwork` parameter is the physical network ID, and the `name` parameter is the port name in above command.

2.2.10 Create Logical Network and Subnets

In this tutorial, we will create two logical networks:

- **Network A**: CIDR 192.168.1.0/24, network ID: network_a, gateway: 192.168.1.1
- **Network B**: CIDR 192.168.2.0/24, network ID: network_b, gateway: 192.168.2.1

The steps are simple and direct. In VLCP, Ethernet related configurations are provided when createing a **Logical Network**, and IP related configurations are provided when creating a **Subnet**. First create two logical networks:

```
curl -g -d 'physicalnetwork=vxlan&id=network_a&mtu=`1450`'\
      'http://localhost:8081/viperflow/createlogicalnetwork'
curl -g -d 'physicalnetwork=vxlan&id=network_b&mtu=`1450`'\
      'http://localhost:8081/viperflow/createlogicalnetwork'
```

Note: VXLAN introduces extra overlay packet header into the packet, so we leave 50 bytes for the header and set MTU=1450. If your underlay network supports larger MTU, you can set a larger MTU instead. The embedded DHCP service uses this configuration to generate a DHCP Option to set MTU on the logical port (vNIC in a virtual machine). *vlcp-docker-plugin* also uses this to generate MTU configurations for docker.

You may use an extra parameter `vni=10001` to explictly specify the VNI used by this logical network. If ommited, VLCP automatically assign a free VNI from the physical network VNI ranges. The creation fails if all the VNIs in VNI ranges are used, or the specified VNI is used.

Then, create a *Subnet* for each logical network:

```
curl -g -d 'logicalnetwork=network_a&cidr=192.168.1.0/24&gateway=192.168.1.1&
↪id=subnet_a'\
      'http://localhost:8081/viperflow/createsubnet'
curl -g -d 'logicalnetwork=network_b&cidr=192.168.2.0/24&gateway=192.168.2.1&
↪id=subnet_b'\
      'http://localhost:8081/viperflow/createsubnet'
```

Note: There are also batch create APIs like `createlogicalnetworks` and `createsubnets`, which accepts a list of dictionaries to create multiple objects in one transact. A batch create operation is an atomic operation, if one of the object is not created successfully, all the other created objects roll back.

2.2.11 Create Logical Ports

We will create one logical port for each logical network and each physical server - means 4 logical ports if you have two physical servers.

Run following commands on each server:

```
SERVER_ID=1

# create namespace
ip netns add vlcp_ns1

# create logicalport id
LOGPORT_ID=lgport-${SERVER_ID}-1

# add internal ovs interface set iface-id logicalport id
ovs-vsctl add-port testbr0 vlcp-port1 -- set interface vlcp-port1 \
      type=internal external_ids:iface-id=${LOGPORT_ID}
```

(continues on next page)

(continued from previous page)

```
# get interface mac address used to create logical port
MAC_ADDRESS=`ip link show dev vlcp-port1 | grep -oP 'link/ether \S+'\
       | awk '{print $2}'`
curl -g -d "id=${LOGPORT_ID}&logicalnetwork=network_a&subnet=subnet_a&mac_address=$
↪{MAC_ADDRESS}"\
       "http://localhost:8081/viperflow/createlogicalport"

# move interface link to namespace and up it
ip link set dev vlcp-port1 netns vlcp_ns1
ip netns exec vlcp_ns1 ip link set dev vlcp-port1 up

# start dhcp to get ip address
ip netns exec vlcp_ns1 dhclient -pf /var/run/dhclient-vlcp-port1.pid\
       -lf /var/lib/dhclient/dhclient-vlcp-port1.leases vlcp-port1

# create another namespace
ip netns add vlcp_ns2

# create another logical port id
LOGPORT_ID=lgport-${SERVER_ID}-2

# add internal ovs interface set iface-id logicalport id
ovs-vsctl add-port testbr0 vlcp-port2 -- set interface vlcp-port2 \
      type=internal external_ids:iface-id=${LOGPORT_ID}

# get interface mac address used to create logical port
MAC_ADDRESS=`ip link show dev vlcp-port2 | grep -oP 'link/ether \S+'\
      | awk '{print $2}'`
curl -g -d "id=${LOGPORT_ID}&logicalnetwork=network_b&subnet=subnet_b&mac_address=$
↪{MAC_ADDRESS}" \
      "http://localhost:8081/viperflow/createlogicalport"

# move interface link to namespace and up it
ip link set dev vlcp-port2 netns vlcp_ns2
ip netns exec vlcp_ns2 ip link set dev vlcp-port2 up

# start dhcp to get ip address
ip netns exec vlcp_ns2 dhclient -pf /var/run/dhclient-vlcp-port2.pid \
      -lf /var/lib/dhclient/dhclient-vlcp-port2.leases vlcp-port2
```

Change `SERVER_ID` to a different number for each of your server to prevent the logical port ID conflicts with each other.

A quick description:

For each port

1. Create a namespace to simulate a logical endpoint with separated devices, IP addresses and routing.
2. Create an ovs internal port to simutate a vNIC. “external_ids:iface-id” is set to the logical port id.
3. Use the logical port ID, logical network ID, subnet ID and the MAC address to create a new logical port configuration.
4. Move the internal port to the created namespace.
5. Start DHCP client in the namespace to acquire IP address configurations.

Note: When creating logical ports, you can specify an extra parameter like `ip_address=192.168.1.2` to

explictly assign an IP address for the logical port; if omitted, a free IP address is automatically choosen from the subnet CIDR. See API references for details.

dhclient is used to use DHCP to retrieve IP address and MTU configurations from embedded DHCP server.

Use:

```
ip netns exec vlcp_ns1 dhclient -x -pf /var/run/dhclient-vlcp-port1.pid \
      -lf /var/lib/dhclient/dhclient-vlcp-port1.leases vlcp-port1
```

to stop it.

You may also configure the IP addresses and MTU yourself, instead of acquiring from DHCP.

It is not necessary to call `createlogicalport` API on the same server where the ovs port is created. The order is also not matter (if you use a fixed MAC address). If you delete the ovs port and re-create it on another server, all configurations are still in effect, so you can easily migrate a virtual machine or docker container easily without network loss.

You may also choose to omit the `id` parameter to let VLCP generate an UUID for you. Then you can set the UUID to `external_ids:iface-id` of the ovs port.

Now you should see the logical ports in the same logical networks can ping each other, while logical ports from different logical networks cannot ping each other. Try it yourself:

```
ip netns exec vlcp_ns1 ping 192.168.1.3
```

2.2.12 Create Virtual Router

As you can see, logical ports in different logical networks cannot access each other with L2 packets. But you can connect different logical networks with a **Virtual Router**, to provide L3 connectivity between logical networks. This keeps the broadcast range of logical networks in a reasonable scale.

Let's create a virtual router and put subnet_a, subnet_b inside it:

```
curl -g 'http://localhost:8081/vrouterapi/createvirtualrouter?id=subnetrouter'
curl -g -d 'interfaces=`[{"router":"subnetrouter","subnet":"subnet_a"},\
                         {"router":"subnetrouter","subnet":"subnet_b"}]`'\
        'http://localhost:8081/vrouterapi/addrouterinterfaces'
```

Now the logical ports should be enabled to ping each other no matter which logical network they are in:

    ip netns exec vlcp_ns1 ping 192.168.2.2

2.2.13 (Optional) Create VLAN Physical Networks

If your server are connected to physical switches, and the ports your server connected to are configured to "trunk mode", and there are VLANs correctly configured and permitted in the physical switches, you may create a VLAN physical network to connect your vNICs through VLAN network. Usually it is an easy way to connect your vNICs to traditional networks.

It is not that different to create a VLAN physical network from creating a VXLAN physical network. We will assume your VLAN network is connected by a physical NIC or bonding device named `bond0`:

```
curl -g -d 'type=vlan&vlanrange=`[[1000,2000]]`&id=vlan'\
       'http://localhost:8081/viperflow/createphysicalnetwork'
curl -g -d 'physicalnetwork=vlan&name=bond0'\
       'http://localhost:8081/viperflow/createphysicalport'
```

And on each server:

```
ovs-vsctl add-port testbr0 bond0
```

Creating logical networks and other parts of the network is same.

Note: If your VLAN network has external gateways, you may want to specify `is_external=`True`` when creating subnets. When this subnet is connected to a virtual router, virtual router uses the external gateway as the default gateway. Static routes should be configured on the external gateway for other logical networks connected to the virtual router. Or you may use NAT instead, though current version does not support NAT yet, it is not too difficult to implement a simple source NAT solution with *iptables*.

2.2.14 Remove Network Objects

When removing configurations from VLCP, use a reversed order: **Logical Ports**, **Virtual Router**, **Subnet**, **Logical Network**, **Physical Ports**, **Physical Network**:

```
SERVER_ID=1
curl -g -d 'ports=`[{"id":"'"lgport-${SERVER_ID}-1"'"},
                    {"id":"'"lgport-${SERVER_ID}-2"'"}]`'\
      'http://localhost:8081/viperflow/deletelogicalports'

curl -g -d 'interfaces=`[{"router":"subnetrouter","subnet":"subnet_a"},
                         {"router":"subnetrouter","subnet":"subnet_b"}]`'\
      'http://localhost:8081/vrouterapi/removerouterinterfaces'
curl -g 'http://localhost:8081/vrouterapi/deletevirtualrouter?id=subnetrouter'

curl -g 'http://localhost:8081/viperflow/deletesubnet?id=subnet_a'
curl -g 'http://localhost:8081/viperflow/deletesubnet?id=subnet_b'
curl -g 'http://localhost:8081/viperflow/deletelogicalnetwork?id=network_a'
curl -g 'http://localhost:8081/viperflow/deletelogicalnetwork?id=network_b'
curl -g 'http://localhost:8081/viperflow/deletephysicalport?name=vxlan0'
curl -g 'http://localhost:8081/viperflow/deletephysicalnetwork?id=vxlan'
```

After this you can remove the ovs bridge and namespace created on each server to restore the environment:

```
ip netns exec vlcp_ns1 dhclient -x -pf /var/run/dhclient-vlcp-port1.pid\
      -lf /var/lib/dhclient/dhclient-vlcp-port1.leases vlcp-port1
ip netns exec vlcp_ns2 dhclient -x -pf /var/run/dhclient-vlcp-port2.pid\
      -lf /var/lib/dhclient/dhclient-vlcp-port2.leases vlcp-port2
ovs-vsctl del-br testbr0
ip netns del vlcp_ns1
ip netns del vlcp_ns2
```

2.3 Docker Integration Quick Guide

This toturial shows how to integrate VLCP into docker with *vlcp-docker-plugin*. With this customized network plugin, you can use standard `docker network create` and `docker run` command to create networks and containers with SDN functions. It offers more functions (L3 networking, VLAN support) and better stabilities than the default *overlay* network plugin.

This toturial assumes the operating system to be CentOS 7, but there should be little differences for other Linux distribution e.g. Ubuntu, Debian, Fedora. This toturial assumes you are using root account, if you run into priviledge problems with a non-root account, you may need `sudo`.

Most of the setup steps should be done on every server in the cluster, except :*Setup Central Database*, and all the API calls with *curl* which create global objects (physical networks, physical ports).

This plugin fully supports using Docker Swarm to manage the docker cluster. with Docker Swarm, you can create containers on your cluster as if you are using a single server.

vlcp-docker-plugin is a separated project in vlcp-docker-plugin GitHub Home Page.

2.3.1 Prepare the Environment

Install OpenvSwitch on each server: this is the same step as :*Install OpenvSwitch*.

Install vlcp-docker-plugin: this is almost the same as :*Prepare Python Environment*, but you must also install vlcp-docker-plugin:

```
pip install vlcp-docker-plugin
```

2.3.2 Setup Central Database

Because docker daemon and docker swarm also need a central database, and they both support ZooKeeper, so it is always recommended to setup a ZooKeeper cluster as mentioned in :*Install ZooKeeper (Recommended)*, so VLCP can share the same cluster with docker - though not necessary.

2.3.3 Install Docker Engine

If Docker Engine is not already installed, refer to the official document (Linux / CentOS 7) to install Docker Engine. You should specify these extra configurations, either from `/etc/docker/daemon.json`, or `/etc/docker/docker.conf`, to enable multi-host networks with an external KV-storage as mentioned in this document:

Name	Description	Example Value
cluster-store	`zk://` started ZooKeeper cluster URL	`zk://server1:port1,server2:port2,../`
hosts	Add an extra TCP socket end-point	`["0.0.0.0:2375","unix:///var/run/docker.sock"]`
cluster-advertise	Advertise the TCP socket end-point	`10.0.1.2:2375`

`cluster-advertise 10.0.1.2:2375` special server ip communicate with other server in cluster.

Caution: It is very dangerous to expose a docker API endpoint to untrust network without protection. Configure *iptables* or enable tls to secure your service.

2.3.4 Create VLCP Service

Create a configuration file with the example configuration in Git Hub:

```
curl https://raw.githubusercontent.com/hubo1016/vlcp/master/example/config/docker.
↪conf\
      > /etc/vlcp.conf
```

Modify the `module.zookeeperdb.url` line with your ZooKeeper cluster addresses.

We will create a system service this time. First create a starting script:

```
tee /usr/sbin/vlcp <<-'EOF'
#!/bin/bash
mkdir -p /run/docker/plugins
rm -f /run/docker/plugins/vlcp.sock
exec /usr/bin/vlcp-start
EOF

chmod +x /usr/sbin/vlcp
```

Note: If you are not using the default Python environment, replace `/usr/bin/vlcp-start` to the full path of the environment, and load virtualenv environment if necessary.

This script creates necessary directory structures, and clean up the socket file if it is not removed correctly, before starting the VLCP service.

Then create a system service with *systemd* (add `sudo` if necessary):

```
tee /usr/lib/systemd/system/vlcp.service <<-'EOF'
[Unit]
Description=VLCP
After=network.target
Before=docker.service
[Service]
ExecStart=/usr/sbin/vlcp
[Install]
WantedBy=multi-user.target
EOF

systemctl daemon-reload
systemctl enable vlcp
systemctl start vlcp
```

The final statement starts the controller and the docker plugin. If Docker Engine is not started, you can start it now.

2.3.5 Configure Physical Network

These are the same steps as :*Configure OpenvSwitch*, :*Create VXLAN Physical Network* and :*Create Physical Port*, but replace the OpenvSwitch bridge name with `dockerbr0`.

Note: When creating physical ports, it is recommended to change the default OpenvSwitch VXLAN port with an extra ovs-ctl command-line option `option:dst_port=4999`, because *overlay* network driver in Docker Engine also uses VXLAN port UDP 4789 for its own networking. If you use *overlay* network and VLCP network in the same time, the network drivers conflict with each other and make either or both stop working. Must replace OpenvSwitch bridge name with `dockerbr0` because vlcp docker plugin attach link to this bridge name.

You may also create VLAN networks as mentioned in :*(Optional) Create VLAN Physical Networks*.

2.3.6 Create Network in Docker

With docker plugin, creating a VLCP network in docker is the same with other network drivers:

```
docker network create -d vlcp -o physicalnetwork=vxlan -o mtu=1450 --ipam-driver vlcp␣
↪\
      --subnet 192.168.1.0/24 --gateway 192.168.1.1 test_network_a
```

Note: You may also use the corresponding docker API /networks/create

The `-o` options pass options to VLCP network, as if they are passed to `viperflow/createlogicalnetwork`. The `physicalnetwork` option is necessary; others are optional. Also the `` quoted extension is supported (Make sure you surround them with `'` to prevent them been executed by Shell). Common options are:

physicalnetwork Should be the ID of the physical network created in :*Create VXLAN Physical Network*

vni/vlanid Specify a VNI / VLAN tag instead of let VLCP choose one for you

mtu Set MTU for this network. Usually you should set the network MTU to 1450 for VXLAN networks to leave space for overlay headers.

subnet:... Options prefixed with `subnet:` are passed when creating the subnet, as if they are passed to `viperflow/createsubnet`. Common options are:

subnet:disablegateway Set this option to "true" or `` `True` `` make VLCP removes the default gateway in the container. This let docker creates an extra vNIC for the container, and connect the container to the `docker_gwbridge` network. If you want to use functions from the bridge network e.g. source NAT, port map (PAT) from/to physical server network. But you will not be able to use **Virtual Routers** to connect these subnets, unless you also specify `subnet:host_routes`.

subnet:host_routes This option creates static routes for the subnet, and they are automatically set in the container. This is useful for many tasks like creating site-to-site VPNs or customized gateways/firewalls. You may also use this option to create routes to the gateway to override `subnet:disablegateway`, making it possible to use **Virtual Router** together with `docker_gwbridge`

subnet:allocated_start This option customized the allowed IP range for the containers. This should be the first IP address allowed to be used by the containers in this network. By default every IP address (except the gateway address) can be assigned to the container; with these two options, the IP addresses for the container are limited to this range, making it possible for a network to share the same address space with existed devices.

subnet:allocated_end This is the end of the customized IP range. This should be the last IP address allowed to be used by the containers in this network.

You may also specify customized options. Unrecognized options are also written to the **Logical Network** or **Subnet** configurations, they may act as metadata or serve integration purposes.

Note: *vlcp-docker-plugin* is both a network driver and an IPAM driver, means it can manage IP addresses itself. It is recommended to use `--ipam-driver vlcp` option to enable VLCP as the IPAM driver instead of using the default IPAM driver, but please be aware that this IPAM driver can only be used with VLCP network driver; it cannot be used with other network drivers.

The default IPAM driver of Docker Engine does not allow containers in different networks use the same IP address. In fact, different networks with a same CIDR shares the same address space. This may lead to difficulties on some task: creating copies of containers with the exactly same IP addresses for example. In contrast, VLCP IPAM driver always uses a separated address space for every logical network, so it is possible to create containers with exactly the same IP address in different networks. This ensures full network virtualization especially for systems which are shared by multiple users. Since different logical networks are explictly isolated with each other in L2, These duplicated IP addresses will not cause any trouble for you.

Global networks are shared among all the server nodes in a cluster. When you create the network in any of the servers, all the other servers should be able to see and use the network.

2.3.7 Create Containers in VLCP network in Docker

It is straight forward to create a container in VLCP network:

```
docker run -it -d --network test_network_a --name test_vlcp_1 centos
```

This uses the official CentOS image to create a new container.

Note: Or you can use the corresponding docker API /containers/create and /containers/(id or name)/start

---

The only important part is to specify the network name or ID with `--network`. You may also use `--ip` to specify an IP address, use `--mac-address` to specify MAC address just as networks created by other drivers.

You can create containers on any server in the cluster with the same network. They can access each other as soon as you create them. Try this on another server:

```
docker run -it -d --network test_network_a centos ping test_vlcp_1
```

2.3.8 Remove Networks and Containers

vlcp-docker-plugin automatically remove the related objects when you use `docker stop`, `docker rm` and `docker network rm` to remove the created objects, basicly you do not need to worry about the underlay constructure.

2.3.9 Restore from a Docker Engine Panic

Docker engine crashes (panic), kernel panics or power losts of physical servers create inconsistencies in docker engine, and may lead to issues which make the containers fail to start. It is usually much easier to restore a VLCP network created with `--ipam-driver=vlcp` enabled. Usually the following script fix all the problems:

```
python -m vlcp_docker.cleanup -H :2375 -f /etc/vlcp.conf
```

Note: This script do the following jobs:

1. Remove the vNICs that are already not in a container.
2. Remove unnecessary or failed ports in OpenvSwitch.
3. Check and remove gabage configurations in VLCP comparing with information from docker

All the information will be double-checked to prevent race conditions, so it is safe to use this script in any situation.

Notice that some problems which make the containers fail to start are not network-related problems, VLCP can do nothing for them. It only ensures the network part does not block you.

CHAPTER 3

User Manual

These are the configuration and API manuals for VLCP.

3.1 Configuration Manual

VLCP uses a configuration file to change the behaviors of all the internal objects. This file determines which modules to load, which strategy the modules should use, and which endpoints the controller should connect to or from.

3.1.1 Edit the Configuration File

By default the configuration file is `/etc/vlcp.conf`, but you can modify the configuration file position with the `-f` option when running `vlcp-start`.

`#` mark and the contents after the mark are remarks. They are removed on parsing.

Each line of the configuration file (except blank lines) should be:

```
<configkey> = <configvalue>
```

`configkey` is a name stands for the configuration target. You can find the list of available configuration keys in :*All Available Configurations*. There should not be any spaces or tabs before `configkey`.

`configvalue` is a literal expression in Python i.e. a Python constant like *int*, *float*, *str*, *bytes*, *unicode*, *set*, *dict*, *list*, *tuple* or any combinations of them. It cannot use other expressions like evaluations or function calls.

The `configvalue` can extend to multiple lines. The extended lines should have spaces at the begining of the lines to identify them from normal configurations. Also they must follow the Python grammer, means you may need to append extra `\` at the end of a line.

It is always a good idea to begin with Example Configurations

3.1.2 Configuration Rules

Usually a configuration key is made up of two or three parts:

```
<type>.<classname>.<config>
<classname>.<config>
```

Keys made up of three parts are configurations of a class derived from a base class like *Protocol*, *Module*. The lower-cased base class name is `<type>`, and the lower-cased class name is `<classname>`. For example, configuration keys of a module `vlcp.service.connection.zookeeperdb.ZooKeeperDB` begin with `module.zookeeperdb.`, where `module` is lower-cased `Module`, `zookeeperdb` is lower-cased `ZooKeeperDB`.

The base classes can also be configured. The keys begin with `<type>.default.`. This configuration replaces default values of all sub classes. If the same `<config>` is also configured on the sub class, sub class configurations take effect. The priority order is (from higher to lower):

```
<type>.<subclass>.<config>
<type>.default.<config>
(default value)
```

Keys made up of two parts are classes which do not have base classes and can not be inherited.

A special kind of keys begin with `proxy.` are proxy module configurations. A *proxy module* is a proxy that routes abstract API calls to an actual implementation. This configuration should be set to a string which stands for a module loading path (`<package>.<classname>`). For example:

```
proxy.kvstorage='vlcp.service.connection.zookeeperdb.ZooKeeperDB'
proxy.updatenotifier='vlcp.service.connection.zookeeperdb.ZooKeeperDB'
```

Set the abstract proxy `kvstorage` and `updatenotifier` to route to `ZooKeeperDB`, thus enables ZooKeeper as the storage service provider.

Most important configurations are :*server* and connection URLs like `module.httpserver.url`. Other configurations usually can be left with their default values.

3.1.3 All Available Configurations

Note: This is a automatically generated list. Not all configurations are meant to be configured from the configuration file: some are debugging / tunning parameters; some are tended for internal usages.

- *protocol.http*
- *protocol.jsonrpc*
- *protocol.openflow*
- *protocol.ovsdb*
- *protocol.default*
- *protocol.raw*
- *protocol.redis*
- *protocol.zookeeper*

- *main*
- *module.default*
- *server*
- *module.httpserver*
- *module.jsonrpcserver*
- *module.openflowserver*
- *module.redisdb*
- *module.zookeeperdb*
- *module.console*
- *module.objectdb*
- *module.redisnotifier*
- *module.manager*
- *module.webapi*
- *module.arpresponder*
- *module.dhcpserver*
- *module.icmpresponder*
- *module.ioprocessing*
- *module.l2switch*
- *module.l3router*
- *module.openflowmanager*
- *module.ovsdbmanager*
- *module.vtepcontroller*
- *module.vxlancast*
- *module.vxlanvtep*
- *module.autoload*
- *module.knowledge*
- *module.remotecall*
- *module.session*
- *module.static*
- *jsonformat*
- *redisclient*
- *webclient*
- *zookeeperclient*
- *module.dockerplugin*
- *proxy*

protocol.http

Note: Refering *vlcp.protocol.http.Http*:

Basic HTTP/1.1 protocol Base on RFC723x, which are more strict than RFC2616

Table 1: Configurations Summary

Configuration Key	Default Value	Description
protocol.http.persist	`False`	(inherited)
protocol.http.pipelinelimit	`10`	Limit the pipelining requests that are executed in parallel. Disable pipelining by setting pipelinelimit = 1
protocol.http.headlinelimit	`8192`	Maximum length for the first line of request or response
protocol.http.headerlimit	`1048576`	Maximum length for a single header
protocol.http.headercountlimit	`1024`	Maximum header count for a single request/response
protocol.http.chunkedheaderlimit	`8192`	Maximum length for a chunked header; it is a special header type which is rarely used.
protocol.http.chunkednumberlimit	`16`	Maximum chunked header count
protocol.http.allownoheader	`True`	Enable HTTP/0.9, in which a response does not have headers
protocol.http.defaultversion	`'1.1'`	Default HTTP protocol
protocol.http.fastfail	`False`	Drop the connection rather than sending a 400 Bad Request when there is a protocol break in the request or response
protocol.http.allowtransfercompress	`False`	Enable TE: deflate header. Not all implements support this.
protocol.http.idletimeout	`20`	Close a HTTP/1.1 keep-alive connection when idles for a specified time
protocol.http.closetimeout	`5`	Close a connection when "Connection: close" is received after the specified time
protocol.http.debugging	`False`	Print debugging logs
protocol.http.useexpect	`False`	Use Expect: 100-continue for POST requests; not all server supports this
protocol.http.useexpectsize	`4096`	If useexpect = True, Use Expect: 100-continue for POST requests with data larger than this size
protocol.http.expecttimeout	`2`	When 100 Continue response is not received for the specified time, send the data
protocol.http.defaultport	`80`	default HTTP port

protocol.jsonrpc

Note: Refering *vlcp.protocol.jsonrpc.JsonRPC*:

JSON-RPC 1.0 Protocol

Table 2: Configurations Summary

Configuration Key	Default Value	Description
protocol.jsonrpc.persist	`True`	(inherited)
protocol.jsonrpc.defaultport	`6632`	This is the OVSDB default port
protocol.jsonrpc.createqueue	`True`	(inherited)
protocol.jsonrpc.debugging	`False`	Print debugging log
protocol.jsonrpc.encoding	`'utf-8'`	JSON encoding
protocol.jsonrpc.buffersize	`65536`	(inherited)
protocol.jsonrpc.messagelimit	`16777216`	Default limit a JSON message to 16MB for security purpose
protocol.jsonrpc.levellimit	`1024`	Default limit JSON scan level to 1024 levels
protocol.jsonrpc.allowedrequests	`None`	Limit the allowed request methods

protocol.openflow

Note: Refering *vlcp.protocol.openflow.openflow.Openflow*:

Openflow control protocol

Table 3: Configurations Summary

Configuration Key	Default Value	Description
protocol.openflow.persist	`True`	(inherited)
protocol.openflow.defaultport	`6653`	Default OpenFlow port
protocol.openflow.allowedversions	`(1, 4)`	Allowed versions for OpenFlow handshake; should be one or both of OFP10_VERSION and OFP13_VERSION
protocol.openflow.hellotimeout	`10`	Disconnect when OFPT_HELLO message is not received for a long time
protocol.openflow.featurerequesttimeout	`30`	Disconnect when OFPT_FEATURES_REQUEST message does not get response
protocol.openflow.keepalivetime	`10`	Send OFPT_ECHO packet when connection is idle
protocol.openflow.keepalivetimeout	`3`	When OFPT_ECHO packet does not get response in the specified time, disconnect
protocol.openflow.createqueue	`True`	(inherited)
protocol.openflow.writequeuesize	`200`	(inherited)
protocol.openflow.buffersize	`65536`	(inherited)
protocol.openflow.debugging	`False`	Show debugging messages in log
protocol.openflow.disablenxext	`False`	Disable nicira extension; when you are not using VLCP with OpenvSwitch (e.g. using with physical switches) you should set this to True
protocol.openflow.disablechainedgroup	`True`	Disable using ofp_action_group in another group: this is only supported in OpenvSwitch 2.5+, so it is disabled by default
protocol.openflow.tcp_nodelay	`True`	(inherited)

protocol.ovsdb

Note: Refering *vlcp.protocol.ovsdb.OVSDB*:

OVSDB protocol, this is a specialized JSON-RPC 1.0 protocol

Table 4: Configurations Summary

Configuration Key	Default Value	Description
protocol.ovsdb.defaultport	`6632`	(inherited)
protocol.ovsdb.allowedrequests	`('echo',)`	Only accept "echo" requests
protocol.ovsdb.keepalivetime	`10`	Send "echo" requests when connection is idle
protocol.ovsdb.keepalivetimeout	`3`	Disconnect when reply is not received for "echo" requests
protocol.ovsdb.tcp_nodelay	`True`	(inherited)

protocol.default

Note: Refering *vlcp.protocol.protocol.Protocol*:

Protocol base class

Table 5: Configurations Summary

Configuration Key	Default Value	Description
protocol.default.messagepriority	400	Message event priority for this protocol
protocol.default.writepriority	600	Data write event priority for this protocol
protocol.default.createqueue	**False**	Create separated queues for data write events from each connection
protocol.default.cleanuptimeout	60	Wait before cleanup the created queues for each connection
protocol.default.writequeuesize	10	Data write event queue size for each connection
protocol.default.messagequeuesize	10	Message event queue size for each connection
protocol.default.keepalivetime	**None**	Enable keep-alive for this protocol: send protocol specified keep-alive packages when no data is read from the connection to detect the connection liveness
protocol.default.writekeepalivetime	**None**	Send protocol specified keep-alive packages when no data is written to the connection
protocol.default.reuseport	**False**	Use SO_REUSEPORT socket option for the connections, so that multiple processes can bind to the same port; can be used to create load-balanced services
protocol.default.persist	**False**	This protocol should automatically reconnect when the connection is disconnected unexpectedly
protocol.default.buffersize	4096	Default read buffer size for this protocol. When the buffer is not large enough to contain a single message, the buffer will automatically be enlarged.
protocol.default.connect_timeout	30	Connect timeout for this protocol
protocol.default.tcp_nodelay	**False**	Enable TCP_NODELAY option for this protocol
protocol.default.sslversion	<_ ↪SSLMethod. ↪PROTOCOL_ ↪SSLv23: ␣ ↪2>	Enabled SSL version, default to PROTOCOL_SSLv23, configure it to a TLS version for more security
protocol.default.listen_persist	**True**	Server socket should retry listening if failed to bind to the specified address
protocol.default.retrylisten_interval	3	Retry interval for listen
protocol.default.backlogsize	2048	Default listen backlog size

protocol.raw

Note: Refering *vlcp.protocol.raw.Raw*:

Raw protocol, provide two streams for input and output

Table 6: Configurations Summary

Configuration Key	Default Value	Description
protocol.raw.persist	**False**	(inherited)
protocol.raw.defaultport	0	(inherited)
protocol.raw.createqueue	**True**	(inherited)
protocol.raw.buffering	**False**	Enable/disable buffering for the output stream. It is dangerous to use buffering in output stream because small amount of data might stay in buffer and not be sent
protocol.raw.writebufferlimit	4096	(inherited)
protocol.raw.splitsize	1048576	Split very large data to chunks to balance the output streaming

protocol.redis

Note: Refering *vlcp.protocol.redis.Redis*:

Redis (RESP) Protocol

Table 7: Configurations Summary

Configuration Key	Default Value	Description
protocol.redis.persist	`True`	(inherited)
protocol.redis.defaultport	`6379`	Default Redis server port
protocol.redis.createqueue	`True`	(inherited)
protocol.redis.messagequeuesize	`4096`	Limit response messages queue size
protocol.redis.bulklimit	`67108864`	Default limit Redis bulk string to 64MB
protocol.redis.levellimit	`128`	Default limit Redis array level to 128 levels
protocol.redis.encoding	`'utf-8'`	Default encoding when using Unicode strings in Redis commands
protocol.redis.hiredis	`True`	Use hiredis if possible
protocol.redis.keepalivetime	`10`	Send PING command when the connection is idle
protocol.redis.keepalivetimeout	`5`	Disconnect when PING command does not have response
protocol.redis.connect_timeout	`5`	Connect timeout
protocol.redis.tcp_nodelay	`True`	(inherited)

protocol.zookeeper

Note: Refering *vlcp.protocol.zookeeper.ZooKeeper*:

ZooKeeper protocol

Table 8: Configurations Summary

Configuration Key	Default Value	Description
protocol.zookeeper.persist	**False**	Usually we should connect to another server, this is done by ZooKeeperClient
protocol.zookeeper.defaultport	2181	default ZooKeeper port
protocol.zookeeper.createqueue	**True**	(inherited)
protocol.zookeeper.buffersize	524288	(inherited)
protocol.zookeeper.messagequeuesize	1024	Limit the response and watcher queue size
protocol.zookeeper.keepalivesend	**None**	Send ping command when the connection is idle
protocol.zookeeper.writekeepalive	5	(inherited)
protocol.zookeeper.keepalivediscount	10	Disconnect when the ping command does not receive response
protocol.zookeeper.connect_timeout	5	(inherited)
protocol.zookeeper.tcp_nodelay	**True**	(inherited)
protocol.zookeeper.writequeuesize	1024	Limit the data write queue size

main

Note: Refering *vlcp.scripts.script.ScriptModule*:

Base script module

Table 9: Configurations Summary

Configuration Key	Default Value	Description
main.args	`()`	This is not meant to be configured in a configuration file; this is done by the command line executor
main.kwargs	`{}`	This is not meant to be configured in a configuration file; this is done by the command line executor

module.default

Note: Refering *vlcp.server.module.Module*:

A functional part which can be loaded or unloaded dynamically

Table 10: Configurations Summary

Configuration Key	Default Value	Description
module.default.service	**`False`**	Service modules are automatically unloaded if all the dependencies are unloaded
module.default.forcestop	**`True`**	Force stop all the RoutineContainers that are started by this module. Subroutines are not stopped.
module.default.autosuccess	**`True`**	Automatically change module status to SUCCEEDED when load() is finished

server

Note: Refering *vlcp.server.server.Server*:

Create a server with all necessary parts

Table 11: Configurations Summary

Configuration Key	Default Value	Description
server.startup	`()`	Startup modules list. Should be a tuple of "package.classname" like: `('vlcp.service.sdn.viperflow.ViperFlow', 'vlcp.service.sdn.vrouterapi.VRouterApi')` Startup modules automatically load their dependencies, so it is not necessary (though not an error) to write them explicitly. If server.startup is null or empty, server tries to load modules in __main__.

Continued on next page

Table 11 – continued from previous page

Configuration Key	Default Value	Description
server.debugging	`False`	enable debugging log for scheduler
server.pollwritepriority	700	File-can-write event priority, usually means socket send() can be used
server.connectionwritepriority	600	ConnectionWrite events priority
server.pollreadpriority	500	File-can-read event priority, usually means data received from socket
server.pollerrorpriority	800	error event priority, usually means the socket is in an error status
server.resolverresppriority	490	responses from resolver
server.resolverreqpriority	650	requests to resolver
server.connectioncontrolpriority	450	shutdown/reset/restart commands for connections
server.routinecontrolpriority	1000	asynchronously starts a routine
server.streamdatapriority	640	streams (vlcp.event.stream.Stream) data
server.timerpriority	900	timers
server.lockpriority	990	a lock can be acquired
server.futurepriority	989	future objects been set
server.moduleloadeventpriority	890	a module is loaded/unloaded
server.sysctlpriority	2000	system high-priority events

Continued on next page

Table 11 – continued from previous page

Configuration Key	Default Value	Description
server.sysctllowpriority	10	system low-priority events
server.moduleapicallpriority	630	module API call
server.moduleapireplypriority	420	module API response
server.modulenotifypriority	410	module notify
server.totalwritelimit	100000	default connection write queue limit for all connections
server.writelimitperconnection	5	connection write events limit per connection
server.preservefornew	100	preserve space for newly created connections in default connection write queue
server.streamdatalimit	100000	stream data limit for all streams
server.datalimitperstream	5	stream data limit per stream
server.resolverpoolsize	64	the default multi-threading resolver pool size
server.ulimitn	32768	try to set open files limit (ulimit -n) to this number if it is smaller
server.logging	**None**	Use logging.config.dictConfig with this dictionary to configure the logging system. It is supported in Python 2.7+
server.loggingconfig	**None**	Use logging.config.fileConfig with this file to configure the logging system.
server.processevents	**None**	Force polling the sockets even if there are still unfinished events after processing this number of events. May be helpful for very high stress. Should be set together with server.queuemaxsize or/and server.queuedefaultsize to prevent memory overflow.
server.queuedefaultsize	**None**	Limit the default queue size, so more events will be blocked until some events are processed. This further limit the processing speed for producers to keep the system stable on very high stress, but may slightly reduce the performance.

Continued on next page

Table 11 – continued from previous page

Configuration Key	Default Value	Description
server.queuemaxsize	`None`	Limit the total size of the event queue, so more events will be blocked until some events are processed. This further limit the processing speed for producers to keep the system stable on very high stress, but may slightly reduce the performance.

module.httpserver

Note: Refering *vlcp.service.connection.httpserver.HttpServer*:

Create HTTP server on specified URLs, vHosts are supported.

Table 12: Configurations Summary

Configuration Key	Default Value	Description
module.httpserver.url	`'tcp:///'`	Default server URL. If there are multiple endpoints to listen, use .urls configuration instead of .url, it should be a list of endpoint URLs. The URL (and also any "connection URL" in other part of VLCP) should be like: `<protocol>://[<address>[:port]]/` For server socket (the listening part), the protocol should be tcp(or ltcp), ssl(or lssl) The address should be the binding address of the server. For client sockets (the connecting part), the protocol should be tcp, udp, ssl, ptcp, pudp, pssl The ptcp, pudp, pssl protocols are passive connections: they start listen and accept only one socket. The address (if specified) should be the IP address of the remote endpoint. The tcp, udp, ssl protocols are normal connections which begins by connecting to the specified address. If .urls and .url are both specified, they are both used. For SSL connections, use .key, .certificate, .ca_certs configurations to specify private key, public key, CA files for the SSL connection. If .ca_certs is not specified, the SSL certificate is not verified which may introduce security holes. the .protocol configuration node may be used to override the global protocol configurations, e.g. use: `module.redisdb.protocol.connect_timeout = 20` to override the protocol.redis.connect_timeout configuration Any TCPServer module can use .vhost node to create multiple servers for different uses. For example, use: `module.httpserver.url='tcp://localhost:80/'` `module.httpserver.vhost.api.url='tcp://localhost:8080/'` `module.httpserver.vhost.api.protocol.fastfail=True` to create two different HTTP servers, potentially with different configurations. Some modules can be configured to bind to specified vhosts. All configurations for the default vhost ('') can be used on a vhost, like .urls and .protocol and even .vhost. Vhosts inherits the settings from the parent node (the default server or another vhost) except URLs.
module.httpserver.connmanage	`False`	If True, the incoming connections are automatically managed, and can be queried with *getconnections* API. Also when the module is unloaded, the incoming connections are closed.
module.httpserver.client	`False`	By default the .url and .urls are recognized as server URLs, so it creates listening sockets. use client=True to make this module recognize the URLs as client URLs, so it creates client connections (either normal or passive)

module.jsonrpcserver

Note: Refering *vlcp.service.connection.jsonrpcserver.JsonRPCServer*:

Create JsonRPC server on specified URLs, vHosts are supported.

Table 13: Configurations Summary

Configuration Key	Default Value	Description
module.jsonrpcserver.url	`'tcp:///'`	Default server URL. If there are multiple endpoints to listen, use .urls configuration instead of .url, it should be a list of endpoint URLs. The URL (and also any "connection URL" in other part of VLCP) should be like: `<protocol>://[<address>[:port]]/` For server socket (the listening part), the protocol should be tcp(or ltcp), ssl(or lssl) The address should be the binding address of the server. For client sockets (the connecting part), the protocol should be tcp, udp, ssl, ptcp, pudp, pssl The ptcp, pudp, pssl protocols are passive connections: they start listen and accept only one socket. The address (if specified) should be the IP address of the remote endpoint. The tcp, udp, ssl protocols are normal connections which begins by connecting to the specified address. If .urls and .url are both specified, they are both used. For SSL connections, use .key, .certificate, .ca_certs configurations to specify private key, public key, CA files for the SSL connection. If .ca_certs is not specified, the SSL certificate is not verified which may introduce security holes. the .protocol configuration node may be used to override the global protocol configurations, e.g. use: `module.redisdb.protocol.connect_timeout = 20` to override the protocol.redis.connect_timeout configuration Any TCPServer module can use .vhost node to create multiple servers for different uses. For example, use: `module.httpserver.url='tcp://localhost:80/'` `module.httpserver.vhost.api.url='tcp://localhost:8080/'` `module.httpserver.vhost.api.protocol.fastfail=True` to create two different HTTP servers, potentially with different configurations. Some modules can be configured to bind to specified vhosts. All configurations for the default vhost ('') can be used on a vhost, like .urls and .protocol and even .vhost. Vhosts inherits the settings from the parent node (the default server or another vhost) except URLs.
module.jsonrpcserver.connmanage	`False`	Enable connection management
module.jsonrpcserver.client	`False`	By default the .url and .urls are recognized as server URLs, so it creates listening sockets. use client=True to make this module recognize the URLs as client URLs, so it creates client connections (either normal or passive)

module.openflowserver

Note: Refering *vlcp.service.connection.openflowserver.OpenflowServer*:

Create OpenFlow server on specified URLs, vHosts are supported.

Table 14: Configurations Summary

Configuration Key	Default Value	Description
module.openflowserver.url	`'tcp:///'`	Default server URL. If there are multiple endpoints to listen, use .urls configuration instead of .url, it should be a list of endpoint URLs. The URL (and also any "connection URL" in other part of VLCP) should be like: `<protocol>://[<address>[:port]]/` For server socket (the listening part), the protocol should be tcp(or ltcp), ssl(or lssl) The address should be the binding address of the server. For client sockets (the connecting part), the protocol should be tcp, udp, ssl, ptcp, pudp, pssl The ptcp, pudp, pssl protocols are passive connections: they start listen and accept only one socket. The address (if specified) should be the IP address of the remote endpoint. The tcp, udp, ssl protocols are normal connections which begins by connecting to the specified address. If .urls and .url are both specified, they are both used. For SSL connections, use .key, .certificate, .ca_certs configurations to specify private key, public key, CA files for the SSL connection. If .ca_certs is not specified, the SSL certificate is not verified which may introduce security holes. the .protocol configuration node may be used to override the global protocol configurations, e.g. use: `module.redisdb.protocol.connect_timeout = 20` to override the protocol.redis.connect_timeout configuration Any TCPServer module can use .vhost node to create multiple servers for different uses. For example, use: `module.httpserver.url='tcp://localhost:80/'` `module.httpserver.vhost.api.url='tcp://localhost:8080/'` `module.httpserver.vhost.api.protocol.fastfail=True` to create two different HTTP servers, potentially with different configurations. Some modules can be configured to bind to specified vhosts. All configurations for the default vhost ('') can be used on a vhost, like .urls and .protocol and even .vhost. Vhosts inherits the settings from the parent node (the default server or another vhost) except URLs.
module.openflowserver.connmanage	`False`	Enable connection management
module.openflowserver.client	`False`	By default the .url and .urls are recognized as server URLs, so it creates listening sockets. use client=True to make this module recognize the URLs as client URLs, so it creates client connections (either normal or passive)

module.redisdb

Note: Refering *vlcp.service.connection.redisdb.RedisDB*:

Create redis clients to connect to redis server

Table 15: Configurations Summary

Configuration Key	Default Value	Description
module.redisdb.url	`'tcp:///'`	Default Redis connection URL
module.redisdb.connmanage	**`False`**	If True, the incoming connections are automatically managed, and can be queried with *getconnections* API. Also when the module is unloaded, the incoming connections are closed.
module.redisdb.db	**`None`**	Default database selected
module.redisdb.serialize	`'json'`	Default serialization protocol: should be "json" or "pickle"
module.redisdb.deflate	**`True`**	Use deflate to compress the serialized data
module.redisdb.pickleversion	`'default'`	Pickling version: should be a number or "default"/"highest"
module.redisdb.cpickle	**`True`**	Allow using cPickle
module.redisdb.enterseq	`20`	Serialize requests on the same key in this process if a transact rolls back too many times
module.redisdb.enterlock	`50`	Try to use locks on the same key for all VLCP processes if a transact rolls back too many times
module.redisdb.maxretry	`160`	Maximum retry times for a transact

module.zookeeperdb

Note: Refering *vlcp.service.connection.zookeeperdb.ZooKeeperDB*:

Create zookeeper clients to connect to redis server

Table 16: Configurations Summary

Configuration Key	Default Value	Description
module.zookeeperdb.url	`'tcp:///'`	This URL is special comparing to other connection URLs, it should be like: `zk://server1[:port1],server2[:port2],server3[:port3]/[chroot]` Firstly there may be multiple hosts (and ports) in the connection URL, they form a ZooKeeper cluster; secondly, there can be a "chroot" path in the URL path. If "chroot" path apears, ZooKeeperDB uses a child node as the root node, makes it easy to host multiple services in the same ZooKeeper cluster.
module.zookeeperdb.connmanage	**False**	If True, the incoming connections are automatically managed, and can be queried with *getconnections* API. Also when the module is unloaded, the incoming connections are closed.
module.zookeeperdb.serialize	`'json'`	Default serialization protocol, "json" or "pickle"
module.zookeeperdb.deflate	**True**	Use deflate to compress the serialized data
module.zookeeperdb.pickleversion	`'default'`	Pickle version, either a number or "default"/"highest"
module.zookeeperdb.cpickle	**True**	Use cPickle if possible
module.zookeeperdb.autosuccess	**False**	Disable autosuccess; this module is loaded successfully after it initialized the ZooKeeper node.
module.zookeeperdb.kvdbvhosts	**None**	Only enable ZooKeeper KVDB service on specified vHosts, this service uses ZooKeeper in a special way. Other vHosts may be used as normal ZooKeeper clients without extra assumption.
module.zookeeperdb.notifiervhost	`''`	Notifier service vHost binding
module.zookeeperdb.singlecastlimit	`32`	If a notification contains more than .singlecastlimit keys, do not replicate the notification to all channels; only broadcast it once in the public channel.
module.zookeeperdb.notifierdeflate	**False**	Use an extra deflate compression on notification, should not be necessary if you already have .deflate=True

module.console

Note: Refering *vlcp.service.debugging.console.Console*:

VLCP debugging console.

Besides the normal functions of Python interactive console, Following variables are provided for debugging purpose:

server, manager, container

Following functions can be used to control VLCP running:

callapi, capture, sendevent, subroutine, execute, breakpoint, syscall, resume, debug, restore_console, console_help

For details call console_help()

Table 17: Configurations Summary

Configuration Key	Default Value	Description
module.console.startinconsole	**False**	Directly start VLCP in the console mode. By default, the console module creates a telnet server and wait for a connection. The console can be used in the telnet session. With startinconsole = True, the module uses stdin/stdout to create the console.
module.console.telnetconsole	`'ptcp://localhost:9923/'`	Default telnet connection URL, this is a passive connection on port 9923, so use: `telnet localhost 9923` to connect to the console.
module.console.key	**None**	If SSL is configured (with pssl://...), specify the private key file
module.console.certificate	**None**	If SSL is configured, specify the certificate file
module.console.ca_certs	**None**	If SSL is configured, specify the CA file

module.objectdb

Note: Refering *vlcp.service.kvdb.objectdb.ObjectDB*:

Abstract transaction layer for KVDB

Table 18: Configurations Summary

Configuration Key	Default Value	Description
module.objectdb.objectupdatepriority	450	Priority for object update event
module.objectdb.debuggingupdater	**False**	Enable debugging mode for updater: all updaters will be called for an extra time to make sure it does not crash with multiple calls

module.redisnotifier

Note: Refering *vlcp.service.kvdb.redisnotifier.RedisNotifier*:

Update notification with Redis Pub/Sub

Table 19: Configurations Summary

Configuration Key	Default Value	Description
module.redisnotifier.vhostbind	`''`	Bind to RedisDB vHost
module.redisnotifier.prefix	`'vlcp.updatenotifier.'`	Use this prefix for subscribe channels
module.redisnotifier.singlecastlimit	`256`	If a notification contains more than .singlecastlimit keys, do not replicate the notification to all channels; only broadcast it once in the public channel.
module.redisnotifier.deflate	**`False`**	Use an extra deflate compression on notification, should not be necessary if you already have module.redisdb.deflate=True

module.manager

Note: Refering *vlcp.service.manage.modulemanager.Manager*:

Manage module loading/unloading. Optionally reload a module when modified.

Table 20: Configurations Summary

Configuration Key	Default Value	Description
module.manager.checkinterval	`5`	Check files change with this interval.
module.manager.autoreload	**`False`**	Automatically check the loaded module files, reload them if they are changed on the disk. Notice that only the file contains the Module class is reloaded, other files will not be reloaded automatically. You should reload them manually with *reloadmodules* API if necessary.

module.webapi

Note: Refering *vlcp.service.manage.webapi.WebAPI*:

Call module API from web. Free access to any module APIs may create serious security problems, make sure to configure this module properly.

Table 21: Configurations Summary

Configuration Key	Default Value	Description
module.webapi.vhostbind	`'api'`	Default bind HttpServer vHost to 'api', so you should use: `module.httpserver.vhost.api.url='http://localhost:8080/'` to create the API endpoint
module.webapi.hostbind	**None**	Bind to a specified Host (HTTP "Host: " header)
module.webapi.rootpath	`'/'`	Bind API endpoint under this path. Each endpoint would be <root-path>/<targetname>/<methodname>. e.g. Manager.reloadmodules would be: `http://<serverurl>/<rootpath>/manager/reloadmodules` targetname and methodnames are always in lower case
module.webapi.acceptmethods	`[b'GET', b'POST']`	Allowed HTTP method, GET/POST or both
module.webapi.acceptjson	**True**	Allow to use JSON format for POST data
module.webapi.authtarget	`'public'`	If authenticate is enabled, WebAPI module uses an extra API call to authenticate the request. This is the target name.
module.webapi.authmethod	**None**	If authenticate is enabled, WebAPI module uses an extra API call to authenticate the request. This is the method name. The params will be {'env': *env*, 'target-name':*targetname*, 'name': *methodname*, 'params': *parameters*}
module.webapi.allowtargets	**None**	Only allow API calls on specified targets (usually a target name is the lower-cased module name)
module.webapi.denytargets	**None**	Disallow API calls on specified targets
module.webapi.typeextension	**True**	Enable a type extension for GET/POST methods: use '' quoted string to represent a Python literal expression like lists, tuples, dictionaries, numbers etc.
module.webapi.errordetails	**True**	Show error details in error responses with JSON format {"error":...}. If errordetails = False, the default HTTP error response is used.
module.webapi.errortrace	**False**	Also output the trace information in the JSON output

module.arpresponder

Note: Refering *vlcp.service.sdn.arpresponder.ARPResponder*:

Send ARP respond

Table 22: Configurations Summary

Configuration Key	Default Value	Description
module.arpresponder.vhostbind	`['']`	Default binding to OpenFlow vHosts
module.arpresponder.prepush	**`True`**	Prepush ARP entries with Flows, so the switch directly responds an ARP request
module.arpresponder.broadcastonly	**`True`**	When using prepush=True, only responds a broadcast ARP request; let unicast ARP request reach the other side
module.arpresponder.disablenxprepush	**`False`**	Drop ARP requests with unknown IP addresses. The ports will not be able to access external IPs, so use with caution.

module.dhcpserver

Note: Refering *vlcp.service.sdn.dhcpserver.DHCPServer*:

DHCP server that responds the DHCP discover/request with static IP address settings

Table 23: Configurations Summary

Configuration Key	Default Value	Description
module.dhcpserver.vhostbind	`['']`	Default binding to OpenFlow vHosts
module.dhcpserver.serveraddress	`'169.254.169.254'`	Responding DHCP server address
module.dhcpserver.servermac	`'1a:23:67:59:63:33'`	Responding DHCP server MAC address
module.dhcpserver.leasetime	**`None`**	DHCP leases timeout time
module.dhcpserver.t1	**`None`**	DHCP default T1 option
module.dhcpserver.t2	**`None`**	DHCP default T2 option

module.icmpresponder

Note: Refering *vlcp.service.sdn.icmpresponder.ICMPResponder*:

Respond ICMP echo (ping) requests to the gateway

Table 24: Configurations Summary

Configuration Key	Default Value	Description
module.icmpresponder.vhostbind	`['']`	Default binding to OpenFlow vHosts
module.icmpresponder.prepush	**`False`**	True : reply icmp ping with flow False: reply icmp ping with controller PACKET_IN/PACKET_OUT Must use prepush=True with OpenvSwitch 2.5+
module.icmpresponder.inroutermac	`'1a:23:67:59:63:33'`	"Gateway" responds with this MAC address

module.ioprocessing

Note: Refering *vlcp.service.sdn.ioprocessing.IOProcessing*:

Ingress and Egress processing

Table 25: Configurations Summary

Configuration Key	Default Value	Description
module.ioprocessing.vhostbind	`['']`	Default binding to OpenFlow vHosts
module.ioprocessing.vhostmap	`{}`	vHost map from OpenFlow vHost to OVSDB vHost. If the OpenFlow vHost is not found in this map, it will map to the default OVSDB vHost ('')
module.ioprocessing.enable_router_forward	**`False`**	Enable forwarding in this server, so it becomes a forwarding node (also known as a N/S gateway)

module.l2switch

Note: Refering *vlcp.service.sdn.l2switch.L2Switch*:

L2 switch functions

Table 26: Configurations Summary

Configuration Key	Default Value	Description
module.l2switch.vhostbind	`['']`	Default binding to OpenFlow vHosts
module.l2switch.learning	**`False`**	Use learning or prepush. Learning strategy creates flows on necessary, but may cause some delay for the first packet; prepush push all necessary flows from beginning.
module.l2switch.nxlearn	**`True`**	When using learning=True, enable OpenvSwitch learn() action. if nxlearn=False, the learning strategy will be done with controller PACKET_IN processing, which may be less efficient.
module.l2switch.learntimeout	`300`	Timeout for a learned flow

module.l3router

Note: Refering *vlcp.service.sdn.l3router.L3Router*:

L3 connectivities with virtual router.

Table 27: Configurations Summary

Configuration Key	Default Value	Description
module.l3router.vhostbind	`['']`	Default binding to OpenFlow vHosts
module.l3router.inroutermac	`↪'1a:23:67:59:63:33↪'`	Router responding MAC address for logical ports on this switch
module.l3router.outroutermacmask	`↪'0a:00:00:00:00:00↪'`	Router responding MAC address mask for outside network. The MAC address is formed with the physical MAC address (NIC MAC address) XORed with this mask
module.l3router.arp_cycle_time	`5`	Retry ARP requests with this interval when there is no respond
module.l3router.prepush	**`False`**	Prepush ARP entries for all the logical ports which are accessible from the router
module.l3router.arp_incomplete_timeout	`60`	if arp entry have no reply , it will send in arp cycle until timeout but if new packet request arp ,, it will flush this timeout in arp entry
module.l3router.arp_complete_timeout	`30`	An ARP entry stays in “COMPLETE” state without sending further ARP requests until this time
module.l3router.buffer_packet_timeout	`30`	The L3 gateway buffers a packet and wait for ARP responds until this time
module.l3router.static_host_arp_refresh_interval	`60`	Refresh interval (external gateway address) ARP (MAC-IP corresponding)
module.l3router.enable_router_forward	**`False`**	Enable forwarding in this server, so it becomes a forwarding node (also known as a N/S gateway) This should be set together with module.ioprocessing.enable_router_forward
module.l3router.addressinfo_autorenew_time	`150`	A forwarding node will acquire a IP address from an external network, and refresh the information to keep the acquire state. This is the refresh interval.
module.l3router.forwardinfo_autorenew_time	`15`	A forwarding node will acknowledge other nodes that it is ready to forward network traffic from other nodes, this is the fresh interval

module.openflowmanager

Note: Refering *vlcp.service.sdn.ofpmanager.OpenflowManager*:

Manage Openflow Connections

Table 28: Configurations Summary

Configuration Key	Default Value	Description
module.openflowmanager.vhostbind	(inherited) **None**	

module.ovsdbmanager

Note: Refering *vlcp.service.sdn.ovsdbmanager.OVSDBManager*:

Manage Openflow Connections

Table 29: Configurations Summary

Configuration Key	Default Value	Description
module.ovsdbmanager.vhostbind	**None**	Bind to JsonRPCServer vHosts. If not None, should be a list of vHost names e.g. `['']`
module.ovsdbmanager.bridgenames	**None**	Only acquire information from bridges with this names

module.vtepcontroller

Note: Refering *vlcp.service.sdn.vtepcontroller.VtepController*:

Controll a physical switch which supports OVSDB hardware_vtep protocol.

Table 30: Configurations Summary

Configuration Key	Default Value	Description
module.vtepcontroller.vhostbind	['vtep']	Default bind controller to OVSDB vHost
module.vtepcontroller.recycleinterval	300	Recycle unused logical switches from hardware_vtep OVSDB
module.vtepcontroller.refreshinterval	3600	Every logical network has a broadcast list which contains all related server nodes, this is the refresh interval to keep current node in the list
module.vtepcontroller.allowedmigratetime	120	When a logical port is migrating from one node to another, there may be multiple instances of the logical port on different or same nodes. This is the maximum time allowed for the migration.
module.vtepcontroller.masterlock	'vlcp_vtepcontroller_masterlock'	When there are multiple controllers started for the same hardware_vtep OVSDB, every instance will try to acquire this lock in the OVSDB, and the one who acquired the lock will be the master.
module.vtepcontroller.prepush	True	Prepush unicast information for MACs on other nodes. If this is not set, the physical switch should support and enable MAC-learning on VXLAN.

module.vxlancast

Note: Refering *vlcp.service.sdn.vxlancast.VXLANCast*:

VXLAN single-cast and broadcast functions

Table 31: Configurations Summary

Configuration Key	Default Value	Description
module.vxlancast.vhostbind	`['']`	Default binding to OpenFlow vHosts
module.vxlancast.learning	**`True`**	Enable learning on VXLAN tunnel IP destination. VXLAN model must know every destination for each unicast MAC address. There are three working mode the VXLANCast module: 1. prepush = True, necessary MAC-tunneldst information will be pushed to the logical switch. learning = True may also be enabled so that external VXLAN MAC addresses may use learning strategy. If learning = False, unknown MAC addresses cannot be forwarded. 2. learning = True, prepush = False. When a packet is received from a destination, its destination tunnel IP address is recorded, a flow is created for this MAC address; every packet with an unknown unicast MAC address as its destination will be broadcasted. 3. learning = False, prepush = False. This enables a first-packet-to-controller strategy: When the first packet is going out from the switch, it is transmitted to the controller, and the controller queries ObjectDB for the tunnel destination, creates the fast-path and retransmit the packet. This may introduce a significant delay for the first packet.
module.vxlancast.prepush	**`True`**	Enable prepush on VXLAN tunnel IP destination, the necessary MAC-tunneldst information will be written into switch
module.vxlancast.learntimeout	`300`	Learning flow timeout
module.vxlancast.packetexpire	`3`	When using first-packet-to-controller strategy, the buffered packets will only be keep for a short time before expiring
module.vxlancast.pushtimeout	`300`	When using first-packet-to-controller strategy, the created flow will be timeout after this time
module.vxlancast.vhostmap	`{}`	Mapping OpenFlow vHosts to OVSDB vHosts
module.vxlancast.refreshinterval	`3600`	Every logical network has a broadcast list which contains all related server nodes, this is the refresh interval to keep current node in the list
module.vxlancast.allowedmigratetime	`120`	When a logical port is migrating from one node to another, there may be multiple instances of the logical port on different or same nodes. This is the maximum time allowed for the migration.
module.vxlancast.grouptimeout	`120`	Timeout waiting for group creation

module.vxlanvtep

Note: Refering *vlcp.service.sdn.vxlanvtep.VXLANVtep*:

Use hardware_vtep instead of software VXLAN

Table 32: Configurations Summary

Configuration Key	Default Value	Description
module.vxlanvtep.vhostbind	`['']`	Default binding to OpenFlow vHosts
module.vxlanvtep.vlanid_pool	`((3000, 4000),)`	Use these VLANs for vtep configuration. Must not be conflicted with VLAN networks.
module.vxlanvtep.remote_api	`True`	remote_api means call remote api to vtep controller
module.vxlanvtep.retryaction	`60`	interval time to retry failed actions

module.autoload

Note: Refering *vlcp.service.utils.autoload.AutoLoad*:

Auto load some modules from a package. Usually used to load network plugins.

Table 33: Configurations Summary

Configuration Key	Default Value	Description
module.autoload.autoloadpackages	`('vlcp.service.sdn.plugins',)`	Autoload packages from some packages

module.knowledge

Note: Refering *vlcp.service.utils.knowledge.Knowledge*:

Simple KV-cache in memory. A base for other KV-DB. Data is automatically removed after timeout. Use knowledge instead of local storage in modules so data is not lost on module restarting.

Table 34: Configurations Summary

Configuration Key	Default Value	Description
module.knowledge.checkinterval	`300`	Check current data set, remove the expired data

module.remotecall

Note: Refering *vlcp.service.utils.remoteapi.RemoteCall*:

Route local API calls to remote management API.

Table 35: Configurations Summary

Configuration Key	Default Value	Description
module.remotecall.target_url_map	`{}`	URL map list for every target, should be `{target: [url, url, ...]}`

module.session

Note: Refering *vlcp.service.utils.session.Session*:

HTTP Session with cookies

Table 36: Configurations Summary

Configuration Key	Default Value	Description
module.session.timeout	`1800`	Timeout for a session
module.session.cookiename	`'_session_id'`	Cookie name used in session id storage

module.static

Note: Refering *vlcp.service.web.static.Static*:

Map specified path to local files

Table 37: Configurations Summary

Configuration Key	Default Value	Description
module.static.checkreferer	`False`	Check HTTP Referer header to protect against external site links. The Referer header must present and match allowed sites
module.static.refererallowlocal	`True`	Grant access when Referer header match Host header
module.static.refererallows	`[]`	Set allowed referers
module.static.allowrange	`True`	Respond to Range header request
module.static.etag	`True`	Generate ETag for static resources, so the browser can use a If-Not-Match request to save bandwidth
module.static.gzip	`True`	If there is a "xxxx.gz" file in the folder, it would be recognized as a "xxxx" file with "Content-Encoding: gzip". If gzip = False, it would be recognized as a normal file
module.static.maxage	`5`	Cache expiration time
module.static.memorycache	`True`	Enable a memory cache for small files to improve performance
module.static.memorycachelimit	`4096`	Maximum file size allowed to be cached in memory
module.static.memorycacheitemlimit	`4096`	Maximum cached file number. When it is exceeded, old caches will be cleared.
module.static.dir	`'static'`	The directory name for the static resource folder. Static module supports multiple directories with different configurations, just like vHosts. Use .vdir node to create separated configurations for different folders, e.g.: `module.static.vdir.rewrite.dir="rewrite"` `module.static.vdir.rewrite.rewriteonly=True` You may also specify multiple directories with .dirs configuration instead, e.g.: `module.static.dir=None` `module.static.dirs=["js","css","images","download"]` The directory name can be a relative path against a Python module file, or an absolute path. .dir and .dirs maps a HTTP GET/HEAD request with path */dir/subpath* to disk file *relativeroot/dir/subpath*. The *subpath* may be a file in sub directories. ".", ".." is also accepted as current folder/parent folder, but it always map to a path inside *relativeroot/dir*, which means: */dir/subpath*/a => *relativeroot/dir*/a */dir/subpath*/b/a => *relativeroot/dir*/b/a */dir/subpath*/b/../a => *relativeroot/dir*/a */dir/subpath*/b/../../a => *relativeroot/dir*/a So it is not possible to use a static file map to access files outside the mapped folder. If you want to map a HTTP path to a directory with different names, use .map instead
module.static.relativemodule	`'__main__'`	Specify the mapped directory is relative to a Python module file. If .relativeroot is also configured and is not empty, .relativeroot takes effect and .relativemodule is ignored. If both are not configured, the current working directory is the relative root. If you start your server with a customized Python script, the default "__main__" will fit. If

jsonformat

Note: Refering *vlcp.utils.jsonencoder.JsonFormat*:

This is an extended JSON formatter used by WebAPI module

Table 38: Configurations Summary

Configuration Key	Default Value	Description
jsonformat.namedstruct	**True**	Enable special format for namedstruct (https://pypi.python.org/pypi/nstruct)
jsonformat.humanread	**True**	Use human read format, so: namedstruct structures are formatted with readable names; very long bytes are viewed like <10000 bytes...> (Python 3 only)
jsonformat.bytesdecode	'ascii'	In Python 3, Try to decode bytes to str with this encoding
jsonformat.byteslimit	256	When bytes object is longer than this and humanread=True, show <*length* bytes...> instead of the exact content
jsonformat.dumpextra	**False**	Dump extra information for namedstruct structures
jsonformat.dumptypeinfo	'flat'	Dump type information for namedstruct structures
jsonformat.dataobject	**True**	Dump the content for data objects
jsonformat.dataattributes	**True**	Dump attributes for data objects

redisclient

Note: Refering *vlcp.utils.redisclient.RedisClientBase*:

Connect to Redis server

Table 39: Configurations Summary

Configuration Key	Default Value	Description
redisclient.url	`'tcp://localhost/'`	Default connection URL for Redis client
redisclient.timeout	`10`	Wait for the connection setup before raise an exception
redisclient.db	`0`	Select database

webclient

Note: Refering *vlcp.utils.webclient.WebClient*:

Convenient HTTP request processing. Proxy is not supported in current version.

Table 40: Configurations Summary

Configuration Key	Default Value	Description
webclient.cleanupinterval	`60`	When a cleanup task is created, the task releases dead connections by this interval
webclient.samehostlimit	`20`	Persist this number of connections at most for each host. If all connections are in use, new requests will wait until there are free connections.
webclient.sameurllimit	**`False`**	Do not allow multiple requests to the same URL at the same time. If sameurllimit=True, requests to the same URL will always be done sequential.
webclient.cafile	**`None`**	CA file used to verify HTTPS certificates. To be compatible with older Python versions, the new SSLContext is not enabled currently, so with the default configuration, the certificates are NOT verified. You may configure this to a .pem file in your system, usually /etc/pki/tls/cert.pem in Linux.
webclient.redirectlimit	`10`	When following redirects and the server redirects too many times, raises an exception and end the process
webclient.verifyhost	**`True`**	Verify the host with the host in certificate

zookeeperclient

Note: Refering *vlcp.utils.zkclient.ZooKeeperClient*:

ZooKeeper client to send requests to a cluster

Table 41: Configurations Summary

Configuration Key	Default Value	Description
zookeeperclient.serverlist	`[]`	Default ZooKeeper server list, should be a list contains connection URLs
zookeeperclient.chroot	`'/'`	Chroot to a child node instead of the root node. All paths used in the program will be mapped to *chroot_path/path*
zookeeperclient.auth	`[]`	Extra authentications, should be list of tuples [(scheme1, auth1), (scheme2, auth2), ...]
zookeeperclient.sessiontimeout	`20`	Zookeeper session timeout
zookeeperclient.rebalancetime	**`None`**	If not None, ZooKeeperClient will disconnect from the server and reconnect to a random server to make sure the connections to ZooKeeper servers are balanced. It sometimes causes problems, so it is disabled by default.

module.dockerplugin

Note: Refering *vlcp_docker.dockerplugin.DockerPlugin*:

Integrate VLCP with Docker

Table 42: Configurations Summary

Configuration Key	Default Value	Description
module.dockerplugin.vhostbind	`'docker'`	Bind Docker API EndPoint (a HTTP service) to specified vHost
module.dockerplugin.ovsbridge	`'dockerbr0'`	OpenvSwitch bridge used in this server
module.dockerplugin.vethprefix	`'vlcp'`	Auto-created veth device prefix
module.dockerplugin.ipcommand	`'ip'`	Path to `ip` command
module.dockerplugin.ovscommand	`'ovs-vsctl'`	Path to `ovs-vsctl` command
module.dockerplugin.dstprefix	`'eth'`	NIC prefix used in the docker container
module.dockerplugin.mactemplate	`'02:00:00:00:00:00'`	Template MAC address used on generating MAC addresses
module.dockerplugin.mtu	`1500`	Default MTU used for networks
module.dockerplugin.autoremoveports	`True`	Try to remove the old port if it is not clean up correctly
module.dockerplugin.pooltimeout	`60`	IPAM pool reserve timeout. If a reserved pool is not used to create a network till timeout, it is automatically released.
module.dockerplugin.iptimeout	`60`	IPAM IP reserve timeout. If an IP address is not used to create an endpoint till timeout, it is automatically released.
module.dockerplugin.cidrrange	`'10.0.0.0/8'`	The default address space used when subnet is not specified. A C-class (like 10.0.1.0/24) subnet will be assigned for each network.

proxy

Note: `proxy` is a special type of configurations that route abstract API calls to different implementations. The value of each item should be a loading path of a module `<package>.<classname>` like `vlcp.service.`

`sdn.viperflow.ViperFlow`

Table 43: Configurations Summary

Configuration Key	Default Value
proxy.updatenotifier	`'vlcp.service.kvdb.redisnotifier.RedisNotifier'`
proxy.kvstorage	`'vlcp.service.connection.redisdb.RedisDB'`
proxy.memorystorage	`'vlcp.service.utils.knowledge.Knowledge'`

3.2 Function Module List

VLCP function modules are classes that can be loaded to / unloaded from the server dynamically. With different modules loaded, VLCP can provide very different functions. Users may decide which functions they would like to use by configurating `server.startup` configuration.

3.2.1 Server Start/Stop and Modules

When VLCP server starts, all startup modules in `server.startup` are loaded into server and started. They each start their own routines to provide their services. Some modules have dependencies on other modules, the dependencies are automatically started first.

The `server.startup` configuration is a tuple of *loading paths* of modules. A module is actually a Python class derived from `vlcp.service.module.Module`. The *loading path* of a module is the full path of the Python module file contains the class, followed by a dot and the class name, like `vlcp.service.connection.httpserver.HttpServer`. The *module name* of a module is the lower-cased class name, like `httpserver`. VLCP modules with the same *module name* cannot be loaded or used together.

Server stop when all the routines stop, this is usually when:

- All connections and server sockets are closed, and all pended jobs are done
- The server receives an end signal (SIGINT, SIGTERM) and stops all the routines
- All modules are unloaded

3.2.2 Module API

Modules provide functions through a lot of methods, the most important one of which is the **Module API**. Module APIs are methods exposed by the module to be called from other modules, or directly from the end user.

Parameters for Module APIs are always provided with keyword-arguments: arguments positions do not matter.

Call Module API from WebAPI

A management module *webapi* exposes all module APIs through a management HTTP service. Through this management function, you can call module APIs with *curl*:

```
curl -g -d 'param1=value1&param2=value2' \
   'http://localhost:8081/modulename/apiname'
```

The URL path of the request should be */modulename/apiname*, where *modulename* is the module name i.e. the lower-cased class name, and *apiname* is the API name i.e. the lower-cased method name.

By default, the management API supports HTTP GET (with query string), HTTP POST (with standard form data), and HTTP POST with JSON-format POST data. Though use the HTTP GET/POST format is usually the easiest way to call the API in Shell command-line, when integrating with other systems JSON-format POST may be more convient.

`` quoted expression is a VLCP-specified extension. Some APIs need data types other than strings for its parameters. When a string parameter is quoted by ``, VLCP recognizes it as a literal expression in Python. You may use numbers, string, tuples, list, dictionary, sets and any combinations of them in a quoted expression:

```
curl -g -d 'complexparam=`[1,2,{"name":("value",set(1,2,3))}]`' \
   'http://localhost:8081/modulename/apiname'
```

Make sure to surround the `` expression with '' to prevent it from excuting as a shell command.

Also notice that '[]' have special meanings in *curl*, that is why we use `-g` option to turn it off.

The return value of the module API is formatted with JSON-format, return with `{"result": <data...>}` format. If any error occurs, the HTTP request returns a 4xx or 5xx HTTP status, with a JSON body `{"error": "<errormessage>"}`. Exception details can be found in system logs.

Call Module API from Debugging Console

When using debugging console module *console*, you can use callapi() method to call module APIs easily in the debugging console as other modules. This method accepts keyword-arguments. for example:

```
for m in callapi(container, "objectdb", "getonce",key="xx"):
    yield m
```

will call objectdb module api getonce.

Note: debugging console module will also start telnet server on localhost:9923 you can choose telnet it when server run in daemon mode.

Module API Discovery

Every module supports a special API `discovery`. When discovery is called, a list of supported endpoints and their descriptions are returned. With an extra parameter `details=true`, it also returns information for arguments and their default values. For example, you can call:

```
curl -g 'http://localhost:8081/viperflow/discover?details=true' | python -m json.tool
```

To view the API details of module *viperflow*.

3.2.3 Dynamic Load / Unload / Reload Modules

If *manager* module is loaded, you can use its APIs to load, unload or reload modules on the fly. When reloading modules, the files containing VLCP module classes are reloaded to use the latest code on disk, so it is possible to upgrade modules without stopping the service.

Note: This should be considered as an advanced feature. Not all modules are strictly tested against reloading. Use this function with caution.

3.2.4 All Module List

These are modules currently shipped with VLCP and vlcp-docker-plugin.

Note: This is a generated list. Not all module APIs are designed to be used by end users directly. Lots of them are providing services for other modules.

- *scriptmodule*
- *httpserver*
- *jsonrpcserver*
- *openflowserver*
- *redisdb*
- *zookeeperdb*
- *console*
- *objectdb*
- *redisnotifier*
- *manager*
- *webapi*
- *arpresponder*
- *dhcpserver*
- *icmpresponder*
- *ioprocessing*
- *l2switch*
- *l3router*
- *openflowmanager*
- *openflowportmanager*
- *ovsdbmanager*
- *ovsdbportmanager*
- *networklocaldriver*
- *networknativedriver*
- *networkvlandriver*
- *networkvxlandriver*

- *viperflow*
- *vrouterapi*
- *vtepcontroller*
- *vxlancast*
- *vxlanvtep*
- *autoload*
- *knowledge*
- *remotecall*
- *session*
- *static*
- *dockerplugin*

scriptmodule

Loading Path	`vlcp.scripts.script.ScriptModule`
Class Reference	*`vlcp.scripts.script.ScriptModule`*
Dependencies	(None)

Base script module

httpserver

Loading Path	`vlcp.service.connection.httpserver.HttpServer`
Class Reference	*`vlcp.service.connection.httpserver.HttpServer`*
Dependencies	(None)

Create HTTP server on specified URLs, vHosts are supported.

API List

`httpserver/getservers (vhost=None)`
Return current servers

Parameters **`vhost`** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

`httpserver/stoplisten (vhost=None)`
Stop listen on current servers

Parameters **`vhost`** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

`httpserver/startlisten (vhost=None)`
Start listen on current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

httpserver/updateconfig ()
Reload configurations, remove non-exist servers, add new servers, and leave others unchanged

httpserver/getconnections (vhost=None)
Return accepted connections, optionally filtered by vhost

jsonrpcserver

Loading Path	`vlcp.service.connection.jsonrpcserver.JsonRPCServer`
Class Reference	*`vlcp.service.connection.jsonrpcserver.JsonRPCServer`*
Dependencies	(None)

Create JsonRPC server on specified URLs, vHosts are supported.

API List

jsonrpcserver/getservers (vhost=None)
Return current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

jsonrpcserver/stoplisten (vhost=None)
Stop listen on current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

jsonrpcserver/startlisten (vhost=None)
Start listen on current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

jsonrpcserver/updateconfig ()
Reload configurations, remove non-exist servers, add new servers, and leave others unchanged

jsonrpcserver/getconnections (vhost=None)
Return accepted connections, optionally filtered by vhost

openflowserver

Loading Path	`vlcp.service.connection.openflowserver.OpenflowServer`
Class Reference	*`vlcp.service.connection.openflowserver.OpenflowServer`*
Dependencies	(None)

Create OpenFlow server on specified URLs, vHosts are supported.

API List

openflowserver/getservers (vhost=None)
Return current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

openflowserver/stoplisten (vhost=None)
Stop listen on current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

openflowserver/startlisten (vhost=None)
Start listen on current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

openflowserver/updateconfig ()
Reload configurations, remove non-exist servers, add new servers, and leave others unchanged

openflowserver/getconnections (vhost=None)
Return accepted connections, optionally filtered by vhost

redisdb

Loading Path	`vlcp.service.connection.redisdb.RedisDB`
Class Reference	*`vlcp.service.connection.redisdb.RedisDB`*
Dependencies	(None)

Create redis clients to connect to redis server

API List

redisdb/getservers (vhost=None)
Return current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

redisdb/stoplisten (vhost=None)
Stop listen on current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

redisdb/startlisten (vhost=None)
Start listen on current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

redisdb/updateconfig ()
Reload configurations, remove non-exist servers, add new servers, and leave others unchanged

redisdb/getconnections (vhost=None)
Return accepted connections, optionally filtered by vhost

redisdb/getclient (vhost='')
Return a tuple of `(redisclient, encoder, decoder)` for specified vhost

redisdb/get (key, timeout=None, vhost='')
Get value from key

redisdb/set (key, value, timeout=None, vhost='')
Set value to key, with an optional timeout

redisdb/delete (key, vhost='')
Delete a key from the storage

redisdb/mget (keys, vhost='')
Get multiple values from multiple keys

redisdb/mgetwithcache (keys, vhost='', cache=None)
Get multiple values, cached when possible

redisdb/mset (kvpairs, timeout=None, vhost='')
Set multiple values on multiple keys

redisdb/update (key, updater, timeout=None, vhost='')
Update in-place with a custom function

Parameters

- **key** – key to update
- **updater** – `func(k,v)`, should return a new value to update, or return None to delete. The function may be call more than once.
- **timeout** – new timeout

Returns the updated value, or None if deleted

redisdb/mupdate (keys, updater, timeout=None, vhost='')
Update multiple keys in-place with a custom function, see update. Either all success, or all fail.

redisdb/updateall (keys, updater, timeout=None, vhost='')
Update multiple keys in-place, with a function `updater(keys, values)` which returns `(updated_keys, updated_values)`.

Either all success or all fail

redisdb/updateallwithtime (keys, updater, timeout=None, vhost='')
Update multiple keys in-place, with a function `updater(keys, values, timestamp)` which returns `(updated_keys, updated_values)`.

Either all success or all fail.

Timestamp is a integer standing for current time in microseconds.

redisdb/listallkeys (vhost='')
Return all keys in the KVDB. For management purpose.

zookeeperdb

Loading Path	`vlcp.service.connection.zookeeperdb.ZooKeeperDB`
Class Reference	*vlcp.service.connection.zookeeperdb.ZooKeeperDB*
Dependencies	(None)

Create zookeeper clients to connect to redis server

API List

zookeeperdb/getservers (vhost=None)
Return current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

zookeeperdb/stoplisten (vhost=None)
Stop listen on current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

zookeeperdb/startlisten (vhost=None)
Start listen on current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

zookeeperdb/updateconfig ()
Reload configurations, remove non-exist servers, add new servers, and leave others unchanged

zookeeperdb/getconnections (vhost=None)
Return accepted connections, optionally filtered by vhost

zookeeperdb/getclient (vhost='')
Return a tuple of `(zookeeperclient, encoder, decoder)` for specified vhost

zookeeperdb/get (key, timeout=None, vhost='')
Get value from key

zookeeperdb/set (key, value, timeout=None, vhost='')
Set value to key, with an optional timeout

zookeeperdb/delete (key, vhost='')
Delete a key from the storage

zookeeperdb/mget (keys, vhost='')
Get multiple values from multiple keys

zookeeperdb/mgetwithcache (keys, vhost='', cache=None)
Get multiple values, cached when possible

zookeeperdb/mset (kvpairs, timeout=None, vhost='')
Set multiple values on multiple keys

zookeeperdb/update (key, updater, timeout=None, vhost='')
Update in-place with a custom function

Parameters

- **key** – key to update
- **updater** – `func(k,v)`, should return a new value to update, or return None to delete. The function may be call more than once.
- **timeout** – new timeout

Returns the updated value, or None if deleted

zookeeperdb/mupdate (keys, updater, timeout=None, vhost='')
Update multiple keys in-place with a custom function, see update.

Either all success, or all fail.

zookeeperdb/updateall (keys, updater, timeout=None, vhost='')
Update multiple keys in-place, with a function `updater(keys, values)` which returns `(updated_keys, updated_values)`.

Either all success or all fail

zookeeperdb/updateallwithtime (keys, updater, timeout=None, vhost='')
Update multiple keys in-place, with a function `updater(keys, values, timestamp)` which returns `(updated_keys, updated_values)`.

Either all success or all fail.

Timestamp is a integer standing for current time in microseconds.

zookeeperdb/recycle (keys, vhost='')
Recycle extra versions from the specified keys.

zookeeperdb/createnotifier (vhost=None)
Create a new notifier object

zookeeperdb/listallkeys (vhost='')
Return all keys in the KVDB. For management purpose.

zookeeperdb/status (vhost='')

console

Loading Path	`vlcp.service.debugging.console.Console`
Class Reference	*vlcp.service.debugging.console.Console*
Dependencies	(None)

VLCP debugging console.

Besides the normal functions of Python interactive console, Following variables are provided for debugging purpose:

server, manager, container

Following functions can be used to control VLCP running:

callapi, capture, sendevent, subroutine, execute, breakpoint, syscall, resume, debug, restore_console, console_help

For details call console_help()

objectdb

Loading Path	`vlcp.service.kvdb.objectdb.ObjectDB`
Class Reference	*vlcp.service.kvdb.objectdb.ObjectDB*
Dependencies	*kvstorage* *updatenotifier*

Abstract transaction layer for KVDB

API List

objectdb/mget (keys, requestid, nostale=False)
Get multiple objects and manage them. Return references to the objects.

objectdb/get (key, requestid, nostale=False)
Get an object from specified key, and manage the object. Return a reference to the object or None if not exists.

objectdb/mgetonce (keys, nostale=False)
Get multiple objects, return copies of them. Referenced objects are not retrieved.

objectdb/getonce (key, nostale=False)
Get a object without manage it. Return a copy of the object, or None if not exists. Referenced objects are not retrieved.

objectdb/mwatch (keys, requestid, nostale=False)
Try to return all the references, see `watch()`

objectdb/watch (key, requestid, nostale=False)
Try to find an object and return a reference. Use `reference.isdeleted()` to test whether the object exists. Use `reference.wait(container)` to wait for the object to be existed.

objectdb/munwatch (keys, requestid)
Cancel management of keys

objectdb/unwatch (key, requestid)
Cancel management of a key

objectdb/unwatchall (requestid)
Cancel management for all keys that are managed by requestid

objectdb/transact (keys, updater, withtime=False, maxtime=60)
Try to update keys in a transact, with an `updater(keys, values)`, which returns `(updated_keys, updated_values)`.

The updater may be called more than once. If `withtime = True`, the updater should take three parameters: `(keys, values, timestamp)` with timestamp as the server time

objectdb/watchlist (requestid=None)
Return a dictionary whose keys are database keys, and values are lists of request ids. Optionally filtered by request id

objectdb/walk (keys, walkerdict, requestid, nostale=False)
Recursively retrieve keys with customized functions. walkerdict is a dictionary `key->walker(key, obj, walk, save)`.

objectdb/gettimestamp ()
Get a timestamp from database server

objectdb/asynctransact (asyncupdater, withtime=False, maxretry=None, maxtime=60)
Read-Write transaction with asynchronous operations.

First, the *asyncupdater* is called with *asyncupdater(last_info, container)*. *last_info* is the info from last *AsyncTransactionLockException*. When *asyncupdater* is called for the first time, last_info = None.

The async updater should be an async function, and return *(updater, keys)*. The *updater* should be a valid updater function used in *transaction* API. *keys* will be the keys used in the transaction.

The async updater can return None to terminate the transaction without exception.

After the call, a transaction is automatically started with the return values of *asyncupdater*.

updater can raise *AsyncTransactionLockException* to restart the transaction from *asyncupdater*.

Parameters

- **asyncupdater** – An async updater *asyncupdater(last_info, container)* which returns *(updater, keys)*
- **withtime** – Whether the returned updater need a timestamp
- **maxretry** – Limit the max retried times
- **maxtime** – Limit the execution time. The transaction is abandoned if still not completed after *maxtime* seconds.

objectdb/writewalk (keys, walker, withtime=False, maxtime=60)

A read-write transaction with walkers

Parameters

- **keys** – initial keys used in walk. Provide keys already known to be necessary to optimize the transaction.
- **walker** – A walker should be *walker(walk, write)*, where *walk* is a function *walk(key)->value* to get a value from the database, and *write* is a function *write(key, value)* to save value to the database.

  A value can be write to a database any times. A *walk* called after *write* is guaranteed to retrieve the previously written value.
- **withtime** – if withtime=True, an extra timestamp parameter is given to walkers, so walker should be *walker(walk, write, timestamp)*
- **maxtime** – max execution time of this transaction

objectdb/asyncwritewalk (asyncwalker, withtime=False, maxtime=60)

A read-write transaction with walker factory

Parameters

- **asyncwalker** – an async function called as *asyncwalker(last_info, container)* and returns (keys, walker), which are the same as parameters of *writewalk*

  param keys initial keys used in walk

  param walker A walker should be *walker(walk, write)*, where *walk* is a function *walk(key)->value* to get a value from the database, and *write* is a function *write(key, value)* to save value to the database.

  A value can be write to a database any times. A *walk* called after *write* is guaranteed to retrieve the previously written value.

  raise AsyncTransactionLockException in walkers to restart the transaction
- **withtime** – if withtime=True, an extra timestamp parameter is given to walkers, so walkers should be *walker(key, value, walk, write, timestamp)*
- **maxtime** – max execution time of this transaction

redisnotifier

Loading Path	`vlcp.service.kvdb.redisnotifier.RedisNotifier`
Class Reference	*vlcp.service.kvdb.redisnotifier.RedisNotifier*
Dependencies	*redisdb*

Update notification with Redis Pub/Sub

API List

redisnotifier/createnotifier ()
Create a new notifier object

manager

Loading Path	`vlcp.service.manage.modulemanager.Manager`
Class Reference	*vlcp.service.manage.modulemanager.Manager*
Dependencies	(None)

Manage module loading/unloading. Optionally reload a module when modified.

API List

manager/enableautoreload (enabled=True)
Enable or disable auto reload.

Parameters **enabled** – enable if True, disable if False

manager/activemodules ()
Return current loaded modules

manager/reloadmodules (pathlist)
Reload specified modules.

Parameters **pathlist** – list of module path

manager/loadmodule (path)
Load specified module

Parameters **path** – module path (e.g. `vlcp.service.connection.httpserver.HttpServer`)

manager/unloadmodule (path)
Unload specified module

Parameters **path** – module path (e.g. `vlcp.service.connection.httpserver.HttpServer`)

webapi

Loading Path	`vlcp.service.manage.webapi.WebAPI`
Class Reference	*`vlcp.service.manage.webapi.WebAPI`*
Dependencies	*httpserver*

Call module API from web. Free access to any module APIs may create serious security problems, make sure to configure this module properly.

arpresponder

Loading Path	`vlcp.service.sdn.arpresponder.ARPResponder`
Class Reference	*`vlcp.service.sdn.arpresponder.ARPResponder`*
Dependencies	*openflowportmanager* *objectdb*

Send ARP respond

API List

`arpresponder/createproxyarp (connection, arpentries)`
Create ARP respond flow for specified ARP entries, each is a tuple `(ip_address, mac_address, logical_network_id, local)`. When local is True, only respond to ARP request from logical port; when local is False, only respond to ARP request from physical port; respond to both else.

`arpresponder/removeproxyarp (connection, arpentries)`
Remove specified ARP entries.

dhcpserver

Loading Path	`vlcp.service.sdn.dhcpserver.DHCPServer`
Class Reference	*`vlcp.service.sdn.dhcpserver.DHCPServer`*
Dependencies	*openflowportmanager* *objectdb* *arpresponder*

DHCP server that responds the DHCP discover/request with static IP address settings

API List

dhcpserver/gettablerequest ()
Table requirement for this module

icmpresponder

Loading Path	`vlcp.service.sdn.icmpresponder.ICMPResponder`
Class Reference	*vlcp.service.sdn.icmpresponder.ICMPResponder*
Dependencies	*openflowportmanager* *objectdb*

Respond ICMP echo (ping) requests to the gateway

API List

icmpresponder/gettablerequest ()
Table requirement for this module

ioprocessing

Loading Path	`vlcp.service.sdn.ioprocessing.IOProcessing`
Class Reference	*vlcp.service.sdn.ioprocessing.IOProcessing*
Dependencies	*openflowportmanager* *ovsdbportmanager* *objectdb*

Ingress and Egress processing

API List

ioprocessing/gettablerequest ()
Table requirement for this module

l2switch

Loading Path	`vlcp.service.sdn.l2switch.L2Switch`
Class Reference	*`vlcp.service.sdn.l2switch.L2Switch`*
Dependencies	*openflowportmanager* *objectdb*

L2 switch functions

API List

`l2switch/gettablerequest ()`
Table requirement for this module

l3router

Loading Path	`vlcp.service.sdn.l3router.L3Router`
Class Reference	*`vlcp.service.sdn.l3router.L3Router`*
Dependencies	*arpresponder* *icmpresponder* *objectdb*

L3 connectivities with virtual router.

API List

`l3router/gettablerequest ()`
Table requirement for this module

openflowmanager

Loading Path	`vlcp.service.sdn.ofpmanager.OpenflowManager`
Class Reference	*`vlcp.service.sdn.ofpmanager.OpenflowManager`*
Dependencies	*openflowserver*

Manage Openflow Connections

API List

openflowmanager/getconnections (datapathid, vhost='')
Return all connections of datapath

openflowmanager/getconnection (datapathid, auxiliaryid=0, vhost='')
Get current connection of datapath

openflowmanager/waitconnection (datapathid, auxiliaryid=0, timeout=30, vhost='')
Wait for a datapath connection

openflowmanager/getdatapathids (vhost='')
Get All datapath IDs

openflowmanager/getalldatapathids ()
Get all datapath IDs from any vhost. Return `(vhost, datapathid)` pair.

openflowmanager/getallconnections (vhost='')
Get all connections from vhost. If vhost is None, return all connections from any host

openflowmanager/getconnectionsbyendpoint (endpoint, vhost='')
Get connection by endpoint address (IP, IPv6 or UNIX socket address)

openflowmanager/getconnectionsbyendpointname (name, vhost='', timeout=30)
Get connection by endpoint name (Domain name, IP or IPv6 address)

openflowmanager/getendpoints (vhost='')
Get all endpoints for vhost

openflowmanager/getallendpoints ()
Get all endpoints from any vhost. Return `(vhost, endpoint)` pairs.

openflowmanager/acquiretable (modulename)
Start to acquire tables for a module on module loading.

openflowmanager/unacquiretable (modulename)
When module is unloaded, stop acquiring tables for this module.

openflowmanager/lastacquiredtables (vhost='')
Get acquired table IDs

openflowportmanager

Loading Path	`vlcp.service.sdn.ofpportmanager.OpenflowPortManager`
Class Reference	*`vlcp.service.sdn.ofpportmanager.OpenflowPortManager`*
Dependencies	*openflowmanager*

Manage Ports from Openflow Protocol

API List

openflowportmanager/getports (datapathid, vhost='')
Return all ports of a specifed datapath

openflowportmanager/getallports (vhost=None)
Return all `(datapathid, port, vhost)` tuples, optionally filterd by vhost

openflowportmanager/getportbyno (datapathid, portno, vhost='')
Return port with specified OpenFlow portno

openflowportmanager/waitportbyno (datapathid, portno, timeout=30, vhost='')
Wait for the specified OpenFlow portno to appear, or until timeout.

openflowportmanager/getportbyname (datapathid, name, vhost='')
Return port with specified port name

openflowportmanager/waitportbyname (datapathid, name, timeout=30, vhost='')
Wait for a port with the specified port name to appear, or until timeout

openflowportmanager/resync (datapathid, vhost='')
Resync with current ports

ovsdbmanager

Loading Path	`vlcp.service.sdn.ovsdbmanager.OVSDBManager`
Class Reference	*`vlcp.service.sdn.ovsdbmanager.OVSDBManager`*
Dependencies	*jsonrpcserver*

Manage Openflow Connections

API List

ovsdbmanager/getconnection (datapathid, vhost='')
Get current connection of datapath

ovsdbmanager/waitconnection (datapathid, timeout=30, vhost='')
Wait for a datapath connection

ovsdbmanager/getdatapathids (vhost='')
Get All datapath IDs

ovsdbmanager/getalldatapathids ()
Get all datapath IDs from any vhost. Return `(vhost, datapathid)` pair.

ovsdbmanager/getallconnections (vhost='')
Get all connections from vhost. If vhost is None, return all connections from any host

ovsdbmanager/getbridges (connection)
Get all `(dpid, name, _uuid)` tuple on this connection

ovsdbmanager/getbridge (connection, name)
Get datapath ID on this connection with specified name

ovsdbmanager/getbridgebyuuid (connection, uuid)
Get datapath ID of bridge on this connection with specified _uuid

ovsdbmanager/waitbridge (connection, name, timeout=30)
Wait for bridge with specified name appears and return the datapath-id

ovsdbmanager/waitbridgebyuuid (connection, uuid, timeout=30)
Wait for bridge with specified _uuid appears and return the datapath-id

ovsdbmanager/getsystemids (vhost='')
Get All system-ids

ovsdbmanager/getallsystemids ()
Get all system-ids from any vhost. Return `(vhost, system-id)` pair.

ovsdbmanager/getconnectionbysystemid (systemid, vhost='')

ovsdbmanager/waitconnectionbysystemid (systemid, timeout=30, vhost='')
Wait for a connection with specified system-id

ovsdbmanager/getconnectionsbyendpoint (endpoint, vhost='')
Get connection by endpoint address (IP, IPv6 or UNIX socket address)

ovsdbmanager/getconnectionsbyendpointname (name, vhost='', timeout=30)
Get connection by endpoint name (Domain name, IP or IPv6 address)

ovsdbmanager/getendpoints (vhost='')
Get all endpoints for vhost

ovsdbmanager/getallendpoints ()
Get all endpoints from any vhost. Return `(vhost, endpoint)` pairs.

ovsdbmanager/getallbridges (vhost=None)
Get all `(dpid, name, _uuid)` tuple for all connections, optionally filtered by vhost

ovsdbmanager/getbridgeinfo (datapathid, vhost='')
Get `(bridgename, systemid, bridge_uuid)` tuple from bridge datapathid

ovsdbmanager/waitbridgeinfo (datapathid, timeout=30, vhost='')
Wait for bridge with datapathid, and return `(bridgename, systemid, bridge_uuid)` tuple

ovsdbportmanager

Loading Path	`vlcp.service.sdn.ovsdbportmanager.OVSDBPortManager`
Class Reference	*`vlcp.service.sdn.ovsdbportmanager.OVSDBPortManager`*
Dependencies	*ovsdbmanager*

Manage Ports from OVSDB Protocol

API List

ovsdbportmanager/getports (datapathid, vhost='')
Return all ports of a specifed datapath

ovsdbportmanager/getallports (vhost=None)
Return all `(datapathid, port, vhost)` tuples, optionally filterd by vhost

ovsdbportmanager/getportbyid (id, vhost='')
Return port with the specified id. The return value is a pair: `(datapath_id, port)`

ovsdbportmanager/waitportbyid (id, timeout=30, vhost='')
Wait for port with the specified id. The return value is a pair `(datapath_id, port)`

ovsdbportmanager/getportbyname (datapathid, name, vhost='')
Return port with specified name

ovsdbportmanager/waitportbyname (datapathid, name, timeout=30, vhost='')
Wait for port with specified name

ovsdbportmanager/getportbyno (datapathid, portno, vhost='')
Return port with specified portno

ovsdbportmanager/waitportbyno (datapathid, portno, timeout=30, vhost='')
Wait for port with specified portno

ovsdbportmanager/resync (datapathid, vhost='')
Resync with current ports

networklocaldriver

Loading Path	`vlcp.service.sdn.plugins.networklocaldriver.NetworkLocalDriver`
Class Reference	*`vlcp.service.sdn.plugins.networklocaldriver.NetworkLocalDriver`*
Dependencies	(None)

Network driver for local networks. Local networks cannot have physical ports; logical networks in local networks do not have external connectivities, only endpoints on the same server can access each other.

networknativedriver

Loading Path	`vlcp.service.sdn.plugins.networknativedriver.NetworkNativeDriver`
Class Reference	*`vlcp.service.sdn.plugins.networknativedriver.NetworkNativeDriver`*
Dependencies	(None)

Network driver for native networks. Native network is a physical network provides only one logical network capacity. Packets from the logical network is directly forwarded to the physical network.

networkvlandriver

Loading Path	`vlcp.service.sdn.plugins.networkvlandriver.NetworkVlanDriver`
Class Reference	*`vlcp.service.sdn.plugins.networkvlandriver.NetworkVlanDriver`*
Dependencies	(None)

Network driver for VXLAN networks. When creating a VXLAN type physical network, you must specify an extra option `vlanrange`.

networkvxlandriver

Loading Path	`vlcp.service.sdn.plugins.networkvxlandriver.NetworkVxlanDriver`
Class Reference	*`vlcp.service.sdn.plugins.networkvxlandriver.NetworkVxlanDriver`*
Dependencies	(None)

Network driver for VXLAN networks. When creating a VXLAN type physical network, you must specify an extra option `vnirange`.

viperflow

Loading Path	`vlcp.service.sdn.viperflow.ViperFlow`
Class Reference	*`vlcp.service.sdn.viperflow.ViperFlow`*
Dependencies	*objectdb*

Standard network model for L2 SDN

API List

`viperflow/createphysicalnetwork (type='vlan', id=None, **kwargs)`
Create physical network.

Parameters

- **`type`** – Network type, usually one of *vlan*, *vxlan*, *local*, *native*
- **`id`** – Specify the created physical network ID. If omitted or None, an UUID is generated.
- **`**kwargs`** – extended creation parameters. Look for the document of the corresponding driver. Common options include:

  vnirange list of `[start,end]` ranges like `[[1000,2000]]`. Both *start* and *end* are included. It specifies the usable VNI ranges for VXLAN network.

  vlanrange list of `[start,end]` ranges like `[[1000,2000]]`. Both *start* and *end* are included. It specifies the usable VLAN tag ranges for VLAN network.

Returns A dictionary of information of the created physical network.

`viperflow/createphysicalnetworks (networks)`
Create multiple physical networks in a transaction.

Parameters **`networks`** – each should be a dictionary contains all the parameters in `createphysicalnetwork`

Returns A list of dictionaries of information of the created physical networks.

viperflow/updatephysicalnetwork (id, **kwargs)
Update physical network with the specified ID.

Parameters

- **id** – physical network ID
- ****kwargs** – attributes to be updated, usually the same attributes for creating.

Returns A dictionary of information of the updated physical network.

viperflow/updatephysicalnetworks (networks)
Update multiple physical networks in a transaction

Parameters **networks** – a list of dictionaries, each contains parameters of `updatephysicalnetwork`

Returns A list of dictionaries of information of the updated physical network.

viperflow/deletephysicalnetwork (id)
Delete physical network with specified ID

Parameters **id** – Physical network ID

Returns `{"status": "OK"}`

viperflow/deletephysicalnetworks (networks)
Delete multiple physical networks with a transaction

Parameters **networks** – a list of `{"id": <id>}` dictionaries.

Returns `{"status": "OK"}`

viperflow/listphysicalnetworks (id=None, **kwargs)
Query physical network information

Parameters

- **id** – If specified, only return the physical network with the specified ID.
- ****kwargs** – customized filters, only return a physical network if the attribute value of this physical network matches the specified value.

Returns A list of dictionaries each stands for a matched physical network

**viperflow/createphysicalport (physicalnetwork, name, vhost='', systemid='%', bridge='%', **
Create physical port

Parameters

- **physicalnetwork** – physical network this port is in.
- **name** – port name of the physical port, should match the name in OVSDB
- **vhost** – only match ports for the specified vHost
- **systemid** – only match ports on this systemid; or '%' to match all systemids.
- **bridge** – only match ports on bridges with this name; or '%' to match all bridges.
- ****kwargs** – customized creation options, check the driver document

Returns A dictionary containing information of the created physical port.

viperflow/createphysicalports (ports)
Create multiple physical ports in a transaction

Parameters **ports** – A list of dictionaries, each contains all parameters for `createphysicalport`

Returns A list of dictionaries of information of the created physical ports

viperflow/updatephysicalport (name, vhost='', systemid='%', bridge='%', **kwargs)
Update physical port

Parameters

- **name** – Update physical port with this name.
- **vhost** – Update physical port with this vHost.
- **systemid** – Update physical port with this systemid.
- **bridge** – Update physical port with this bridge name.
- ****kwargs** – Attributes to be updated

Returns Updated result as a dictionary.

viperflow/updatephysicalports (ports)
Update multiple physical ports with a transaction

Parameters **ports** – a list of `updatephysicalport` parameters

Returns Updated result as a list of dictionaries.

viperflow/deletephysicalport (name, vhost='', systemid='%', bridge='%')
Delete a physical port

Parameters

- **name** – physical port name.
- **vhost** – physical port vHost.
- **systemid** – physical port systemid.
- **bridge** – physcial port bridge.

Returns `{"status": "OK"}`

viperflow/deletephysicalports (ports)
Delete multiple physical ports in a transaction

Delete a physical port

Parameters **ports** – a list of `deletephysicalport` parameters

Returns `{"status": "OK"}`

viperflow/listphysicalports (name=None, physicalnetwork=None, vhost='', systemid='%', bridg
Query physical port information

Parameters

- **name** – If specified, only return the physical port with the specified name.
- **physicalnetwork** – If specified, only return physical ports in that physical network
- **vhost** – If specified, only return physical ports for that vHost.
- **systemid** – If specified, only return physical ports for that systemid.
- **bridge** – If specified, only return physical ports for that bridge.

- ****kwargs** – customized filters, only return a physical network if the attribute value of this physical network matches the specified value.

Returns A list of dictionaries each stands for a matched physical network

viperflow/createlogicalnetwork (physicalnetwork, id=None, **kwargs)
Create logical network

Parameters

- **physicalnetwork** – physical network ID that contains this logical network
- **id** – logical network ID. If ommited an UUID is generated.
- ****kwargs** – customized options for logical network creation. Common options include:

  vni/vxlan Specify VNI / VLAN tag for VXLAN / VLAN network. If omitted, an unused VNI / VLAN tag is picked automatically.

  mtu MTU value for this network. You can use 1450 for VXLAN networks.

Returns A dictionary of information of the created logical port

viperflow/createlogicalnetworks (networks)
Create multiple logical networks in a transaction.

Parameters **networks** – a list of `createlogicalnetwork` parameters.

Returns a list of dictionaries for the created logical networks.

viperflow/updatelogicalnetwork (id, **kwargs)
Update logical network attributes of the ID

viperflow/updatelogicalnetworks (networks)
Update multiple logical networks in a transaction

viperflow/deletelogicalnetwork (id)
Delete logical network

viperflow/deletelogicalnetworks (networks)
Delete logical networks

Parameters **networks** – a list of `{"id":id}`

Returns `{"status": "OK"}`

viperflow/listlogicalnetworks (id=None, physicalnetwork=None, **kwargs)
Query logical network information

Parameters

- **id** – If specified, only return the logical network with the specified ID.
- **physicalnetwork** – If specified, only return logical networks in this physical network.
- ****kwargs** – customized filters, only return a logical network if the attribute value of this logical network matches the specified value.

Returns A list of dictionaries each stands for a matched logical network

viperflow/createlogicalport (logicalnetwork, id=None, subnet=None, **kwargs)
Create logical port

Parameters

- **logicalnetwork** – logical network containing this port
- **id** – logical port id. If omitted an UUID is created.

- **subnet** – subnet containing this port
- ****kwargs** – customized options for creating logical ports. Common options are:

  mac_address port MAC address

  ip_address port IP address

Returns a dictionary for the logical port

viperflow/createlogicalports (ports)
Create multiple logical ports in a transaction

viperflow/updatelogicalport (id, **kwargs)
Update attributes of the specified logical port

viperflow/updatelogicalports (ports)
Update multiple logcial ports

viperflow/deletelogicalport (id)
Delete logical port

viperflow/deletelogicalports (ports)
Delete multiple logical ports

viperflow/listlogicalports (id=None, logicalnetwork=None, **kwargs)
Query logical port

Parameters

- **id** – If specified, returns only logical port with this ID.
- **logicalnetwork** – If specified, returns only logical ports in this network.
- ****kwargs** – customzied filters

Returns return matched logical ports

viperflow/createsubnet (logicalnetwork, cidr, id=None, **kwargs)
Create a subnet for the logical network.

Parameters

- **logicalnetwork** – The logical network is subnet is in.
- **cidr** – CIDR of this subnet like `"10.0.1.0/24"`
- **id** – subnet ID. If omitted, an UUID is generated.
- ****kwargs** – customized creating options. Common options are:

  gateway Gateway address for this subnet

  allocated_start First IP of the allowed IP range.

  allocated_end Last IP of the allowed IP range.

  host_routes A list of `[dest_cidr, via]` like `[["192.168.1.0/24", "192.168.2.3"],["192.168.3.0/24","192.168.2.4"]]`. This creates static routes on the subnet.

  isexternal This subnet can forward packet to external physical network

  pre_host_config A list of `[{systemid, bridge, cidr, local_address, gateway, ...}]` Per host configuration, will union with public info when used

Returns A dictionary of information of the subnet.

viperflow/createsubnets (subnets)
Create multiple subnets in a transaction.

viperflow/updatesubnet (id, **kwargs)
Update subnet attributes

viperflow/updatesubnets (subnets)
Update multiple subnets

viperflow/deletesubnet (id)
Delete subnet

viperflow/deletesubnets (subnets)
Delete multiple subnets

viperflow/listsubnets (id=None, logicalnetwork=None, **kwargs)
Query subnets

Parameters

- **id** – if specified, only return subnet with this ID
- **logicalnetwork** – if specified, only return subnet in the network
- ****kwargs** – customized filters

Returns A list of dictionaries each stands for a matched subnet.

vrouterapi

Loading Path	`vlcp.service.sdn.vrouterapi.VRouterApi`
Class Reference	*vlcp.service.sdn.vrouterapi.VRouterApi*
Dependencies	*objectdb*

Standard network model for L3 SDN

API List

vrouterapi/createvirtualrouter (id=None, **kwargs)
Create a virtual router

Parameters

- **id** – Virtual router id. If omitted, an UUID is generated.
- ****kwargs** – extra attributes for creation.

Returns A dictionary of information of the virtual router.

vrouterapi/createvirtualrouters (routers)
Create multiple virtual routers in a transaction

vrouterapi/updatevirtualrouter (id, **kwargs)
Update virtual router

vrouterapi/updatevirtualrouters (routers)
Update multiple virtual routers

vrouterapi/deletevirtualrouter (id)
Delete virtual router

vrouterapi/deletevirtualrouters (routers)
Delete multiple virtual routers

vrouterapi/listvirtualrouters (id=None, **kwargs)
Query virtual router

Parameters

- **id** – if specified, only return virtual router with this ID
- ****kwargs** – customized filter

Returns a list of dictionaries each stands for a matched virtual router.

vrouterapi/addrouterinterface (router, subnet, id=None, **kwargs)
Connect virtual router to a subnet

Parameters

- **router** – virtual router ID
- **subnet** – subnet ID
- **id** – router port ID
- ****kwargs** – customized options

Returns A dictionary of information of the created router port

vrouterapi/addrouterinterfaces (interfaces)
Create multiple router interfaces

vrouterapi/removerouterinterface (router, subnet)
Remote a subnet from the router

Parameters

- **router** – virtual router ID
- **subnet** – subnet ID

Returns `{"status": "OK"}`

vrouterapi/removerouterinterfaces (interfaces)
Remote multiple subnets from routers

vrouterapi/listrouterinterfaces (id, **kwargs)
Query router ports from a virtual router

Parameters

- **id** – virtual router ID
- ****kwargs** – customized filters on router interfaces

Returns a list of dictionaries each stands for a matched router interface

vtepcontroller

Loading Path	`vlcp.service.sdn.vtepcontroller.VtepController`
Class Reference	*`vlcp.service.sdn.vtepcontroller.VtepController`*
Dependencies	*jsonrpcserver* *objectdb*

Controll a physical switch which supports OVSDB hardware_vtep protocol.

API List

vtepcontroller/listphysicalports (physicalswitch=None)
Get physical ports list from this controller, grouped by physical switch name

Parameters **physicalswitch** – physicalswitch name. Return all switches if is None.

Returns dictionary: {physicalswitch: [physicalports]} e.g. {'ps1': ['port1', 'port2']}

vtepcontroller/listphysicalswitches (physicalswitch=None)
Get physical switch info

Parameters **physicalswitch** – physicalswitch name. Return all switches if is None.

Returns dictionary: {physicalswitch: {key: value}} keys include: management_ips, tunnel_ips, description, switch_fault_status

vtepcontroller/updatelogicalswitch (physicalswitch, physicalport, vlanid, logicalnetwork, v
Bind VLAN on physicalport to specified logical network, and update logical port vxlan info

Parameters

- **physicalswitch** – physical switch name, should be the name in PhysicalSwitch table of OVSDB vtep database
- **physicalport** – physical port name, should be the name in OVSDB vtep database
- **vlanid** – the vlan tag used for this logicalswitch
- **logicalnetwork** – the logical network id, will also be the logical switch id
- **vni** – the VXLAN VNI of the logical network
- **logicalports** – a list of logical port IDs. The VXLAN info of these ports will be updated.

vtepcontroller/unbindlogicalswitch (physicalswitch, physicalport, vlanid, logicalnetwork)
Remove bind of a physical port

Parameters

- **physicalswitch** – physical switch name, should be the name in PhysicalSwitch table of OVSDB vtep database
- **physicalport** – physical port name, should be the name in OVSDB vtep database
- **vlanid** – the vlan tag used for this logicalswitch
- **logicalnetwork** – the logical network id, will also be the logical switch id

vtepcontroller/unbindphysicalport (physicalswitch, physicalport)
Remove all bindings for a physical port

Parameters

- **physicalswitch** – physical switch name, should be the name in PhysicalSwitch table of OVSDB vtep database
- **physicalport** – physical port name, should be the name in OVSDB vtep database

vxlancast

Loading Path	`vlcp.service.sdn.vxlancast.VXLANCast`
Class Reference	*vlcp.service.sdn.vxlancast.VXLANCast*
Dependencies	*openflowportmanager* *objectdb* *ovsdbportmanager*

VXLAN single-cast and broadcast functions

vxlanvtep

Loading Path	`vlcp.service.sdn.vxlanvtep.VXLANVtep`
Class Reference	*vlcp.service.sdn.vxlanvtep.VXLANVtep*
Dependencies	(None)

Use hardware_vtep instead of software VXLAN

API List

vxlanvtep/get_vxlan_bind_info (systemid=None)
get vxlan -> vlan , bind info

autoload

Loading Path	`vlcp.service.utils.autoload.AutoLoad`
Class Reference	*vlcp.service.utils.autoload.AutoLoad*
Dependencies	(None)

Auto load some modules from a package. Usually used to load network plugins.

knowledge

Loading Path	`vlcp.service.utils.knowledge.Knowledge`
Class Reference	*`vlcp.service.utils.knowledge.Knowledge`*
Dependencies	(None)

Simple KV-cache in memory. A base for other KV-DB. Data is automatically removed after timeout. Use knowledge instead of local storage in modules so data is not lost on module restarting.

API List

`knowledge/get (key, timeout=None)`
Get value from key

`knowledge/set (key, value, timeout=None)`
Set value to key, with an optional timeout

`knowledge/delete (key)`
Delete a key

`knowledge/mget (keys)`
Get multiple values from multiple keys

`knowledge/mgetwithcache (keys, cache=None)`
Get multiple values, cached when possible

`knowledge/mset (kvpairs, timeout=None)`
Set multiple values on multiple keys

`knowledge/update (key, updater, timeout=None)`
Update in-place with a custom function

Parameters

- **`key`** – key to update
- **`updater`** – `func(k,v)`, should return a new value to update, or return None to delete
- **`timeout`** – new timeout

Returns the updated value, or None if deleted

`knowledge/mupdate (keys, updater, timeout=None)`
Update multiple keys in-place one by one with a custom function, see update. Either all success, or all fail.

`knowledge/updateall (keys, updater, timeout=None)`
Update multiple keys in-place, with a function `updater(keys, values)` which returns `(updated_keys, updated_values)`. Either all success or all fail

`knowledge/updateallwithtime (keys, updater, timeout=None)`
Update multiple keys in-place, with a function `updater(keys, values, timestamp)` which returns `(updated_keys, updated_values)`. Either all success or all fail.

Timestamp is a integer standing for current time in microseconds.

remotecall

Loading Path	`vlcp.service.utils.remoteapi.RemoteCall`
Class Reference	*vlcp.service.utils.remoteapi.RemoteCall*
Dependencies	(None)

Route local API calls to remote management API.

API List

remotecall/call (remote_module, method, timeout, params)
Call remote API

Parameters

- **remote_module** – target name for the remote module
- **method** – method name of the API
- **timeout** – timeout for the call
- **params** – A dictionary contains all the parameters need for the call

Returns Return result from the remote call

session

Loading Path	`vlcp.service.utils.session.Session`
Class Reference	*vlcp.service.utils.session.Session*
Dependencies	*memorystorage*

HTTP Session with cookies

API List

session/start (cookies, cookieopts=None)
Session start operation. First check among the cookies to find existed sessions; if there is not an existed session, create a new one.

Parameters

- **cookies** – cookie header from the client
- **cookieopts** – extra options used when creating a new cookie

Returns `(session_handle, cookies)` where session_handle is a SessionHandle object, and cookies is a list of created Set-Cookie headers (may be empty)

session/create ()
Create a new session object

Returns Session handle for the created session object.

session/get (sessionid, refresh=True)
Get the seesion object of the session id

Parameters

- **sessionid** – a session ID
- **refresh** – if True, refresh the expiration time of this session

Returns Session object or None if not exists

session/destroy (sessionid)
Destroy a session

Parameters **sessionid** – session ID

Returns a list of Set-Cookie headers to be sent to the client

static

Loading Path	`vlcp.service.web.static.Static`
Class Reference	*vlcp.service.web.static.Static*
Dependencies	*httpserver*

Map specified path to local files

API List

static/updateconfig ()
Reload configurations, remove non-exist servers, add new servers, and leave others unchanged

dockerplugin

Loading Path	`vlcp_docker.dockerplugin.DockerPlugin`
Class Reference	*vlcp_docker.dockerplugin.DockerPlugin*
Dependencies	*httpserver* *viperflow* *objectdb*

Integrate VLCP with Docker

API List

dockerplugin/getdockerinfo (portid)
Get docker info for specified port

3.2.5 Proxy Module List

Proxy modules are configurable modules that proxy abstract API requests to different implementations. See *proxy* for more information.

updatenotifier

Default	*redisnotifier*

kvstorage

Default	*redisdb*

memorystorage

Default	*knowledge*

CHAPTER 4

Development Guide

Achitecture design and technical information of VLCP. You may want to read this if you are interested in the working theories or would like to be a contributor.

4.1 Contributing

Any contribution is welcome - Report bugs or new ideas on GitHub, fix this document, edit GitHub Wiki, fix this document, contributing with pull requests or recommend this software to your friends or co-workers.

Project GitHub address is https://github.com/hubo1016/vlcp.

Author E-mail: hubo1016@126.com

4.2 Architecture Overview

This is the overall design of VLCP

4.2.1 Technology Stack

VLCP consists of multiple layers, each solves different problems and use different techniques. Following figure *Technology Stack of VLCP* shows the technology stack of VLCP:

4.2.2 Asynchronous Programming Framework

VLCP uses a special designed asynchronous programming framework to deal with network I/O and other problems. Similar to *asyncio* in Python 3.4+, This framework uses Python generators as coroutines, but there are some differents between *asyncio* and VLCP.

For further reading: *Asynchronous Core Design*.

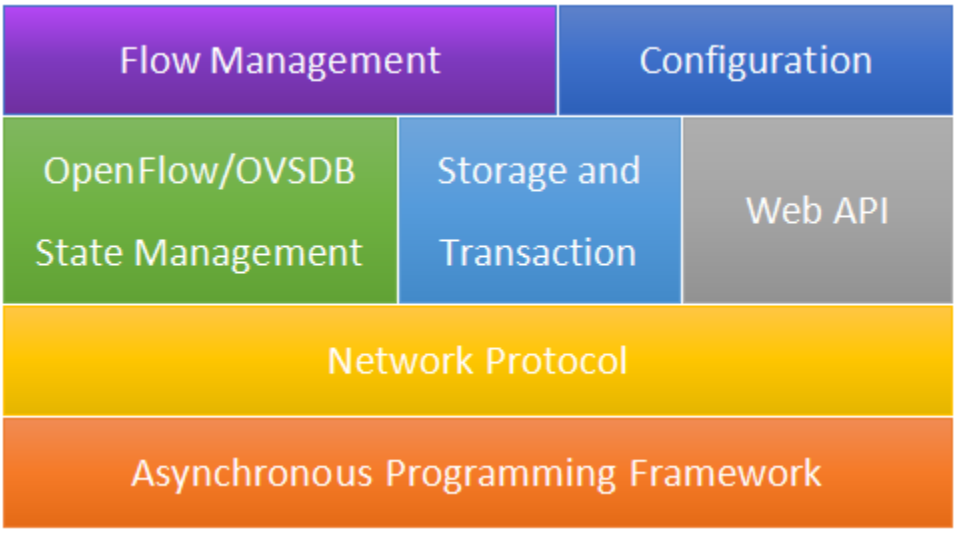

Fig. 1: Technology Stack of VLCP

4.2.3 Network Protocol Parsing

VLCP framework uses an extensible way to deal with network protocols, which separates lower-level socket processing with specified protocol processing. For each network protocol, a protocol class derived from *vlcp.protocol.protocol.Protocol* is created to parse the byte stream to messages.

VLCP currently implements: OpenFlow, JSON-RPC(OVSDB), HTTP, RESP(Redis) and ZooKeeper protocols. OpenFlow (1.0 and 1.3) and ZooKeeper protocols are processed with namedstruct, which is a flexible library for parsing and constructing binary structures. Structs are defined like:

```
ofp_flow_mod = nstruct(
  (uint64, 'cookie'),            #  /* Opaque controller-issued identifier. */
# /* Mask used to restrict the cookie bits
# that must match when the command is
# OFPFC_MODIFY* or OFPFC_DELETE*. A value
# of 0 indicates no restriction. */
  (uint64, 'cookie_mask'),
# /* ID of the table to put the flow in.
# For OFPFC_DELETE_* commands, OFPTT_ALL
# can also be used to delete matching
# flows from all tables. */
  (ofp_table, 'table_id'),
  (ofp_flow_mod_command.astype(uint8), 'command'),           #  /* One of OFPFC_*. */
  (uint16, 'idle_timeout'),     #  /* Idle time before discarding (seconds). */
  (uint16, 'hard_timeout'),     #  /* Max time before discarding (seconds). */
  (uint16, 'priority'),         #  /* Priority level of flow entry. */
# /* Buffered packet to apply to, or
# OFP_NO_BUFFER.
# Not meaningful for OFPFC_DELETE*. */
  (ofp_buffer_id, 'buffer_id'),
# /* For OFPFC_DELETE* commands, require
# matching entries to include this as an
# output port.  A value of OFPP_ANY
# indicates no restriction. */
  (ofp_port_no, 'out_port'),
```

(continues on next page)

(continued from previous page)

```
# /* For OFPFC_DELETE* commands, require
# matching entries to include this as an
# output group.  A value of OFPG_ANY
# indicates no restriction. */
  (ofp_group, 'out_group'),
  (ofp_flow_mod_flags, 'flags'),            #  /* Bitmap of OFPFF_* flags. */
  (uint8[2],),
  (ofp_match, 'match'),         #  /* Fields to match. Variable size. */
#    /* The variable size and padded match is always followed by instructions. */
# /* Instruction set - 0 or more.
# The length of the instruction
# set is inferred from the
# length field in the header. */
  (ofp_instruction[0], 'instructions'),
  base = ofp_msg,
  name = 'ofp_flow_mod',
  criteria = lambda x: x.header.type == OFPT_FLOW_MOD,
  classifyby = (OFPT_FLOW_MOD,),
  init = packvalue(OFPT_FLOW_MOD, 'header', 'type')
)
```

and used like:

```
import vlcp.protocol.openflow.defs.openflow13 as ofdef

flow_mod_command = ofdef.ofp_flow_mod(
       table_id = arp,
       cookie = 0x1 | (0x2 if islocal else 0),
       cookie_mask = 0xffffffffffffffff,
       command = ofdef.OFPFC_ADD,
       buffer_id = ofdef.OFP_NO_BUFFER,
       out_port = ofdef.OFPP_ANY,
       out_group = ofdef.OFPG_ANY,
       priority = ofdef.OFP_DEFAULT_PRIORITY + 10,
       match = ofdef.ofp_match_oxm(
                     oxm_fields = [
                           ofdef.create_oxm(ofdef.OXM_OF_IN_PORT, pid),
                           match_network(nid),
                           ofdef.create_oxm(ofdef.OXM_OF_ETH_TYPE, ofdef.ETHERTYPE_
↪ARP),
                           ofdef.create_oxm(ofdef.OXM_OF_ARP_TPA, ofdef.ip4_addr(ip)),
                           ofdef.create_oxm(ofdef.OXM_OF_ARP_OP, ofdef.ARPOP_REQUEST)]
                         + ([ofdef.create_oxm(ofdef.OXM_OF_ETH_DST_W, b
↪'\x01\x00\x00\x00\x00\x00', b'\x01\x00\x00\x00\x00\x00')]
                           if broadcast else [])
                ),
       instructions = [ofdef.ofp_instruction_actions(type = ofdef.OFPIT_CLEAR_
↪ACTIONS)]
       )
```

Read namedstruct document for more information.

4.2.4 OpenFlow/OVSDB State Management

VLCP keeps a list of ports from OVSDB and OpenFlow. When there are changes in OpenvSwitch, notifications are received from OVSDB and OpenFlow connections. VLCP deals with these notifications and send messages to

acknowledge higher-level modules to update flows.

4.2.5 Storage and Transaction

VLCP uses KV-databases like ZooKeeper or Redis for central data storage. Every VLCP node can commit data changes to the central database, and when commiting, a notification is sent to any nodes that "watch" any of the updated keys. When nodes receive the update notification, they reload the data from the central database and update flows according to the new data.

Commits can be done to multiple keys at once which forms a *transaction*. The transaction layer guarantees:

1. For one transaction, either all keys are updated successfully, or all keys are not updated (Atomic)
2. For one transaction, all keys are updated once in the same time; there is not a time point when some keys are updated and other keys are not (Consistency)
3. a transaction is always performed in an isolated view; updates from other transactions do not interfere this transaction (Isolation)
4. updates from a transaction is durable once the transaction succeeded (Durability)

This provides the same consistency with RDBMS. Every VLCP node can have a consistent view of the central database (though they may not be updated at exactly the same time)

For further reading: *Transaction Layer: ObjectDB*

4.2.6 Web API

The Web API module exports module APIs to a HTTP service. See *Call Module API from WebAPI*

4.2.7 Flow Management

VLCP controls OpenFlow switches (like OpenvSwitch) with OpenFlow protocol. It creates flows to program the packet fowarding rules to provide the SDN functions. The flows are created from port state and central configurations. Once a logical port is created, and a port on OpenvSwitch has an *iface-id* same with the logical port ID, VLCP creates flows for this logical port and the logical network containing this port. Once a port on OpenvSwitch has a name that matches a physical port configuration, VLCP creates flows for this physical port and the physical network containing this port. When the port state is updated, or the related database keys are updated, VLCP updates the flows according to the update of ports or database keys.

For further reading: *SDN Design and Implementations*

4.2.8 Network Configuration

Configurations are created, updated or deleted with module APIs of *viperflow* and *vrouterapi*.

4.2.9 Modules Design

Current modules and their relationships are shown in the figure *Modules of VLCP*:

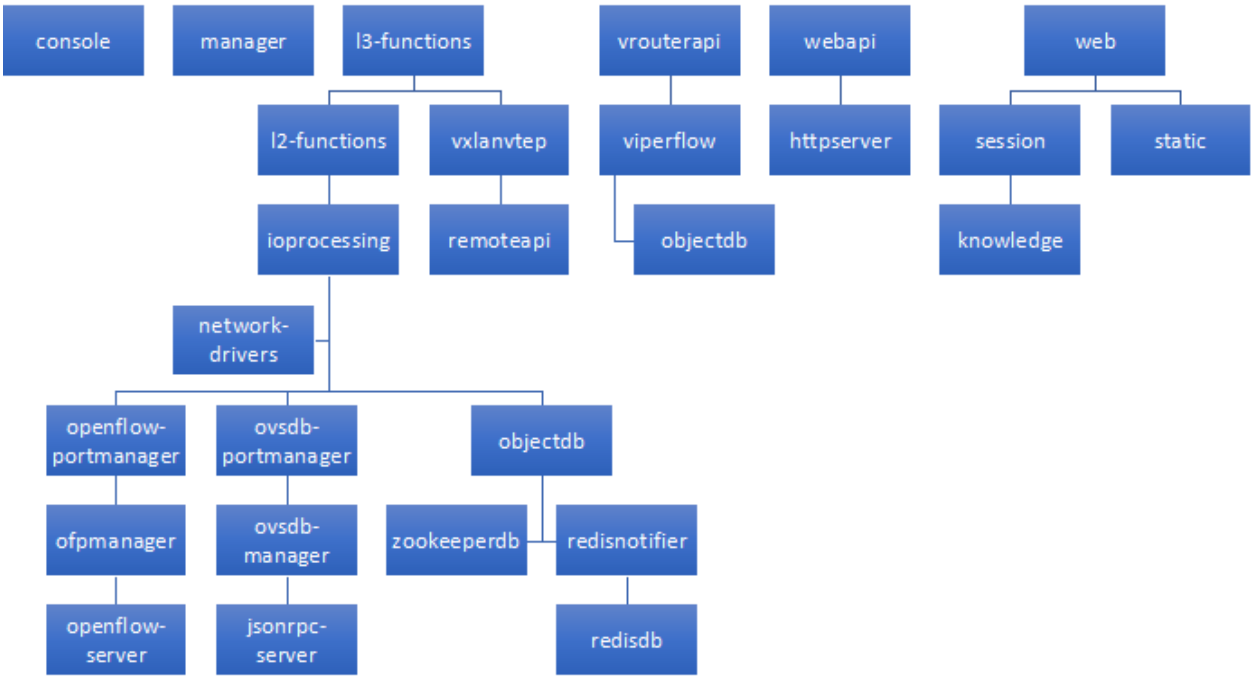

Fig. 2: Modules of VLCP

4.3 Asynchronous Core Design

This is the design of VLCP asynchronous core.

4.3.1 Event Objects

The most principal synchronizing method in VLCP is **Event Object/Event Matcher**.

Event Objects are instances of event classes. An event class is a subclass of *vlcp.event.event.Event*. When defining an event class, the decorator `withIndices()` is used to create 0, 1 or more indices for this event class. When an event object is created, one value for each index must be assigned from the arguments. For example:

```
@withIndices('id', 'network')
class PortCreatedEvent(Event):
    pass
```

defines an event class *PortCreatedEvent*, with two indices: the port id *id* and the network id *network*. And:

```
new_event = PortCreatedEvent('new_port', 'my_network')
id = new_event.id
network = new_event.network
```

Creates a new event whose port id is *'new_port'* and network id is *'my_network'*. Notice that value for each index does not have to be strings, but must be hashable values like integers, objects or tuples, and cannot be *None*. The values for the indices are automatically set to corresponding attributes of the event object. Other attributes can be set with extra keyword arguments.

An **Event Matcher** matches some types of event objects. It can be created from *createMatcher* method of an event class:

```
# Matches PortCreatedEvent with id = 'new_port' and network = 'my_network'
my_matcher1 = PortCreatedEvent.createMatcher('new_port', 'my_network')
# Matches PortCreatedEvent with id = 'new_port'
my_matcher2 = PortCreatedEvent.createMatcher('new_port')
# Matches PortCreatedEvent with network = 'my_network'
my_matcher3 = PortCreatedEvent.createMatcher(None, 'my_network')
# Matches any PortCreatedEvent
my_matcher4 = PortCreatedEvent.createMatcher()
# Use a customized function to test events
my_matcher5 = PortCreatedEvent.createMatcher('new_port',
                  _ismatch = lambda x: x.network.startswith('my_'))
```

An event matcher matches an event object when:

1. The event object is an instance of the corresponding event class (or one of its sub class)
2. All the specified index values match the values of the event object
3. If *_ismatch* is specified, the customized function must return True for this event object.

Note: Whenever possible, event matchers should use indices to match the events. Matching an event object with event matchers having different index values is O(1), while matching an event object with event matchers having different *_ismatch* is O(N) - every event matcher is tested once.

An event class can be subclassed. The subclassed event class inherits all the indices from the parent class, and can have its own indices. Rules for matches for subclassed event classes are:

1. An event matcher of the parent class can match a subclassed event object
2. An event matcher of a subclassed event class CANNOT match an event object of the parent class or other subclasses

When an event matcher or an event object of a subclassed event is created, the arguments should consist of all indices from the ancestors to the descendants.

Note: event objects and event matchers also accept keyword-arguments on initializing, but mixing placement arguments and keyword arguments is NOT supported.

4.3.2 VLCP Routines and Routine Containers

VLCP is a coroutine-based framework. Each coroutine is a Python async coroutine (created by an async function):

```
from vlcp.event import M_

async def new_routine():
    # Wait for event1
    ev = await event_matcher1
    # Some work...
    # Wait for event2
    ev = await event_matcher2
    # Wait for multiple events, return the first matched event and the corresponding␣
↪matcher
```

(continues on next page)

(continued from previous page)

```
    ev, m = await M_(event_matcher3, event_matcher4)
    if m is event_matcher3:
        ...
    else:
        ...
```

Note: From v2.0, async coroutines are used instead of generators. Notice that the async functions are not compatible with *asyncio*: *asyncio* awaitable cannot be awaited in VLCP, and VLCP awaitables cannot be awaited in *asyncio*.

Following objects are awaitables in VLCP:

1. An event matcher - when awaited, return the matched event
2. A *vlcp.event.M_* object - wait for multiple matchers, return (event, matcher) tuple
3. A *vlcp.event.future.Future* object - return the result of the future
4. (internal) *vlcp.event.event.Diff_* and *vlcp.event.event.DiffRef_* object - specialized event matcher tuples for efficient differencing, used by *wait_for_all*.
5. Other coroutines (created by an async method)

The routine is suspended by scheduler to wait for an event object which matches one of the yielded event matchers inside a *await* expression. When this event object appears, scheduler wake up the routine to let it continue. An event object can wake up multiple routines, and the routines will be executed in order.

Each routine is associated with a **Routine Container**. The routine container is an object of type *vlcp.event.runnable.RoutineContainer*. It is used as the executing context of the routine. When the routine awakes, the matched event object and the matcher is sent to the routine as *await* return values. Routines use these variables to determine what to do next.

Note: In v1.x, *container.event* and *container.matcher* is used to receive the event and matcher. It is no longer supported in v2.0 - use return value instead.

An async method can also be awaited:

```
async def my_method():
    ev = await my_event_matcher
    return ev.result

return_value = await my_method()
```

Note: You must use *await* to call a coroutine method. Use only *my_method()* does not have any effect. Beginners are easy to make this mistake. Python 3.5+ will show warnings on coroutines not awaited.

Coroutine methods can return value like *return_value* in the above example.

Note: In v1.x, *container.retvalue* is used for return value of a coroutine method (because Python 2.x does not support returning a value from a generator method). This is no longer supported in v2.0

Routine containers have some helpful methods to construct common work flows. One of the most important methods is *wait_for_send*, which sends an event object to wake up other routines:

```
await container.wait_for_send(my_event)
```

Note: From v2.0, many methods of *RoutineContainer* class uses name consistent with PEP 8(lower_case_with_underscores), but the previous mixedCase names (like *waitForSend*) is kept for compatibility.

The sending process is asynchronous, which means the method returns before other routines receive this event object.

Another method *subroutine* creates a new routine and let it executes independently:

```
async def new_routine():
    ...

container.subroutine(new_routine())
```

It is quite similar to the *go* statement in Golang.

Note: In v1.x, many async methods can only be called by routines in the same *RoutineContainer*. This limit is removed in v2.0, so *delegate* methods are deprecated.

See *vlcp.event.runnable.RoutineContainer* for all the useful methods.

4.3.3 Scheduler

A VLCP scheduler consists of an event queue, a match tree and a polling provider, like in figure *Scheduler Work Flow*.

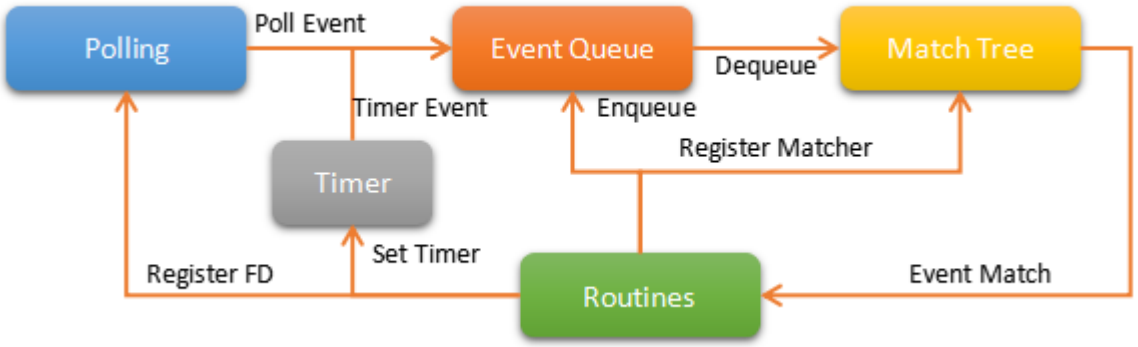

Fig. 3: Scheduler Work Flow

Match Tree is a *trie* (or *prefix tree*) which matches event objects with event matchers. It returns event matchers which matches an event object in O(1) time. Routines register the event matchers they are waiting for into the match tree with *yield* expression, and wait for a matching event.

In the main loop, scheduler takes event objects out of the **Event Queue**. Then scheduler uses the event object to find matched event matchers and their corresponding routines. Scheduler executes the routines until they reach the next *yield* expression. In the while, the routines may send new event objects into the event queue, register file descriptors to the polling provider, or set timers.

When there is no event objects in the event queue, or the event processing limit is reached, scheduler uses the polling provider to wait for socket activities until next timer is triggered. The generated polling events and timer events are sent to the event queue.

4.3.4 Event Queue

Event queue in VLCP stores and reorders event objects. The main queue consists of multiple subqueues, each has an event matcher. Event objects are classified by these event matchers into subqueues. Each subqueue has a different priority, so that events in higher priority subqueues are retrieved first. Events in different subqueues with a same priority are retrieved in round-robin order. This helps on reducing latency for critical messages or balancing CPU usage to different connections on high load. A subqueue can also have subqueues to provide more control on event priorities.

A subqueue can have size limit, so that when the subqueue is full, the routine which tries to send an event with *wait_for_send()* stops and wait for the queue to have space for more events. This provides an easy way to create a robust consumer-producer system.

Subqueues can be created or removed by routines when the scheduler is running. Event senders and receivers do not have to care about subqueues, they always send to / receive from the main event queue, thus are not affected by subqueue changes. Events stored in a subqueue can be cleared if necessary.

Routines can also wait for a subqueue to be empty.

4.3.5 Blocking Events

Usually when an event object is taken from the event queue, it will be ignored if there are no event matchers matching this event. Some important events cannot be ignored and must be processed correctly. If there are no matching event matchers currently, the event is delayed until a matching event matcher is registered. This kind of events is called blocking events.

A blocking event is simply an event object with *canignore* = *False*. Usually it is set on the event class to make the event object blocking by default, like:

```
@withIndices('id')
class MyBlockingEvent(Event):
    canignore = False
```

When an event object is processed, the routine should set *canignore* = *True* on the event object immediately:

```
async def my_routine():
    matcher = MyBlockingEvent.createMatcher(12)
    ev = await matcher
    ev.canignore = True
```

When a blocking event is not processed correctly, it goes back to the subqueue from the front end, and blocks the subqueue until it is matched by a newly registered event matcher. The processing order of the events are not changed. If the subqueue contains this event has a size limit, producers of these events are blocked until consumers begin to process these events.

Sometimes we need to discard blocking events that are no longer needed. Besides clearing the subqueue which contains these events, the event class can provide an *canignorenow()* function to make the blocking conditional. When scheduler processes an event object with *canignore=False* and also *canignorenow()*, it executes *canignorenow()* which returns a boolean value. If *canignorenow()* returns True, scheduler set *canignore=True* on the event object and ignore it. This only happens when an event object is take out from the event queue, so events which are already blocking the subqueues cannot be ignored. A routine should use scheduler method *vlcp.event.core.Scheduler.ignore()* together with *canignorenow()* to correctly ignore these events.

4.3.6 Connection Processing

VLCP processes all sockets (including TCP connections and UNIX connections) with routines.

The `vlcp.event.connection.Connection` class is responsible for all the lower-level socket operations. It creates a reading routine, a writing routine and a controlling routine for each connection.

Reading routine uses a protocol class (subclass of `vlcp.protocol.protocol.Protocol`) to parse the byte stream into event objects. When sending the event objects, queue size limit may cause the routine to stop to wait for event processing, thus stop receiving on the socket. For streaming sockets (TCP, UNIX), traffic control on this connection makes the remote side stop sending more data.

Writing routine waits for `vlcp.event.connection.ConnectionWriteEvent` event objects for this connection. Data retrieved from the event objects are sent to the socket. *ConnectionWriteEvent* objects are blocking events, so when the writing routine cannot write more data to the socket, it will start to wait for the polling event for socket write, so routines generating the *ConnectionWriteEvent* are blocked until more data can be written to the socket. Producers do not need to worry about generating too many data to send.

Controlling routine waits for connection control events which shutdown or restart the connection.

4.3.7 Connector

A connector is a bridge between VLCP schedulers and other threads or processes. It sends events to other threads from a thread-safe queue, and receives events from a pipe. With connectors it is quite easy to create a thread pool to execute methods in a multi-threaded way.

A specialized connector *TaskPool* (`vlcp.utils.connector.TaskPool`) is a simple thread pool implementation for executing tasks in other threads.

4.4 Transaction Layer: ObjectDB

ObjectDB is the transaction layer in VLCP. Usually, VLCP controllers are deployed on each server. All the information needed by the controller is stored in an external KV-database like ZooKeeper or Redis, as the figure *figure_centralstorage* shows:

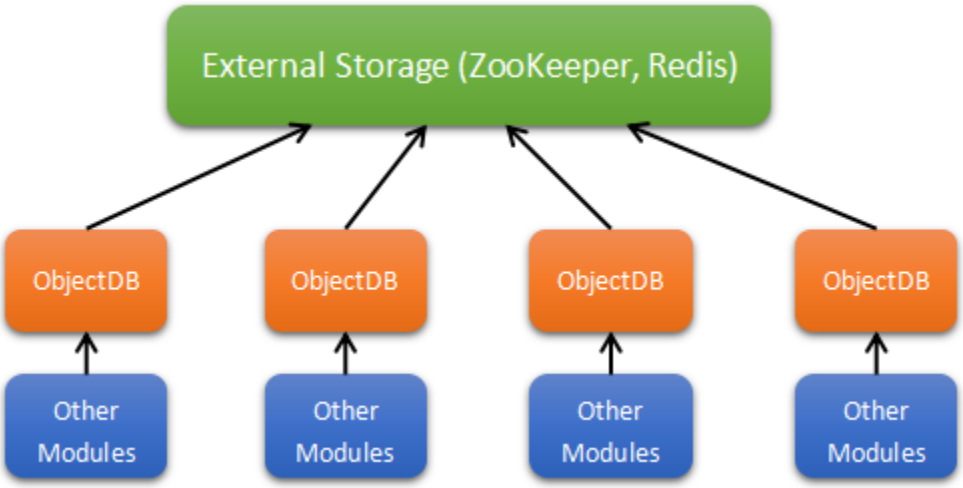

Fig. 4: Central Storage

Every node query the central storage to get the information they care about. They may also write back to the central storage to store state information or get synchronized with other nodes. Every node is equal to each other, they can read or write all the keys at the same time. The transaction layer *ObjectDB* is implemented to synchronize these read or write operations to provide atomic update for multiple keys

4.4.1 Data Structures

ObjectDB stores data with *DataObjects*. Each *DataObject* is a Python object which can be serialized with JSON (or pickle, if configured). Every *DataObject* must have an unique key, which is determind by its primary attributes (e.g. id), like *viperflow.logicalport.docker-7c857946c3a4ba7f4e1066d7c942d7ed3b3c245a443a8f43ed19baa23c56dd73*.

The *DataObject* is serialized to JSON and stored to the KV-database with the specified key. When a node wants to query the data of an object, it provides the key of the object and get the JSON-deserialized object.

When an object need to reference other *DataObjects*, it stores its key to one of its attributes, or stores the keys to a list. When the object is retrieved, the program can further retrieve the referenced objects.

But there are some problems:

1. When we update a *DataObject*, we read the object from the database, update the value and write it back. This may overwrite other updates between the read and the write. (Not **Isolated**)
2. When we update multiple *DataObjects* at the same time, some of them may success while others may fail. (Not **Atomic**)
3. When two nodes both update two *DataObjects* A and B, there is a chance that the final result of A and B are from different nodes (Not **Consistent**)
4. When we retrieve a node, and try to retrieve further references, the references may already be changed by other nodes (Not **Consistent**)
5. When we updated multiple keys, a remote node may only update part of them (Not **Consistent**)

That is why we need a transaction layer to solve these problems.

4.4.2 Basic Design

The basic design of ObjectDB is shown in figure *ObjectDB Basic Design*:

ObjectDB depends on two components: the *KVStorage* module and the *Notifier* module. A *KVStorage* module provides two basic interfaces:

`KVStorage.`**`updateallwithtime`**(*keys*, *updater*)

Basic write transaction on the storage. The update process must be atomic.

Parameters

- **`keys`** – a tuple of keys of DataObjects
- **`updater`** – a python function

  `updater`(*keys*, *values*, *timestamp*)

  A function describing a transaction. The function may be called more than once and it must return the same result with the same parameters. It cannot be a routine method.

  Parameters

  - **`keys`** – a tuple of keys of DataObjects. It is the same as the keys in *updateallwithtime*.
  - **`values`** – a tuple of DataObject values. If the object do not exist in the storage, the corresponding value is None.

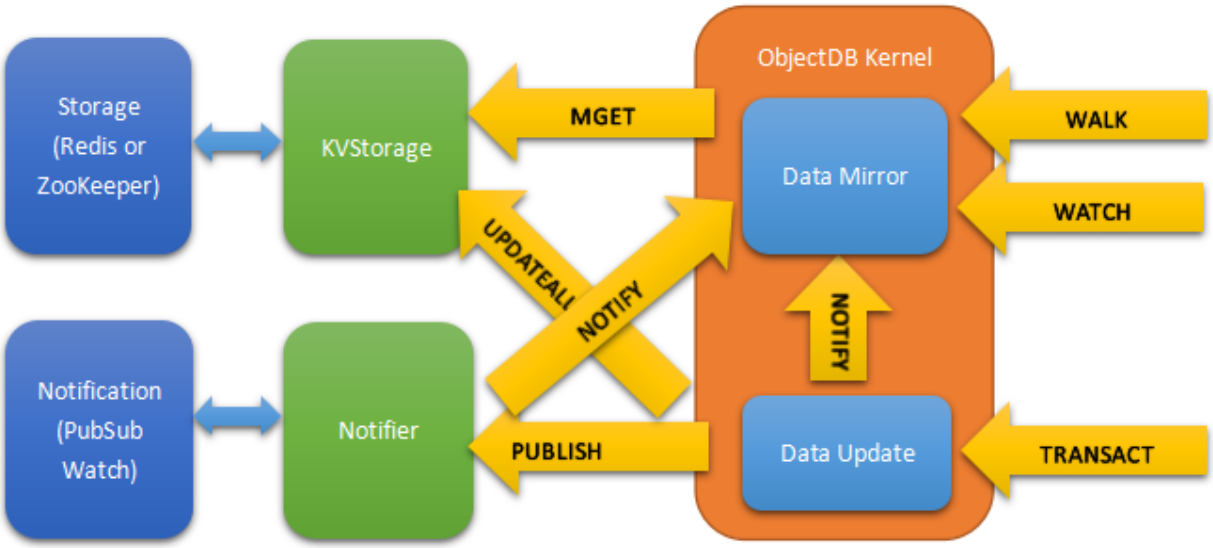

Fig. 5: ObjectDB Basic Design

– **timestamp** – a server side timestamp from the central database with micro seconds

Returns *(updated_keys, updated_values)* to write to the central database.

If the function raises any exception, the transaction is aborted.

Returns the final return value of *updater*

KVStorage.**mgetwithcache**(*keys*, *cache* = *None*)

Basic read transaction on the storage. The read process must be atomic.

Parameters **keys** – a tuple of keys of DataObjects

Returns *(result, cache)* tuple. *result* is the DataObjects corresponding to the keys. If a DataObject is not found in the central database, *None* is returned for this key. *cache* is a cache object used by later calls, to cache necessary information for acceleration (for example, if the data is not changed, storage module can return the exact same instance stored in the cache object by previous calls).

These two methods provide the basic abilities for transaction. It is implemented with the lower-level KV-database functions, for example, Redis uses the "WATCH/MULTI/EXEC" procedure, and ZooKeeper uses the *MultiRequest* command.

ObjectDB creates *DataObject* caches for all the watching keys. The cache is called the *data mirror*. All routines that query data from ObjectDB get references of the *DataObjects*. The *ReferenceObject* is a proxy object which reads the attributes from the *DataObject* but prevent writing to them. This makes sure the *DataObject* is correctly shared between different routines.

4.4.3 Transact and Notification

A read-only transaction uses *mgetwithcache* to get values in the same DB version. A read-only transaction can only use data from the same DB version. If some necessary is missing or out-dated, a *mgetwithcache* is used to retrieve all the needed keys from KVStorage.

A read-write transact is done with *updateallwithtime*, so it is natually a transact operate. After the transaction, a notification is sent to this node and other nodes.

Notifications contain the full list of keys that are updated. When nodes receive this notification, it always retrieve these updated keys with a *mgetwithcache*, so the view on each node is always consistent.

Notification for this node is from a shortcut to let the data mirror been updated immediately.

4.4.4 Walk Method

Walker method are high-level transaction methods, they provide generic transaction ability, and are easy to use.

ObjectDB provides a *walk* method for read-only transaction. It retrieve related *DataObjects* at once. It uses *walker* functions to retrieve data:

walker(*key*, *object*, *walk*, *save*)

A function describing a reading transaction. The function may be called more than once when executed in *ObjectDB.walk*. It should use *walk* and *save* interactively to retrieve the results. If the function raises an exception, the transaction is aborted.

Parameters

- **key** – The key of the initial starting object.
- **object** – The value of the initial starting object.
- **walk** – A function to retrieve a DataObject:

  walk(*key*)

  Parameters **key** – key of a DataObject to retrieve
  Returns the DataObject value, or None if not existed.
  Raises **vlcp.utils.exception.WalkKeyNotRetrieved** – raised if the key has not been retrieved from the central database yet. The walker function should catch this exception and stop further retrieving depends on the return value. ObjectDB will call *walker* again after the keys are retrieved.

  WalkKeyNotRetrieved exception is a subclass of *KeyError*

- **save** – A function to save a retrieved key:

  save(*key*)

  Parameters **key** – key of a DataObject to save. It must be either the original *key* when *walker* is called, or has been retrieved with *walk*

Saved keys from the walker is returned from ObjectDB, and is registered to ObjectDB as been *watching*. A key been watching receives update notifications when it is updated by other operatings either from this node or from other nodes. Use *unwatch* to cancel monitoring of the key.

When the walk method is called, ObjectDB first tries to execute the walkers in the current data mirror. If there are keys that are not retrieved, ObjectDB tries to retrieve **all keys that are used by the walker** with *mgetwithcache*.

Each *mgetwithcache* call creates a different version of data mirror. Data mirror before current execution is version -1. For each retrieved key, the valid version range is calculated. For example, if key *A* is in data mirror before execution (version -1), and is retrieved at version 4, but the value is not changed, then the valid version range is [-1, 4], closed. If key *B* is retrieved in version 1, and version 4, but the value in version 4 has been changed, then the valid version range is [4, 4].

When a walker is executing, only keys that has at least one compatible data mirror version can be retrieved. That means the walker is always executed in a consistent dataset. This is described with figure *Isolation of Data Space for walkers*:

When the keys needed do not have a compatible data mirror version, all the keys will be retrieved with *mgetwithcache* in the next version, so they will have a compatible version on next execution. If some values are changed, the keys

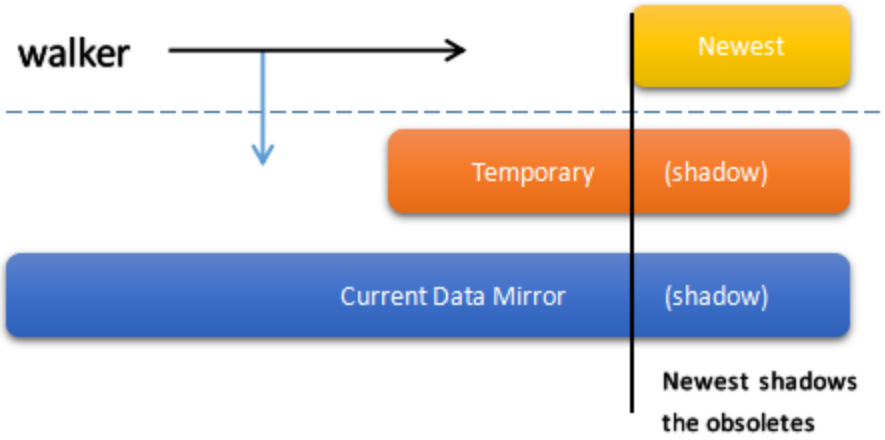

Fig. 6: Isolation of Data Space for walkers

retrieved by the walker may differ from the previous execution. This will continue until the walker can successfully finish executing in a complete and compatible dataset.

If update notifications are received during the updating procedure, the keys are updated with the same *mgetwithcache*. If a *mgetwithcache* retrieves *DataObjects* that are newer than the latest update notification, ObjectDB waits for the update notification to update all the other keys at the same time. When the values are updated, all the related walkers are restarted to use the latest value.

4.4.5 Write Walk Methods

Write walk methods are high-level read-write transaction methods. Similar to walk, a *walker* function is needed. The parameters are slightly different, and only one *walker* function is needed:

walker(*walk*, *write*)

A function describing a read-write transaction. The function may be called more than once when executed in *ObjectDB.writewalk*. It should use *walk* and *write* interactively to modify values. If the function raises an exception, the transaction is aborted.

Parameters

- **walk** – A function to retrieve a DataObject:

  walk(*key*)

  Parameters **key** – key of a DataObject to retrieve
  Returns the DataObject value, or None if not existed.
  Raises **vlcp.utils.exception.WalkKeyNotRetrieved** – raised if the key has not been retrieved from the central database yet. The walker function should catch this exception and stop further retrieving depends on the return value. ObjectDB will call *walker* again after the keys are retrieved.

  WalkKeyNotRetrieved exception is a subclass of *KeyError*

- **write** – A function to write a value to a key:

  write(*key*, *value*)

  Parameters
  - **key** – key of a DataObject to write.

- **value** – a DataObject for updating or *None* for deleting.

Modifed values must be written to database with *write* methods even if it is modified in-place. *write* can be used on the same key for multiple times, and the last value is written when transaction ends. *walk* always retrieved the last written value of a key if it has been written for at least once.

Sometimes the execution of the transaction depends on the current (server) time. with *timestamp=True*, an extra parameter *timestamp* can be used in *walker* function:

walker(*walk*, *write*, *timestamp*)

Parameters **timestamp** – A server-side timestamp in micro seconds

When a transaction needs async support (e.g. some related information are retrieved from network), an *asyncwritewalk* method can be used instead. The *asyncwritewalk* method uses an *asyncwalker* method as a walker factory:

(async) asyncwalker(last_info, container)

A function describing an async read-write transaction. Each time *asyncwalker* is executed, it returns a *(keys, walker)* tuple for a *writewalk*. the returned *walker* may raises *vlcp.utils.exception.AsyncTransactionLockException* to interrupt the transaction and give some extra info for the next execution, so *asyncwalker* can recreate the walker.

Parameters

- **last_info** – When *asyncwalker* is called for the first time, last_info is *None*. After that, it is the first argument of the last *AsyncTransactionLockException* raised by *walker*
- **container** – The routine container that executes the current routine

Returns *(keys, walker)* where *keys* are the estimated keys which are needed by the transaction (for performance optimizing only). *walker* has the same signature used in *writewalk*, but can raise *AsyncTransactionLockException* to interrupt current transaction and retry from *asyncwalker*. The first argument of *AsyncTransactionLockException* will be passed as *last_info* when calling *asyncwalker* next time.

writewalk and *asyncwritewalk* has following guarantees:

1. All value retrieved by *walk* in *walker* are at the same DB version (*Consistent*)
2. Written values can only be commited to database if all the retrieved values are not modified by other transactions (*Isolated*)
3. Either all written values are written, or none of them are written if transaction rolls back (*Atomic*)

writewalk internally uses *asynctransact*, which calls lower-level *transact* repeatedly with current estimated keys. The internal updater calls *walker* with a local cache. If the keys retrieving by *walk* is not in the current estimated keys, it is added to the list on next try. If the *walker* completes without missing keys, the written values are returned to let the transaction finish in the KVStorage.

4.5 SDN Design and Implementations

VLCP SDN framework allows the modules to operate Flows on a high level. With the support of lower-level modules, flow generations are easy to understand and easy to implement.

4.5.1 State Management of OpenFlow

VLCP controller always flush all the flows in the switch when it is connected to the controller. This makes sure the flows in the switches are consist with the view of the controller.

While the connection is alive, controller tries to add/remove/update minimal flows when necessary. This makes the network stable on state changes.

Modules use notifications from the transaction layer to update flows. The steps are:

1. Query data from the central database
2. Update flows base on the latest data
3. Wait for update notifications from *objectdb*
4. If it is necessary to restart the query, goto 1; else goto 2

With the ACID guarantees from *objectdb* module, it is easy to update flows in a safe way.

4.5.2 Plugable Tables

In OpenFlow, tables are identified by table ID, which is a number. Processing on a packet is done from the lower IDs to the higher IDs. This makes it difficult to extend an existed model: we need to insert or remove tables between existing tables.

VLCP uses an extensible way to allocate table IDs for each module. It uses an unique name to identify a table. Each table should declare none or more tables which must have smaller IDs than this table, they are called *ancestors* of this table. This makes sure a flow can use *GOTO* instruction from these tables to the defined table.

A table is always in one *path*. A *path* is a chain of tables which are processed one by one. VLCP inserts a default flow for each table in a *path* to GOTO the next table in the same path, so you can insert extra tables in a path without disturbing the original processing order. Each *path* also has an unique name, and the name of the default *path* is an empty string *“”*. Flows in tables can use *GOTO* instruction to jump to another *path* for extra processing. Modules may also replace the default flow in a table to drop unmatched packets or upload the packet to controller with *OPF_PACKET_IN* message.

On module loading, each module starts to acquire tables from *openflowmanager* with *acquiretable* API. *openflowmanager* module queries each module with a *gettablerequest* API. The API should return a tuple:

```
(table_requests, vhost_bind)
```

vhost_bind is a list of vhosts this module is binding to. It defaults to *[“”]*, which binds only the default vHost.

table_requests is the following structure:

```
((name1, (ancestor1_1, ancestor1_2, ...), pathname1),
 (name2, (ancestor2_1, ancestor2_2, ...), pathname2),
 ...)
```

Each line acquires a table. The first element *name* is the unique name of this table; if multiple modules acquire the same name, it is considered to be the same table. *ancestors* are tuples of table names, they may not be defined in this *table_requests* structure. *pathname* is the path name of this table.

For example, the module *ioprocessing* defines two tables:

```
(("ingress", (), ''),
 ("egress", ("ingress",),''))
```

An *ingress* table and an *egress* table, all in the default path. The *egress* table must have larger ID than the *ingress* table. If *ioprocessing* is the only SDN module loaded, there will be only two tables used in the switch.

In module *l2switch*, more tables are defined:

```
(("l2input", ('ingress',), ''),
 ("l2output", ('l2input',), ''),
 ('egress', ('l2output', 'l2learning'), ''),
 ("l2learning", ('l2output',), 'l2learning'))
```

This creates *l2input*, *l2output* and *l2learning* tables. They must be in *ingress* -> *l2input* -> *l2output* -> *l2learning* -> *egress* order. The *l2learning* table is not in the default path, so a packet does not go through *l2output* to *l2learning* by default.

4.5.3 Strategies

Some modules can have different strategies. Usually there are three types of strategies:

Prepush The controller pre-creates all flows which endpoints may need to use. This has the best stabilities and performance for reasonable sized logical networks. When load on the central database is high, there may be a delay of a few seconds before the flows are created.

Learning The controller uses information from the incoming packets to create flows for outgoing packets. For example, input port of a packet with specified MAC address is memorised and saved to a flow. When an outgoing packet with the specified MAC address as the destination MAC is forwarded, the flow directs the packet to the original input port. If the needed flow is not created by the incoming packets, switch uses broadcast instead. This is the triditional way for switches to process packets. Extra broadcasting packets may be sent in this mode. There are two types of learning techniques:

nx_learn This is an OpenFlow extension of OpenvSwitch. This action allows the learning procedure executed directly on OpenvSwitch, thus has better performance. This is recommended for very large scale of logical networks.

controller learning This is a replacement for *nx_learn*. If you are not using OpenvSwitch (e.g. using physical switches), this uses OFP_PACKET_IN to upload the packet to controller for the learning procedure, which may increase the load of controller.

First-Upload The switch sends a packet which does not match any exising flows to controller via OFP_PACKET_IN message. The controller looks up the information for this packet and generate a flow for it. Further packets with the same properties are processed by the created flow. This introduces a quite large delay for the first packet, but eliminates the broadcasting packets. Usually this is not recommended.

These strategies can be configured from the module configurations, see *Configuration Manual* for details.

4.5.4 Flow Table Design

Current SDN modules (with L3 support) and tables they used can be expressed with the figure *OpenFlow Tables*:

Description for each table:

ingress This table do inital processes on the packets, initializing registers

l2input This table drops packets which should not be forwarded (e.g. STP packets, packets with broadcast source MACs). If learning is enabled, this table uses *nx_learn* action or OFP_PACKET_IN to creating learning flows which matches the source MAC with the input port.

vxlaninput If learning is enabled, this table uses *nx_learn* action to create learning flows which matches the source MAC with the tunnel source IP address.

arp ARP responders. Endpoints send broadcasting ARP packets to look up the MAC address for a specified IP address. This table directly responds these broadcasting ARP packets with the correct MAC address to eliminate these ARP packets.

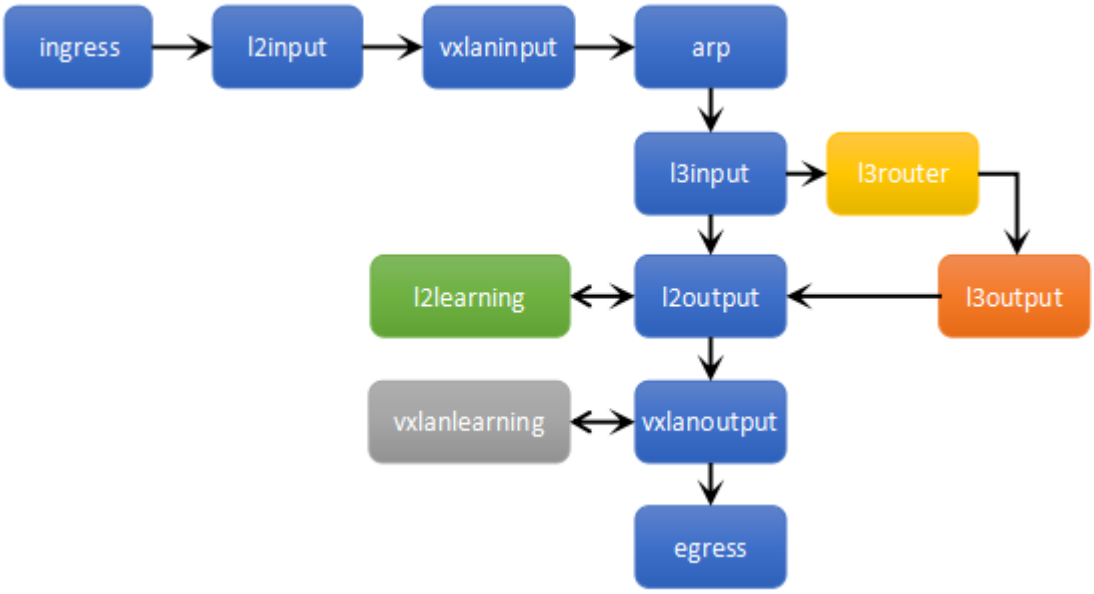

Fig. 7: OpenFlow Tables

l3input Embedded DHCP service uses this table to upload DHCP requests to the controller. Virtual routers uses this table to redirect packets sent to the router gateway to *l3router* table.

l3router Routing tables for each virtual router. When there is a next-hop IP address, source MAC of the packet is changed to the router MAC, destination MAC of the packet is changed to the next-hop MAC address; when the next-hop is on a connected network, goto *l3output*

l3output Lookup destination MAC address for L3 outgoing packets.

l2output Lookup the output port for this packet

l2learning If nx_learn is used, this table contains the learned flows, and is used by *l2output*

vxlanoutput Lookup the tunnel destination IP address for packets in an overlay network (VXLAN)

vxlanlearning If nx_learn is used, this table contains the learned flows, and is used by *vxlanoutput*

egress Output of packets

4.6 VLCP Configuration Desigin and Implementations

As *Configuration Manual* description, VLCP use config file to change the behaviors of all the internal objects. each key value in file will be store into `ConfigTree` an data structure named `manager`. every key made up of multi parts split by . , describe an inherit relationship, so config key has a close relationship with source class code. class instance attribution will map with `ConfigTree` , so we can change value through config file. The steps are:

1. read config file to manager ConfigTree.
2. map class instance attribution with ConfigTree.

4.6.1 Read config file to ConfigTree

When VLCP start, it will read config , parse it , and store it to `manager`. as code below:

```
server.py:: main: manager.loadfrom(configpath)
config.py:: manager: loadfromfile(file)
```

every key value will be store as `manager` attribution.

Note: ConfigTree class has implements __setitems__ , so you can read config.py to know how to store attribution.

4.6.2 Map class instace attribution

After perpare `ConfigTree` , every class want to map attribution to manager `ConfigTree`, must be inherit from `Configurable` an class has implements `__getattr__` method. as steps:

```
1. try return manager[getattr(cls, 'configkey') + '.' + key] as value
2. try return cls.__dict__['_default_' + key] as value
3. try parent class goto 1
```

so when class get attribution will get manager value first.

but as step 1 we need class `configkey` attribution to find value. so class must have `configkey` attribution also. there is three help method in `config.py` to create configkey attribution:

```
configbase(key)
config(key)
defaultconfig(key)
```

for example an configurable class mybe link this:

```
@config('server')
class Server(Configurable):
    ......
```

Note: configkey has relationship with base class , you can read function declaration in source code.

CHAPTER 5

Reference

5.1 vlcp.config

Note: This document is generated from the source file.

View Source on GitHub

5.1.1 vlcp.config.config

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/6/25

author hubo

class vlcp.config.config.**ConfigTree**

A basic config node. A node supports both attributes get/set and dict-like operations. When using dict-like interfaces, configurations in child nodes can directly be used:

```
node['child.childconfig'] = 42
node.child.childconfig  # 42
```

__contains__(*key*)

Support dict-like *in* operator

__delitem__(*key*)

Support dict-like deletion

__getitem__(*key*)
Support dict-like retrieval

__init__()
Initialize self. See help(type(self)) for accurate signature.

__iter__()
Support a dict-like iterate

__len__()
Return size of children stored in this node, either sub nodes or configuration values

__setitem__(*key*, *value*)
Support dict-like assignment

clear()
Support dict-like clear

config_items(*sortkey=False*)
Return all *(key, value)* tuples for configurations in this node, including configurations on children nodes.

config_keys(*sortkey=False*)
Return all configuration keys in this node, including configurations on children nodes.

config_value_items(*sortkey=False*)
Return *(key, value)* tuples for configuration directly stored in this node. Configurations in child nodes are not included.

config_value_keys(*sortkey=False*)
Return configuration keys directly stored in this node. Configurations in child nodes are not included.

get(*key*, *defaultvalue=None*)
Support dict-like get (return a default value if not found)

gettree(*key*, *create=False*)
Get a subtree node from the key (path relative to this node)

items()
Return *(key, value)* tuples for children in this node, either sub nodes or configuration values

keys()
Return all children in this node, either sub nodes or configuration values

loadconfig(*keysuffix*, *obj*)
Copy all configurations from this node into obj

setdefault(*key*, *defaultvalue=None*)
Support dict-like setdefault (create if not existed)

todict()
Convert this node to a dictionary tree.

withconfig(*keysuffix*)
Load configurations with this decorator

class vlcp.config.config.**Configurable**
Base class for a configurable object. Undefined attributes of a configurable object is mapped to global configurations. The attribute value of a configurable object is:

1. The original attribute value if it is set on the instance or class
2. The configuration value *manager[self.configkey + '.' + attrname]* if exists
3. The configuration value *manager[parent.configkey + '.' + attrname]* if parent classes have *configkey* defined and configuration exists.
4. The *_default_<attrname>* attribute value of the instance

5. raises AttributeError

Attributes begins with '_' is not mapped.

configkey and *configbase* attribute should be set on this class. Usually they are set by decorators *@defaultconfig*, *@configbase* or *@config*

__init__()
Initialize self. See help(type(self)) for accurate signature.

config_value_items(*sortkey=False*)
Return *(key, value)* tuples for all mapped configurations for this object

config_value_keys(*sortkey=False*)
Return all mapped configuration keys for this object

classmethod getConfigRoot(*create=False*)
Return the mapped configuration root node

classmethod getConfigurableParent()
Return the parent from which this class inherits configurations

class vlcp.config.config.**Manager**
Configuration manager. Use the global variable *manager* to access the configuration system.

__init__()
Initialize self. See help(type(self)) for accurate signature.

loadfrom(*path*)
Read configurations from path

loadfromfile(*filelike*)
Read configurations from a file-like object, or a sequence of strings. Old values are not cleared, if you want to reload the configurations completely, you should call *clear()* before using *load** methods.

loadfromstr(*string*)
Read configurations from string

save(*sortkey=True*)
Save configurations to a list of strings

saveto(*path*, *sortkey=True*)
Save configurations to path

savetofile(*filelike*, *sortkey=True*)
Save configurations to a file-like object which supports *writelines*

savetostr(*sortkey=True*)
Save configurations to a single string

vlcp.config.config.**config**(*key*)
Decorator to map this class directly to a configuration node. It uses *<parentbase>.key* for configuration base and configuration mapping.

vlcp.config.config.**configbase**(*key*)
Decorator to set this class to configuration base class. A configuration base class uses *<parentbase>.key.* for its configuration base, and uses *<parentbase>.key.default* for configuration mapping.

vlcp.config.config.**defaultconfig**(*cls*)
Generate a default configuration mapping bases on the class name. If this class does not have a parent with *configbase* defined, it is set to a configuration base with *configbase=<lowercase-name>* and *configkey=<lowercase-name>.default*; otherwise it inherits *configbase* of its parent and set *configkey=<parentbase>.<lowercase-name>*

Refer to :ref::*configurations* for normal rules.

5.2 vlcp.event

Note: This document is generated from the source file.

View Source on GitHub

5.2.1 vlcp.event.connection

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/6/19

author hubo

class vlcp.event.connection.**Client**(*url*, *protocol*, *scheduler=None*, *key=None*, *certificate=None*, *ca_certs=None*, *bindaddress=None*)

A single connection to a specified target

__init__(*url*, *protocol*, *scheduler=None*, *key=None*, *certificate=None*, *ca_certs=None*, *bindaddress=None*)

Constructor

__repr__(**args*, ***kwargs*)

Return repr(self).

main()

The main routine method, should be rewritten to an async method

class vlcp.event.connection.**Connection**(*protocol*, *sockobj=None*, *scheduler=None*)

A connection on a socket

__init__(*protocol*, *sockobj=None*, *scheduler=None*)

Constructor

__repr__(**args*, ***kwargs*)

Return repr(self).

main()

The main routine method, should be rewritten to an async method

reconnect(*force=True*, *connmark=None*)

Can call without delegate

reset(*force=True*, *connmark=None*)

Can call without delegate

shutdown(*force=False*, *connmark=-1*)

Can call without delegate

write(*event*, *ignoreException=True*)

Can call without delegate

class vlcp.event.connection.**ConnectionControlEvent**(**args*, ***kwargs*)

exception vlcp.event.connection.**ConnectionResetException**

class vlcp.event.connection.**ConnectionWriteEvent**(**args*, ***kwargs*)
Event used to send data to a connection

canignorenow()
Extra criteria for an event with canignore = False. When this event returns True, the event is safely ignored.

class vlcp.event.connection.**ResolveRequestEvent**(**args*, ***kwargs*)

class vlcp.event.connection.**ResolveResponseEvent**(**args*, ***kwargs*)

class vlcp.event.connection.**TcpServer**(*url*, *protocol*, *scheduler=None*, *key=None*, *certificate=None*, *ca_certs=None*)
A server receiving multiple connections

__init__(*url*, *protocol*, *scheduler=None*, *key=None*, *certificate=None*, *ca_certs=None*)
Create the routine container.

Parameters

- **scheduler** – The scheduler. This must be set; if None is used, it must be set with *container.bind(scheduler)* before using.
- **daemon** – If daemon = True, the main routine *container.main* is set to be a daemon routine. A daemon routine does not stop the scheduler from quitting; if all non-daemon routines are quit, the scheduler stops.

__repr__(**args*, ***kwargs*)
Return repr(self).

main()
The main routine method, should be rewritten to an async method

shutdown(*connmark=-1*)
Can call without delegate

startlisten(*connmark=-1*)
Can call without delegate

stoplisten(*connmark=-1*)
Can call without delegate

5.2.2 vlcp.event.core

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/6/12

author hubo

exception vlcp.event.core.**InterruptedBySignalException**
In Python 3.x, we must raise an exception to interrupt the polling, or it will be automatically retried

class vlcp.event.core.**PollEvent**(**args*, ***kwargs*)

exception vlcp.event.core.**QuitException**

class vlcp.event.core.**Scheduler**(*polling=None*, *processevents=None*, *queuedefault=None*, *queuemax=None*, *defaultQueueClass=<class 'vlcp.event.pqueue.CBQueue.FifoQueue'>*, *defaultQueuePriority=0*)

Event-driven scheduler

__init__(*polling=None*, *processevents=None*, *queuedefault=None*, *queuemax=None*, *defaultQueueClass=<class 'vlcp.event.pqueue.CBQueue.FifoQueue'>*, *defaultQueuePriority=0*)

Constructor

Parameters

- **polling** – a polling source to retrieve events
- **processevents** – max events processed before starting another poll
- **queuedefault** – max length of default queue
- **queuemax** – total max length of the event queue
- **defaultQueueClass** – default queue class, see CBQueue
- **defaultQueuePriority** – default queue priority, see CBQueue

cancelTimer(*timer*)

Cancel the timer

Parameters **timer** – the timer handle

emergesend(*event*)

Force send a new event to the main queue.

ignore(*matcher*)

Unblock and ignore the matched events, if any.

main(*installsignal=True*, *sendinit=True*)

Start main loop

modifyPolling(*fd*, *options*)

modify options of a registered file descriptor

quit(*daemononly=False*)

Send quit event to quit the main loop

register(*matchers*, *runnable*)

Register an iterator(runnable) to scheduler and wait for events

Parameters

- **matchers** – sequence of EventMatchers
- **runnable** – an iterator that accept send method
- **daemon** – if True, the runnable will be registered as a daemon.

registerPolling(*fd*, *options=5*, *daemon=False*)

register a polling file descriptor

Parameters

- **fd** – file descriptor or socket object
- **options** – bit mask flags. Polling object should ignore the incompatible flag.

send(*event*)

Send a new event to the main queue. If the queue or sub-queue is full, return a wait event

Returns None if succeeded. Matcher for QueueCanWriteEvent if sub-queue is full.

setDaemon(*runnable*, *isdaemon*, *noregister=False*)
If a runnable is a daemon, it will not keep the main loop running. The main loop will end when all alived runnables are daemons.

setTimer(*start*, *interval=None*)
Generate a TimerEvent on specified time

Parameters

- **start** – offset for time from now (seconds), or datetime for a fixed time
- **interval** – if not None, the timer is regenerated by interval seconds.

Returns a timer handle to wait or cancel the timer

syscall(*func*)
Call the func in core context (main loop).

func should like:

```
def syscall_sample(scheduler, processor):
    something...
```

where processor is a function which accept an event. When calling processor, scheduler directly process this event without sending it to queue.

An event matcher is returned to the caller, and the caller should wait for the event immediately to get the return value from the system call. The SyscallReturnEvent will have 'retvalue' as the return value, or 'exception' as the exception thrown: (type, value, traceback)

Parameters **func** – syscall function

Returns an event matcher to wait for the SyscallReturnEvent. If None is returned, a syscall is already scheduled; return to core context at first.

unregister(*matchers*, *runnable*)
Unregister an iterator(runnable) and stop waiting for events

Parameters

- **matchers** – sequence of EventMatchers
- **runnable** – an iterator that accept send method

unregisterPolling(*fd*, *daemon=False*)
Unregister a polling file descriptor

Parameters **fd** – file descriptor or socket object

unregisterall(*runnable*)
Unregister all matches and detach the runnable. Automatically called when runnable returns StopIteration.

wantContinue()
The next main loop will generate a SystemControlEvent('continue'), allowing time-consuming jobs to suspend and let other threads do their work

yield_(*runnable*)
Pend this runnable to be wake up later

class vlcp.event.core.**SyscallReturnEvent**(**args*, ***kwargs*)

class vlcp.event.core.**SystemControlEvent**(**args*, ***kwargs*)

class vlcp.event.core.**SystemControlLowPriorityEvent**(**args*, ***kwargs*)

class vlcp.event.core.**TimerEvent**(**args*, ***kwargs*)

vlcp.event.core.**syscall_clearqueue**(*queue*)
Clear a queue

vlcp.event.core.**syscall_clearremovequeue**(*queue*, *index*)
Clear the subqueue *queue[index]* and remove it from queue.

vlcp.event.core.**syscall_direct**(**events*)
Directly process these events. This should never be used for normal events.

vlcp.event.core.**syscall_generator**(*generator*)
Directly process events from a generator function. This should never be used for normal events.

vlcp.event.core.**syscall_removequeue**(*queue*, *index*)
Remove subqueue *queue[index]* from queue.

5.2.3 vlcp.event.event

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/06/01

author hubo

class vlcp.event.event.**DiffRef_**(*origin*, *add*)
Append some matchers to a diff without breaking the difference structure

__init__(*origin*, *add*)
Initialize self. See help(type(self)) for accurate signature.

two_way_difference(*b*)
Return (self - b, b - self)

class vlcp.event.event.**Diff_**(*base=()*, *add=()*, *remove=()*)
Special "differenced" set. Items in 'base', 'add', 'remove' must not be same

Used by *wait_for_all*

__init__(*base=()*, *add=()*, *remove=()*)
Initialize self. See help(type(self)) for accurate signature.

two_way_difference(*b*, *extra_add=()*, *extra_remove=()*)
Return (self - b, b - self)

class vlcp.event.event.**Event**(**args*, ***kwargs*)
A generated event with indices

__init__(**args*, ***kwargs*)

Parameters

- **args** – index values like 12,"read",... content are type-depended.
- **kwargs** –

  indices input indices by name

canignore if the event is not processed, whether it is safe to ignore the event.

If it is not, the processing queue might be blocked to wait for a proper event processor. Default to True.

others the properties will be set on the created event

__repr__()
Return repr(self).

canignorenow()
Extra criteria for an event with canignore = False. When this event returns True, the event is safely ignored.

classmethod createMatcher(**args*, ***kwargs*)

Parameters

- **_ismatch** – user-defined function ismatch(event) for matching test
- ***args** – indices
- ****kwargs** – index_name=index_value for matching criteria

classmethod getTypename()

Returns return the proper name to match

classmethod indicesNames()

Returns names of indices

class vlcp.event.event.**EventMatcher**(*indices*, *judgeFunc=None*)
A matcher to match an event

__await__()
event = yield from matcher

or

event = await matcher

__init__(*indices*, *judgeFunc=None*)
Initialize self. See help(type(self)) for accurate signature.

__repr__()
Return repr(self).

exception vlcp.event.event.**IsMatchExceptionWarning**

class vlcp.event.event.**M_**(**matchers*)
Awaitable object for multiple matchers

```
` event, matcher = await M_(matcher1, matcher2) `
```

__await__()
```
` event, matcher = yield from M_(matcher1, matcher2) `
```

equivalent to

```
` event, matcher = yield (matcher1, matcher2) `
```

__init__(**matchers*)
Initialize self. See help(type(self)) for accurate signature.

vlcp.event.event.**withIndices**(**args*)
Create indices for an event class. Every event class must be decorated with this decorator.

vlcp.event.event.**with_indices**(**args*)
Create indices for an event class. Every event class must be decorated with this decorator.

5.2.4 vlcp.event.future

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/9/28

author hubo

Future is a helper class to simplify the process of retrieving a result from other routines. The implementation is straight-forward: first check the return value, if not set, wait for the event. Multiple routines can wait for the same Future object.

The interface is similar to asyncio, but:

- Cancel is not supported - you should terminate the sender routine instead. But *RoutineFuture* supports *close()* (and *cancel()* which is the same)
- Callback is not supported - start a subroutine to wait for the result instead.
- *result()* returns None if the result is not ready; *exception()* is not supported.
- New *wait()* async function: get the result, or wait for the result until available. It is always the recommended way to use a future; *result()* is not recommended.

  wait() will NOT cancel the *Future* (or *RoutineFuture*) when the waiting coroutine is closed. This is different from *asyncio.Future*. To ensure that the future closes after awaited, use *wait_and_close()* of *RoutineFuture*.
- *ensure_result()* returns a context manager: this should be used in the sender routine, to ensure that a result is always set after exit the with scope. If the result is not set, it is set to None; if an exception is raised, it is set with set_exception.

Since v2.0, you can directly use *await future* to wait for the result

class vlcp.event.future.**Future**(*scheduler*)
Basic future object

__init__(*scheduler*)
Initialize self. See help(type(self)) for accurate signature.

done()

Returns True if the result is available; False otherwise.

ensure_result(*supress_exception=False*, *defaultresult=None*)
Context manager to ensure returning the result

result()

Returns None if the result is not ready, the result from set_result, or raise the exception from set_exception. If the result can be None, it is not possible to tell if the result is available; use done() to determine that.

set_exception(*exception*)
Set an exception to Future object, wake up all the waiters

Parameters **exception** – exception to set

set_result(*result*)
Set the result to Future object, wake up all the waiters

Parameters **result** – result to set

wait(*container=None*)

Parameters **container** – DEPRECATED container of current routine

Returns The result, or raise the exception from set_exception.

exception vlcp.event.future.**FutureCancelledException**

class vlcp.event.future.**FutureEvent**(**args*, ***kwargs*)

class vlcp.event.future.**RoutineFuture**(*subprocess*, *container*)
Quick wrapper to create a subroutine and return the result to a Future object

__init__(*subprocess*, *container*)
Start the subprocess

Parameters

- **subprocess** – a generator process, which returns the result to future on exit
- **container** – the routine container to run the subprocess with

cancel()
Same as close()

close()
Terminate the subprocess

wait_and_close()
wait for result; always close no matter success or failed

5.2.5 vlcp.event.lock

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/12/14

author hubo

Lock is created by limiting queue length of LockEvent, so there is only one event in the queue. Other send request is blocked by the queue.

class vlcp.event.lock.**Lock**(*key*, *scheduler*, *context='default'*)
An lock object. Normal usage:

```
my_lock = Lock(lock_obj, container.scheduler)
await my_lock.lock(container)
with my_lock:
    ...
```

Or use async with:

```
async with my_lock:
    ...
```

__init__(*key*, *scheduler*, *context='default'*)
Create a lock object. You do not need to share this object with other routines; all locks with the same *key* and *context* are mutual exclusive.

Parameters

- **key** – Any hashable value.
- **scheduler** – The scheduler
- **context** – An extra object to separate keys into different context

beginlock(*container*)
Start to acquire lock in another routine. Call trylock or lock later to acquire the lock. Call unlock to cancel the lock routine

lock(*container=None*)
Wait for lock acquire

trylock()
Try to acquire lock and return True; if cannot acquire the lock at this moment, return False.

unlock()
Unlock the key

class vlcp.event.lock.**LockEvent**(**args*, ***kwargs*)

canignorenow()
Extra criteria for an event with canignore = False. When this event returns True, the event is safely ignored.

class vlcp.event.lock.**LockedEvent**(**args*, ***kwargs*)

class vlcp.event.lock.**Semaphore**(*key*, *size*, *scheduler*, *context='default'*, *priority=1000*)
Change the default behavior of Lock for specified context and key from lock to semaphore. The default behavior of Lock allows only one routine for a specified key; when a semaphore is created, limited number of routines can retrieve the Lock at the same time.

__init__(*key*, *size*, *scheduler*, *context='default'*, *priority=1000*)
Prepare to change locks on *key* and *context* to a semaphore.

Parameters

- **key** – Hashable object used by locks.
- **size** – Semaphore size, which means the maximum allowed routines to retrieve the lock at the same time
- **scheduler** – The scheduler
- **context** – context object used by locks.
- **priority** – priority for the created queue.

create()
Create the subqueue to change the default behavior of Lock to semaphore.

destroy(*container=None*)
Destroy the created subqueue to change the behavior back to Lock

5.2.6 vlcp.event.matchtree

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/06/01

author hubo

class vlcp.event.matchtree.**EventTree**(*parent=None*, *branch=5*)
Store events; match matchers

__init__(*parent=None*, *branch=5*)
Constructor

subtree(*event*, *create=False*)
Find a subtree from an event

class vlcp.event.matchtree.**MatchTree**(*parent=None*)
A dictionary tree for fast event match

__init__(*parent=None*)
Constructor

insert(*matcher*, *obj*)
Insert a new matcher

Parameters

- **matcher** – an EventMatcher
- **obj** – object to return

matches(*event*)
Return all matches for this event. The first matcher is also returned for each matched object.

Parameters **event** – an input event

matchesWithMatchers(*event*)
Return all matches for this event. The first matcher is also returned for each matched object.

Parameters **event** – an input event

matchfirst(*event*)
Return first match for this event

Parameters **event** – an input event

matchfirstwithmatcher(*event*)
Return first match with matcher for this event

Parameters **event** – an input event

remove(*matcher*, *obj*)
Remove the matcher

Parameters

- **matcher** – an EventMatcher
- **obj** – the object to remove

subtree(*matcher*, *create=False*)
Find a subtree from a matcher

Parameters

- **matcher** – the matcher to locate the subtree. If None, return the root of the tree.
- **create** – if True, the subtree is created if not exists; otherwise return None if not exists

5.2.7 vlcp.event.polling

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/6/17

author hubo

class vlcp.event.polling.**EPollPolling**(*options=2147483648*, *maxwait=60*)
Poll event from epoll

__init__(*options=2147483648*, *maxwait=60*)
Constructor

class vlcp.event.polling.**SelectPolling**(*options=0*, *maxwait=60*)
Compatible event polling with select

__init__(*options=0*, *maxwait=60*)
Constructor

5.2.8 vlcp.event.pqueue

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/06/02

author hubo

class vlcp.event.pqueue.**AutoClassQueueCanWriteEvent**(*queue*, *key*, *newonly*, *firstonly*)

__init__(*queue*, *key*, *newonly*, *firstonly*)

Parameters

- **args** – index values like 12,"read",... content are type-depended.
- **kwargs** –

  indices input indices by name

  canignore if the event is not processed, whether it is safe to ignore the event.

  If it is not, the processing queue might be blocked to wait for a proper event processor. Default to True.

others the properties will be set on the created event

class vlcp.event.pqueue.**CBQueue**(*tree=None*, *parent=None*, *maxdefault=None*, *maxtotal=None*, *defaultQueueClass=<class 'vlcp.event.pqueue.CBQueue.FifoQueue'>*, *defaultQueuePriority=0*)

A multi-queue model with priority and balance. When first created, there is a default queue with priority 0. More sub-queues maybe created with addSubQueue. Each sub-queue is a CBQueue which accepts more sub-queues. Sub-queues are considered as black-box to the outer parent.

class AutoClassQueue(*parent=None*, *maxlength=None*, *key='owner'*, *preserveForNew=1*, *maxstat=None*, *subqueuelimit=None*)

A queue classify events into virtual sub-queues by key

__init__(*parent=None*, *maxlength=None*, *key='owner'*, *preserveForNew=1*, *maxstat=None*, *subqueuelimit=None*)

Each value are classified and put into virtual subqueues.

Parameters

- **parent** – parent queue
- **maxlength** – total limit for this queue
- **key** – classify value by this attribute
- **preserveForNew** – preserve some space for new subqueues. When putting values with old keys into the queue, the queue would block the value when it has size *maxlength - preserveForNew*. But, a value with new key (thus creates a new virtual queue) will be accepted by this queue. It makes sure a value with new key has enough priority to be enqueued.
- **maxstat** – Object passes each virtual queue lately are counted. When a virtual subqueue is passing less events, the priority of this queue is increased to make it fair. *maxstat* is the limit of objects counted
- **subqueuelimit** – limit of each subqueue

class FifoQueue(*parent=None*, *maxlength=None*)

A wrapper for a FIFO queue

__init__(*parent=None*, *maxlength=None*)

Initialize self. See help(type(self)) for accurate signature.

class MultiQueue(*parent=None*, *priority=0*)

A multi-queue container, every queue in a multi-queue has the same priority, and is popped in turn.

class CircleListNode(*value*)

Circle link list

__init__(*value*)

Initialize self. See help(type(self)) for accurate signature.

__init__(*parent=None*, *priority=0*)

Initialize self. See help(type(self)) for accurate signature.

class PriorityQueue(*parent=None*, *maxlength=None*, *key='priority'*)

A queue with inner built priority. Event must have a "priority" property to use with this type of queue. For fail-safe, events without "priority" property have the lowest priority.

NOTICE: different from the queue priority, the priority property is smaller-higher, and is not limited to integers. This allows datetime to be used as an increasing priority

__init__(*parent=None*, *maxlength=None*, *key='priority'*)
Initialize self. See help(type(self)) for accurate signature.

__getitem__(*name*)
Get a sub-queue through q['sub-queue-name']

__init__(*tree=None*, *parent=None*, *maxdefault=None*, *maxtotal=None*, *defaultQueueClass=<class 'vlcp.event.pqueue.CBQueue.FifoQueue'>*, *defaultQueuePriority=0*)
Constructor

addSubQueue(*priority*, *matcher*, *name=None*, *maxdefault=None*, *maxtotal=None*, *defaultQueueClass=<class 'vlcp.event.pqueue.CBQueue.FifoQueue'>*)
add a sub queue to current queue, with a priority and a matcher

Parameters

- **priority** – priority of this queue. Larger is higher, 0 is lowest.
- **matcher** – an event matcher to catch events. Every event match the criteria will be stored in this queue.
- **name** – a unique name to identify the sub-queue. If none, the queue is anonymous. It can be any hashable value.
- **maxdefault** – max length for default queue.
- **maxtotal** – max length for sub-queue total, including sub-queues of sub-queue

append(*event*, *force=False*)
Append an event to queue. The events are classified and appended to sub-queues

Parameters

- **event** – input event
- **force** – if True, the event is appended even if the queue is full

Returns None if appended successfully, or a matcher to match a QueueCanWriteEvent otherwise

block(*event*, *emptyEvents=()*)
Return a recently popped event to queue, and block all later events until unblock.

Only the sub-queue directly containing the event is blocked, so events in other queues may still be processed. It is illegal to call block and unblock in different queues with a same event.

Parameters

- **event** – the returned event. When the queue is unblocked later, this event will be popped again.
- **emptyEvents** – reactivate the QueueIsEmptyEvents

canAppend()
Whether the queue is full or not. Only check the total limit. Sub-queue may still be full (even default).

Returns False if the queue is full, True if not. If there are sub-queues, append() may still fail if the sub-queue is full.

canPop()
Whether the queue is empty/blocked or not

Returns False if the queue is empty or blocked, or True otherwise

clear()
Clear all the events in this queue, including any sub-queues.

Returns ((queueEvents,...), (queueEmptyEvents,...)) where queueEvents are QueueCanWriteEvents generated by clearing.

getPriority(*queue*)
get priority of a sub-queue

notifyAppend(*queue*, *force*)
Internal notify for sub-queues

Returns If the append is blocked by parent, an EventMatcher is returned, None else.

notifyBlock(*queue*, *blocked*)
Internal notify for sub-queues been blocked

notifyPop(*queue*, *length=1*)
Internal notify for sub-queues been poped

Returns List of any events generated by this pop

pop()
Pop an event from the queue. The event in the queue with higher priority is popped before ones in lower priority. If there are multiple queues with the same priority, events are taken in turn from each queue. May return some queueEvents indicating that some of the queues can be written into.

Returns *(obj, (queueEvents,...), (queueEmptyEvents,...))* where obj is the popped event, queueEvents are QueueCanWriteEvents generated by this pop and queueEmptyEvents are QueueIsEmptyEvents generated by this pop

removeSubQueue(*queue*)
remove a sub queue from current queue.

This unblock the sub-queue, retrieve all events from the queue and put them back to the parent.

Call clear on the sub-queue first if the events are not needed any more.

Parameters **queue** – the name or queue object to remove

Returns ((queueevents,...), (queueEmptyEvents,...)) Possible queue events from removing sub-queues

setPriority(*queue*, *priority*)
Set priority of a sub-queue

unblock(*event*)
Remove a block

unblockall()
Remove all blocks from the queue and all sub-queues

unblockqueue(*queue*)
Remove blocked events from the queue and all subqueues. Usually used after queue clear/unblockall to prevent leak.

Returns the cleared events

waitForEmpty()
Make this queue generate a QueueIsEmptyEvent when it is empty

Returns matcher for QueueIsEmptyEvent, or None if the queue is already empty

class vlcp.event.pqueue.**QueueCanWriteEvent**(*queue*, ***kwargs*)

__init__(*queue*, ***kwargs*)

Parameters

- **args** – index values like 12,"read",... content are type-depended.
- **kwargs** –

  indices input indices by name

  canignore if the event is not processed, whether it is safe to ignore the event.

  If it is not, the processing queue might be blocked to wait for a proper event processor. Default to True.

  others the properties will be set on the created event

class vlcp.event.pqueue.**QueueIsEmptyEvent**(**args*, ***kwargs*)

5.2.9 vlcp.event.ratelimiter

Note: This document is generated from the source file.

View Source on GitHub

Created on 2018/4/19

Preventing many time-consuming operations to be done in the same loop

author hubo

class vlcp.event.ratelimiter.**RateLimiter**(*limit*, *container*)

Limit operations executed in current loop, ensure sockets are still processed in time-consuming operations

__init__(*limit*, *container*)

Parameters

- **limit** – "resources" limited in a single loop. "resources" can be any countable things like operations executed or bytes sent
- **container** – a *RoutineContainer*

limit(*use=1*)

Acquire "resources", wait until enough "resources" are acquired. For each loop, *limit* number of "resources" are permitted.

Parameters **use** – number of "resouces" to be used.

class vlcp.event.ratelimiter.**RateLimitingEvent**(**args*, ***kwargs*)

5.2.10 vlcp.event.runnable

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/6/16

author hubo

class vlcp.event.runnable.**EventHandler**(*scheduler=None*, *daemon=False*)
Runnable with an event handler model.

__init__(*scheduler=None*, *daemon=False*)
Initialize self. See help(type(self)) for accurate signature.

__iter__()
Keep it like a iterator. Not very useful.

__next__()
Python 3 next

next()
Keep it like a iterator. Not very useful.

registerAllHandlers(*handlerDict*)
Register self to scheduler

registerHandler(*matcher*, *handler*)
Register self to scheduler

send(*etup*)
Handle events

exception vlcp.event.runnable.**GeneratorExit_**
Bypass PyPy3 bug

exception vlcp.event.runnable.**IllegalMatchersException**

exception vlcp.event.runnable.**MultipleException**(*exceptions*)
Special exception type raised from :py:method::*vlcp.event.runnable.RoutineContainer.executeAll*

__init__(*exceptions*)
Initialize self. See help(type(self)) for accurate signature.

vlcp.event.runnable.**Routine**(*coroutine*, *scheduler*, *asyncStart=True*, *container=None*, *manualStart=False*, *daemon=False*)
This wraps a normal coroutine to become a VLCP routine. Usually you do not need to call this yourself; *container.start* and *container.subroutine* calls this automatically.

class vlcp.event.runnable.**RoutineContainer**(*scheduler=None*, *daemon=False*)
A routine container groups several routines together and shares data among them. It is also used to pass important information like events, matchers and return values to the routine.

Several attributes are commonly used:
currentroutine Always set to the executing routine - which is the wrapped routine object of the routine itself
mainroutine Set to the main routine *container.main* (started by *container.start*)
__init__(*scheduler=None*, *daemon=False*)
Create the routine container.

Parameters

- **scheduler** – The scheduler. This must be set; if None is used, it must be set with *container.bind(scheduler)* before using.
- **daemon** – If daemon = True, the main routine *container.main* is set to be a daemon routine. A daemon routine does not stop the scheduler from quitting; if all non-daemon routines are quit, the scheduler stops.

beginDelegateOther(*subprocess*, *container*, *retnames=('')*)
DEPRECATED Start the delegate routine, but do not wait for result, instead returns a (matcher routine) tuple. Useful for advanced delegates (e.g. delegate multiple subprocesses in the same time). This is NOT a coroutine method.

Parameters

- **subprocess** – a coroutine
- **container** – container in which to start the routine
- **retnames** – get return values from keys. '' for the return value (for compatibility with earlier versions)

Returns (matcher, routine) where matcher is a event matcher to get the delegate result, routine is the created routine

begin_delegate(*subprocess*)

Start the delegate routine, but do not wait for result, instead returns a (matcher, routine) tuple. Useful for advanced delegates (e.g. delegate multiple subprocesses in the same time). This is NOT a coroutine method.

WARNING: this is not a safe way for asynchronous executing and get the result. Use *RoutineFuture* instead.

Parameters **subprocess** – a coroutine

Returns (matcher, routine) where matcher is a event matcher to get the delegate result, routine is the created routine

begin_delegate_other(*subprocess*, *container*, *retnames=('',)*)

DEPRECATED Start the delegate routine, but do not wait for result, instead returns a (matcher routine) tuple. Useful for advanced delegates (e.g. delegate multiple subprocesses in the same time). This is NOT a coroutine method.

Parameters

- **subprocess** – a coroutine
- **container** – container in which to start the routine
- **retnames** – get return values from keys. '' for the return value (for compatibility with earlier versions)

Returns (matcher, routine) where matcher is a event matcher to get the delegate result, routine is the created routine

bind(*scheduler*)

If scheduler is not specified, bind the scheduler

close()

Same as *terminate()*

delegate(*subprocess*, *forceclose=False*)

Run a subprocess without container support

Many subprocess assume itself running in a specified container, it uses container reference like self.events. Calling the subprocess in other containers will fail.

With delegate, you can call a subprocess in any container (or without a container):

```
r = await c.delegate(c.someprocess())
```

Returns original return value

delegateOther(*subprocess*, *container*, *retnames=('',)*, *forceclose=False*)

DEPRECATED Another format of delegate allows delegate a subprocess in another container, and get some returning values the subprocess is actually running in 'container'.

```
ret = await self.delegate_other(c.method(), c)
```

Returns a tuple for retnames values

delegate_other(*subprocess*, *container*, *retnames=('',)*, *forceclose=False*)
DEPRECATED Another format of delegate allows delegate a subprocess in another container, and get some returning values the subprocess is actually running in 'container'.

```
ret = await self.delegate_other(c.method(), c)
```

Returns a tuple for retnames values

classmethod destroy_container_cache(*scheduler*)
Remove cached container

doEvents()
Suspend this routine until the next polling. This can be used to give CPU time for other routines and socket processings in long calculating procedures. Can call without delegate.

do_events()
Suspend this routine until the next polling. This can be used to give CPU time for other routines and socket processings in long calculating procedures. Can call without delegate.

end_delegate(*delegate_matcher*, *routine=None*, *forceclose=False*)
Retrieve a begin_delegate result. Must be called immediately after begin_delegate before any other *await*, or the result might be lost.

Do not use this method without thinking. Always use *RoutineFuture* when possible.

executeAll(*subprocesses*, *container=None*, *retnames=('',)*, *forceclose=True*)
DEPRECATED Execute all subprocesses and get the return values.

Parameters

- **subprocesses** – sequence of subroutines (coroutines)
- **container** – if specified, run subprocesses in another container.
- **retnames** – DEPRECATED get return value from container.(name) for each name in retnames. '' for return value (to be compatible with earlier versions)
- **forceclose** – force close the routines on exit, so all the subprocesses are terminated on timeout if used with executeWithTimeout

Returns a list of tuples, one for each subprocess, with value of retnames inside: *[('retvalue1',),('retvalue2',),...]*

executeWithTimeout(*timeout*, *subprocess*)
Execute a subprocess with timeout. If time limit exceeds, the subprocess is terminated, and *is_timeout* is set to True; otherwise the *is_timeout* is set to False.

You can uses *execute_with_timeout* with other help functions to create time limit for them:

```
timeout, result = await container.execute_with_timeout(10, container.execute_
↪all([routine1(), routine2()]))
```

Returns (is_timeout, result) When is_timeout = True, result = None

execute_all(*subprocesses*, *forceclose=True*)
Execute all subprocesses and get the return values.

Parameters

- **subprocesses** – sequence of subroutines (coroutines)
- **forceclose** – force close the routines on exit, so all the subprocesses are terminated on timeout if used with executeWithTimeout

Returns a list of return values for each subprocess

execute_all_with_names(*subprocesses*, *container=None*, *retnames=('',)*, *forceclose=True*)
DEPRECATED Execute all subprocesses and get the return values.

Parameters

- **subprocesses** – sequence of subroutines (coroutines)
- **container** – if specified, run subprocesses in another container.
- **retnames** – DEPRECATED get return value from container.(name) for each name in retnames. '' for return value (to be compatible with earlier versions)
- **forceclose** – force close the routines on exit, so all the subprocesses are terminated on timeout if used with executeWithTimeout

Returns a list of tuples, one for each subprocess, with value of retnames inside: *[('retvalue1',),('retvalue2',),...]*

execute_with_timeout(*timeout*, *subprocess*)
Execute a subprocess with timeout. If time limit exceeds, the subprocess is terminated, and *is_timeout* is set to True; otherwise the *is_timeout* is set to False.

You can uses *execute_with_timeout* with other help functions to create time limit for them:

```
timeout, result = await container.execute_with_timeout(10, container.execute_
↪all([routine1(), routine2()]))
```

Returns (is_timeout, result) When is_timeout = True, result = None

classmethod get_container(*scheduler*)
Create temporary instance for helper functions

main()
The main routine method, should be rewritten to an async method

start(*asyncStart=False*)
Start *container.main* as the main routine.

Parameters **asyncStart** – if True, start the routine in background. By default, the routine starts in foreground, which means it is executed to the first *yield* statement before returning. If the started routine raises an exception, the exception is re-raised to the caller of *start*

subroutine(*iterator*, *asyncStart=True*, *name=None*, *daemon=False*)
Start extra routines in this container.

Parameters

- **iterator** – A coroutine object i.e the return value of an async method *my_routine()*

- **asyncStart** – if False, start the routine in foreground. By default, the routine starts in background, which means it is not executed until the current caller reaches the next *yield* statement or quit.
- **name** – if not None, *container.<name>* is set to the routine object. This is useful when you want to terminate the routine from outside.
- **daemon** – if True, this routine is set to be a daemon routine. A daemon routine does not stop the scheduler from quitting; if all non-daemon routines are quit, the scheduler stops.

syscall(*func*, *ignoreException=False*)
Call a syscall method and retrieve its return value

syscall_noreturn(*func*)
Call a syscall method. A syscall method is executed outside of any routines, directly in the scheduler loop, which gives it chances to directly operate the event loop. See :py:method::*vlcp.event.core.Scheduler.syscall*.

terminate(*routine=None*)
Stop a routine.

Parameters **routine** – if None, stop the main routine. If not None, it should be a routine object. You can specify a name for a subroutine, and use *container.<name>* to retrieve it.

waitForAll(**matchers*, *eventlist=None*, *eventdict=None*, *callback=None*)
Wait until each matcher matches an event. When this coroutine method returns, *eventlist* is set to the list of events in the arriving order (may not be the same as the matchers); *eventdict* is set to a dictionary *{matcher1: event1, matcher2: event2, ...}*

Parameters

- **eventlist** – use external event list, so when an exception occurs (e.g. routine close), you can retrieve the result from the passed-in list
- **eventdict** – use external event dict
- **callback** – if not None, the callback should be a callable callback(event, matcher) which is called each time an event is received

Returns (eventlist, eventdict)

waitForAllEmpty(**queues*)
Wait for multiple queues to be empty at the same time.

Require delegate when calling from coroutines running in other containers

waitForAllToProcess(**matchers*, *eventlist=None*, *eventdict=None*, *callback=None*)
Similar to *waitForAll*, but set *canignore=True* for these events. This ensures blocking events are processed correctly.

waitForEmpty(*queue*)
Wait for a queue to be empty. Can call without delegate

waitForSend(*event*, *, *until=None*)
Send an event to the main event queue. Can call without delegate.

Parameters **until** – if the callback returns True, stop sending and return

Returns the last True value the callback returns, or None

waitWithTimeout(*timeout*, **matchers*)
Wait for multiple event matchers, or until timeout.

Parameters

- **timeout** – a timeout value
- ***matchers** – event matchers

Returns (is_timeout, event, matcher). When is_timeout = True, event = matcher = None.

wait_for_all(**matchers*, *eventlist=None*, *eventdict=None*, *callback=None*)
Wait until each matcher matches an event. When this coroutine method returns, *eventlist* is set to the list of events in the arriving order (may not be the same as the matchers); *eventdict* is set to a dictionary *{matcher1: event1, matcher2: event2, ...}*

Parameters

- **eventlist** – use external event list, so when an exception occurs (e.g. routine close), you can retrieve the result from the passed-in list
- **eventdict** – use external event dict
- **callback** – if not None, the callback should be a callable callback(event, matcher) which is called each time an event is received

Returns (eventlist, eventdict)

wait_for_all_empty(**queues*)
Wait for multiple queues to be empty at the same time.

Require delegate when calling from coroutines running in other containers

wait_for_all_to_process(**matchers*, *eventlist=None*, *eventdict=None*, *callback=None*)
Similar to *waitForAll*, but set *canignore=True* for these events. This ensures blocking events are processed correctly.

wait_for_empty(*queue*)
Wait for a queue to be empty. Can call without delegate

wait_for_send(*event*, ***, *until=None*)
Send an event to the main event queue. Can call without delegate.

Parameters **until** – if the callback returns True, stop sending and return

Returns the last True value the callback returns, or None

wait_with_timeout(*timeout*, **matchers*)
Wait for multiple event matchers, or until timeout.

Parameters

- **timeout** – a timeout value
- ***matchers** – event matchers

Returns (is_timeout, event, matcher). When is_timeout = True, event = matcher = None.

withCallback(*subprocess*, *callback*, **matchers*, *intercept_callback=None*)
Monitoring event matchers while executing a subprocess. *callback(event, matcher)* is called each time an event is matched by any event matchers. If the callback raises an exception, the subprocess is terminated.

Parameters **intercept_callback** – a callback called before a event is delegated to the inner subprocess

withException(*subprocess*, **matchers*)
Monitoring event matchers while executing a subprocess. If events are matched before the subprocess ends, the subprocess is terminated and a RoutineException is raised.

with_callback(*subprocess*, *callback*, **matchers*, *intercept_callback=None*)
Monitoring event matchers while executing a subprocess. *callback(event, matcher)* is called each time an event is matched by any event matchers. If the callback raises an exception, the subprocess is terminated.

Parameters **intercept_callback** – a callback called before a event is delegated to the inner subprocess

with_exception(*subprocess*, **matchers*)
Monitoring event matchers while executing a subprocess. If events are matched before the subprocess ends, the subprocess is terminated and a RoutineException is raised.

class vlcp.event.runnable.**RoutineControlEvent**(**args*, ***kwargs*)

exception vlcp.event.runnable.**RoutineException**(*matcher*, *event*)
Special exception type raised from :py:method::*vlcp.event.runnable.RoutineContainer.withException*. e.matcher is set to the matcher and e.event is set to the matched event.

__init__(*matcher*, *event*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.event.runnable.**generatorwrapper**(*run*, *name='coroutine'*, *classname='routine'*)
Default __repr__ of a generator is not readable, use a wrapper to improve the readability

__init__(*run*, *name='coroutine'*, *classname='routine'*)
Initialize self. See help(type(self)) for accurate signature.

__repr__(**args*, ***kwargs*)
Return repr(self).

5.2.11 vlcp.event.stream

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/8/14

author hubo

class vlcp.event.stream.**BaseStream**(*isunicode=False*, *encoders=[]*)
Streaming base

__init__(*isunicode=False*, *encoders=[]*)
Constructor

Parameters

- **isunicode** – if True, the data used in this stream outputs unicode string. Otherwise it outputs bytes.
- **encoders** – a list of functions *enc(data, is_eof)* which encodes input data.

close(*scheduler*, *allowwrite=False*)
Read side close. To close at the write side, use *eof=True* with *write*.

Parameters

- **scheduler** – the scheduler
- **allowwrite** – do not raise exception on the write side

copyTo(*dest*, *container*, *buffering=True*)
Coroutine method to copy content from this stream to another stream.

copy_to(*dest*, *container*, *buffering=True*)
Coroutine method to copy content from this stream to another stream.

error(*container*, *ignoreexception=False*)
Raises error on this stream, so that the receiving end gets an IOError exception.

getEncoderList()
Return the encoder list

prepareRead(*container=None*)
A coroutine method to read the next chunk of data.

read(*container=None*, *size=None*)
Coroutine method to read from the stream and return the data. Raises EOFError when the stream end has been reached; raises IOError if there are other errors.

Parameters

- **container** – A routine container
- **size** – maximum read size, or unlimited if is None

readline(*container=None*, *size=None*)
Coroutine method which reads the next line or until EOF or size exceeds

readonce(*size=None*)
Read from current buffer. If current buffer is empty, returns an empty string. You can use *prepareRead* to read the next chunk of data.

This is not a coroutine method.

write(*data*, *container*, *eof=False*, *ignoreexception=False*, *buffering=True*, *split=True*)
Coroutine method to write data to this stream.

Parameters

- **data** – data to write
- **container** – the routine container
- **eof** – if True, this is the last chunk of this stream. The other end will receive an EOF after reading this chunk.
- **ignoreexception** – even if the stream is closed on the other side, do not raise exception.
- **buffering** – enable buffering. The written data may not be sent if buffering = True; if buffering = False, immediately send any data in the buffer together with this chunk.
- **split** – enable splitting. If this chunk is too large, the stream is allowed to split it into smaller chunks for better balancing.

class vlcp.event.stream.**FileStream**(*fobj*, *encoders=[]*, *isunicode=None*, *size=None*, *readlimit=65536*)
A stream from a file-like object. The file-like object must be in blocking mode

__init__(*fobj*, *encoders=[]*, *isunicode=None*, *size=None*, *readlimit=65536*)
Constructor

close(*scheduler=None*, *allowwrite=False*)
Read side close. To close at the write side, use *eof=True* with *write*.

Parameters

- **scheduler** – the scheduler
- **allowwrite** – do not raise exception on the write side

prepareRead(*container=None*)
A coroutine method to read the next chunk of data.

class vlcp.event.stream.**FileWriter**(*fobj*)
Write to file

__init__(*fobj*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.event.stream.**MemoryStream**(*data*, *encoders=[]*, *isunicode=None*)
A stream with readonly data

__init__(*data*, *encoders=[]*, *isunicode=None*)
Constructor

Parameters

- **data** – all input data
- **encoders** – encoder list
- **isunicode** – Whether this stream outputs unicode. Default to be the same to data. Notice that the encoders may change bytes data to unicode or vice-versa

prepareRead(*container=None*)
A coroutine method to read the next chunk of data.

class vlcp.event.stream.**Stream**(*isunicode=False*, *encoders=[]*, *writebufferlimit=4096*, *splitsize=1048576*)
Streaming data with events

__init__(*isunicode=False*, *encoders=[]*, *writebufferlimit=4096*, *splitsize=1048576*)
Constructor

Parameters

- **isunicode** – True if this stream outputs unicode; False if this stream outputs bytes
- **encoders** – a list of functions *enc(data, is_eof)* which encodes input data.
- **writebufferlimit** – if *buffering=True* on *write*, do not send data until there is more data than this limit
- **splitsize** – if *split=True* on *write*, split chunks larger than this to chunks with this size

close(*scheduler*, *allowwrite=False*)
Read side close. To close at the write side, use *eof=True* with *write*.

Parameters

- **scheduler** – the scheduler
- **allowwrite** – do not raise exception on the write side

error(*container*, *ignoreexception=False*)
Raises error on this stream, so that the receiving end gets an IOError exception.

prepareRead(*container=None*)
A coroutine method to read the next chunk of data.

write(*data*, *container*, *eof=False*, *ignoreexception=False*, *buffering=True*, *split=True*)
Coroutine method to write data to this stream.

Parameters

- **data** – data to write
- **container** – the routine container
- **eof** – if True, this is the last chunk of this stream. The other end will receive an EOF after reading this chunk.
- **ignoreexception** – even if the stream is closed on the other side, do not raise exception.
- **buffering** – enable buffering. The written data may not be sent if buffering = True; if buffering = False, immediately send any data in the buffer together with this chunk.
- **split** – enable splitting. If this chunk is too large, the stream is allowed to split it into smaller chunks for better balancing.

class vlcp.event.stream.**StreamDataEvent**(**args*, ***kwargs*)

canignorenow()
Extra criteria for an event with canignore = False. When this event returns True, the event is safely ignored.

5.3 vlcp.protocol

Note: This document is generated from the source file.

View Source on GitHub

5.3.1 vlcp.protocol.openflow

Note: This document is generated from the source file.

View Source on GitHub

vlcp.protocol.openflow.defs

Note: This document is generated from the source file.

View Source on GitHub

vlcp.protocol.openflow.defs.common

Note: This document is generated from the source file.

View Source on GitHub

/* Copyright (c) 2008, 2011, 2012, 2013, 2014 The Board of Trustees of The Leland Stanford

- Junior University
-
- We are making the OpenFlow specification and associated documentation
- (Software) available for public use and benefit with the expectation
- that others will use, modify and enhance the Software and contribute
- those enhancements back to the community. However, since we would
- like to make the Software available for broadest use, with as few
- restrictions as possible permission is hereby granted, free of
- charge, to any person obtaining a copy of this Software to deal in
- the Software under the copyrights without restriction, including
- without limitation the rights to use, copy, modify, merge, publish,
- distribute, sublicense, and/or sell copies of the Software, and to
- permit persons to whom the Software is furnished to do so, subject to
- the following conditions:
-
- The above copyright notice and this permission notice shall be
- included in all copies or substantial portions of the Software.
-
- THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND,
- EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
- MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
- NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS
- BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
- ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
- CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
- SOFTWARE.
-
- The name and trademarks of copyright holder(s) may NOT be used in
- advertising or publicity pertaining to the Software or any
- derivatives without specific, written prior permission.

*/

/*

- Copyright (c) 2008-2014 Nicira, Inc.
-
- Licensed under the Apache License, Version 2.0 (the “License”);
- you may not use this file except in compliance with the License.
- You may obtain a copy of the License at:
-
- http://www.apache.org/licenses/LICENSE-2.0
-
- Unless required by applicable law or agreed to in writing, software
- distributed under the License is distributed on an “AS IS” BASIS,
- WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
- See the License for the specific language governing permissions and
- limitations under the License.

*/

Created on 2015/7/13

author hubo

vlcp.protocol.openflow.defs.common.**OFPBRC_BAD_SUBTYPE = 4**
/* ofp_error_msg ‘code’ values for OFPET_BAD_ACTION. ‘data’ contains at least

- the first 64 bytes of the failed request. */

vlcp.protocol.openflow.defs.common.**nx_vendor_code = nx_vendor_code**
/* ofp_error_msg ‘code’ values for OFPET_HELLO_FAILED. ‘data’ contains an

- ASCII text string that may give failure details. */

vlcp.protocol.openflow.defs.common.**ofp_error_experimenter_msg = ofp_error_experimenter_msg**
/* ofp_error msg ‘code’ values for NXET_VENDOR. */

vlcp.protocol.openflow.defs.defination

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/7/30

author hubo

vlcp.protocol.openflow.defs.nicira_ext

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/8/3

author hubo

vlcp.protocol.openflow.defs.nicira_ext.**create_extension**(*namespace*, *nicira_header*, *nx_action*, *nx_stats_request*, *nx_stats_reply*, *msg_subtype*, *action_subtype*, *stats_subtype*)

/* This command enables or disables an Open vSwitch extension that allows a

- controller to specify the OpenFlow table to which a flow should be added,
- instead of having the switch decide which table is most appropriate as
- required by OpenFlow 1.0. Because NXM was designed as an extension to
- OpenFlow 1.0, the extension applies equally to ofp10_flow_mod and
- nx_flow_mod. By default, the extension is disabled.
-
- When this feature is enabled, Open vSwitch treats struct ofp10_flow_mod's
- and struct nx_flow_mod's 16-bit 'command' member as two separate fields.
- The upper 8 bits are used as the table ID, the lower 8 bits specify the
- command as usual. A table ID of 0xff is treated like a wildcarded table ID.
-
- The specific treatment of the table ID depends on the type of flow mod:
-
- – OFPFC_ADD: Given a specific table ID, the flow is always placed in that
- table. If an identical flow already exists in that table only, then it
- is replaced. If the flow cannot be placed in the specified table,
- either because the table is full or because the table cannot support
- flows of the given type, the switch replies with an OFPFMFC_TABLE_FULL
- error. (A controller can distinguish these cases by comparing the
- current and maximum number of entries reported in ofp_table_stats.)
-
- If the table ID is wildcarded, the switch picks an appropriate table
- itself. If an identical flow already exist in the selected flow table,
- then it is replaced. The choice of table might depend on the flows

- that are already in the switch; for example, if one table fills up then
- the switch might fall back to another one.
-
- – OFPFC_MODIFY, OFPFC_DELETE: Given a specific table ID, only flows
- within that table are matched and modified or deleted. If the table ID
- is wildcarded, flows within any table may be matched and modified or
- deleted.
-
- – OFPFC_MODIFY_STRICT, OFPFC_DELETE_STRICT: Given a specific table ID,
- only a flow within that table may be matched and modified or deleted.
- If the table ID is wildcarded and exactly one flow within any table
- matches, then it is modified or deleted; if flows in more than one
- table match, then none is modified or deleted.

*/

vlcp.protocol.openflow.defs.nicira_ext.**nx_flow_format = nx_flow_format**
/* 'flags' bits in struct nx_flow_monitor_request. */

vlcp.protocol.openflow.defs.nicira_ext.**nx_flow_monitor_flags = nx_flow_monitor_flags**
/* 'event' values in struct nx_flow_update_header. */

vlcp.protocol.openflow.defs.nicira_ext.**nx_role = nx_role**
/* Flexible flow specifications (aka NXM = Nicira Extended Match).

-
- OpenFlow 1.0 has "struct ofp10_match" for specifying flow matches. This
- structure is fixed-length and hence difficult to extend. This section
- describes a more flexible, variable-length flow match, called "nx_match" for
- short, that is also supported by Open vSwitch. This section also defines a
- replacement for each OpenFlow message that includes struct ofp10_match.
-
-
- Format
-
-
- An nx_match is a sequence of zero or more "nxm_entry"s, which are
- type-length-value (TLV) entries, each 5 to 259 (inclusive) bytes long.
- "nxm_entry"s are not aligned on or padded to any multibyte boundary. The
- first 4 bytes of an nxm_entry are its "header", followed by the entry's
- "body".
-

- An nxm_entry's header is interpreted as a 32-bit word in network byte order:
-
- |<————————— nxm_type —————————>|
- |
- |31 16 15 9| 8 7 0
-
- nxm_vendor | nxm_field |hm| nxm_length |
-
-
- The most-significant 23 bits of the header are collectively "nxm_type".
- Bits 16...31 are "nxm_vendor", one of the NXM_VENDOR_* values below. Bits
- 9...15 are "nxm_field", which is a vendor-specific value. nxm_type normally
- designates a protocol header, such as the Ethernet type, but it can also
- refer to packet metadata, such as the switch port on which a packet arrived.
-
- Bit 8 is "nxm_hasmask" (labeled "hm" above for space reasons). The meaning
- of this bit is explained later.
-
- The least-significant 8 bits are "nxm_length", a positive integer. The
- length of the nxm_entry, including the header, is exactly 4 + nxm_length
- bytes.
-
- For a given nxm_vendor, nxm_field, and nxm_hasmask value, nxm_length is a
- constant. It is included only to allow software to minimally parse
- "nxm_entry"s of unknown types. (Similarly, for a given nxm_vendor,
- nxm_field, and nxm_length, nxm_hasmask is a constant.)
-
-
- Semantics
-
-
- A zero-length nx_match (one with no "nxm_entry"s) matches every packet.
-
- An nxm_entry places a constraint on the packets matched by the nx_match:
-

- – If nxm_hasmask is 0, the nxm_entry's body contains a value for the
- field, called "nxm_value". The nx_match matches only packets in which
- the field equals nxm_value.
-
- – If nxm_hasmask is 1, then the nxm_entry's body contains a value for the
- field (nxm_value), followed by a bitmask of the same length as the
- value, called "nxm_mask". For each 1-bit in position J in nxm_mask, the
- nx_match matches only packets for which bit J in the given field's value
- matches bit J in nxm_value. A 0-bit in nxm_mask causes the
- corresponding bit in nxm_value is ignored (it should be 0; Open vSwitch
- may enforce this someday), as is the corresponding bit in the field's
- value. (The sense of the nxm_mask bits is the opposite of that used by
- the "wildcards" member of struct ofp10_match.)
-
- When nxm_hasmask is 1, nxm_length is always even.
-
- An all-zero-bits nxm_mask is equivalent to omitting the nxm_entry
- entirely. An all-one-bits nxm_mask is equivalent to specifying 0 for
- nxm_hasmask.
-
- When there are multiple "nxm_entry"s, all of the constraints must be met.
-
-
- Mask Restrictions
-
-
- Masks may be restricted:
-
- – Some nxm_types may not support masked wildcards, that is, nxm_hasmask
- must always be 0 when these fields are specified. For example, the
- field that identifies the port on which a packet was received may not be
- masked.
-
- – Some nxm_types that do support masked wildcards may only support certain
- nxm_mask patterns. For example, fields that have IPv4 address values
- may be restricted to CIDR masks.

•
• These restrictions should be noted in specifications for individual fields.
• A switch may accept an nxm_hasmask or nxm_mask value that the specification
• disallows, if the switch correctly implements support for that nxm_hasmask
• or nxm_mask value. A switch must reject an attempt to set up a flow that
• contains a nxm_hasmask or nxm_mask value that it does not support.
•
•
• Prerequisite Restrictions
•
•
• The presence of an nxm_entry with a given nxm_type may be restricted based
• on the presence of or values of other “nxm_entry”s. For example:
•
• – An nxm_entry for nxm_type=NXM_OF_IP_TOS is allowed only if it is
• preceded by another entry with nxm_type=NXM_OF_ETH_TYPE, nxm_hasmask=0,
• and nxm_value=0x0800. That is, matching on the IP source address is
• allowed only if the Ethernet type is explicitly set to IP.
•
• – An nxm_entry for nxm_type=NXM_OF_TCP_SRC is allowed only if it is
• preceded by an entry with nxm_type=NXM_OF_ETH_TYPE, nxm_hasmask=0, and
• nxm_value either 0x0800 or 0x86dd, and another with
• nxm_type=NXM_OF_IP_PROTO, nxm_hasmask=0, nxm_value=6, in that order.
• That is, matching on the TCP source port is allowed only if the Ethernet
• type is IP or IPv6 and the IP protocol is TCP.
•
• These restrictions should be noted in specifications for individual fields.
• A switch may implement relaxed versions of these restrictions. A switch
• must reject an attempt to set up a flow that violates its restrictions.
•
•
• Ordering Restrictions
•
•
• An nxm_entry that has prerequisite restrictions must appear after the
• “nxm_entry”s for its prerequisites. Ordering of “nxm_entry”s within an

- nx_match is not otherwise constrained.
-
- Any given nxm_type may appear in an nx_match at most once.
-
-
- nxm_entry Examples
-
-
- These examples show the format of a single nxm_entry with particular
- nxm_hasmask and nxm_length values. The diagrams are labeled with field
- numbers and byte indexes.
-
-
- 8-bit nxm_value, nxm_hasmask=1, nxm_length=2:
-
- 0 3 4 5
  -
- header | v | m |
  -
-
-
- 16-bit nxm_value, nxm_hasmask=0, nxm_length=2:
-
- 0 3 4 5
  -
- header | value|
  -
-
-
- 32-bit nxm_value, nxm_hasmask=0, nxm_length=4:
-
- 0 3 4 7
  -
- header | nxm_value |
  -

-
-
- 48-bit nxm_value, nxm_hasmask=0, nxm_length=6:
-
- 0 3 4 9
-
- header | nxm_value |
-
-
-
- 48-bit nxm_value, nxm_hasmask=1, nxm_length=12:
-
- 0 3 4 9 10 15
-
- header | nxm_value | nxm_mask |
-
-
-
- Error Reporting
-
-
- A switch should report an error in an nx_match using error type
- OFPET_BAD_REQUEST and one of the NXBRC_NXM_* codes. Ideally the switch
- should report a specific error code, if one is assigned for the particular
- problem, but NXBRC_NXM_INVALID is also available to report a generic
- nx_match error.

*/

`vlcp.protocol.openflow.defs.nicira_ext.`**`nxt_subtype = nxt_subtype`**
/* Fields to use when hashing flows. */

vlcp.protocol.openflow.defs.openflow10

Note: This document is generated from the source file.

View Source on GitHub

/*

- Copyright (c) 2008, 2009, 2010, 2011, 2012, 2013 Nicira, Inc.
-
- Licensed under the Apache License, Version 2.0 (the “License”);
- you may not use this file except in compliance with the License.
- You may obtain a copy of the License at:
-
- http://www.apache.org/licenses/LICENSE-2.0
-
- Unless required by applicable law or agreed to in writing, software
- distributed under the License is distributed on an “AS IS” BASIS,
- WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
- See the License for the specific language governing permissions and
- limitations under the License.

*/

/* OpenFlow: protocol between controller and datapath. */ Created on 2015/7/13

author hubo

vlcp.protocol.openflow.defs.openflow10.**OFP_VLAN_NONE = 65535**
/* Fields to match against flows */

vlcp.protocol.openflow.defs.openflow10.**nicira_header = nicira_header**
/* Header for Nicira-defined actions. */

vlcp.protocol.openflow.defs.openflow10.**ofp_action_dl_addr = ofp_action_dl_addr**
/* Action structure for OFPAT10_SET_NW_SRC/DST and OFPAT11_SET_NW_SRC/DST. */

vlcp.protocol.openflow.defs.openflow10.**ofp_action_enqueue = ofp_action_enqueue**
/* Send packet (controller -> datapath). */

vlcp.protocol.openflow.defs.openflow10.**ofp_action_nw_addr = ofp_action_nw_addr**
/* Action structure for OFPAT10_SET_NW_TOS and OFPAT11_SET_NW_TOS. */

vlcp.protocol.openflow.defs.openflow10.**ofp_action_nw_tos = ofp_action_nw_tos**
/* Action structure for OFPAT10_SET_TP_SRC/DST and OFPAT11_SET_TP_SRC/DST. */

vlcp.protocol.openflow.defs.openflow10.**ofp_action_output = ofp_action_output**
/* Action structure for OFPAT10_SET_VLAN_VID and OFPAT11_SET_VLAN_VID. */

vlcp.protocol.openflow.defs.openflow10.**ofp_action_tp_port = ofp_action_tp_port**
/* OpenFlow 1.0 specific features of physical ports available in a datapath. */

vlcp.protocol.openflow.defs.openflow10.**ofp_action_vendor = ofp_action_vendor**
/* Action structure for OFPAT10_OUTPUT, which sends packets out ‘port’.

- When the ‘port’ is the OFPP_CONTROLLER, ‘max_len’ indicates the max
- number of bytes to send. A ‘max_len’ of zero means no bytes of the
- packet should be sent. */

vlcp.protocol.openflow.defs.openflow10.**ofp_action_vlan_pcp = ofp_action_vlan_pcp**
/* Action structure for OFPAT10_SET_DL_SRC/DST and OFPAT11_SET_DL_SRC/DST. */

vlcp.protocol.openflow.defs.openflow10.**ofp_action_vlan_vid = ofp_action_vlan_vid**
/* Action structure for OFPAT10_SET_VLAN_PCP and OFPAT11_SET_VLAN_PCP. */

vlcp.protocol.openflow.defs.openflow10.**ofp_aggregate_stats_reply = ofp_aggregate_stats_repl**
/* Body of reply to OFPST_TABLE request. */

vlcp.protocol.openflow.defs.openflow10.**ofp_aggregate_stats_request = ofp_aggregate_stats_re**
/* Body of reply to OFPST_AGGREGATE request. */

vlcp.protocol.openflow.defs.openflow10.**ofp_capabilities = ofp_capabilities**
/* OpenFlow 1.0 specific current state of the physical port. These are not

- configurable from the controller.

*/
/* The OFPPS10_STP_* bits have no effect on switch operation. The

- controller must adjust OFPPC_NO_RECV, OFPPC_NO_FWD, and
- OFPPC_NO_PACKET_IN appropriately to fully implement an 802.1D spanning
- tree. */

vlcp.protocol.openflow.defs.openflow10.**ofp_desc_stats_reply = ofp_desc_stats_reply**
/* Stats request of type OFPST_AGGREGATE or OFPST_FLOW. */

vlcp.protocol.openflow.defs.openflow10.**ofp_error_type = ofp_error_type**
/* ofp_error_msg 'code' values for OFPET_FLOW_MOD_FAILED. 'data' contains * at least the first 64 bytes of the failed request. */

vlcp.protocol.openflow.defs.openflow10.**ofp_flow_mod = ofp_flow_mod**
/* Flow removed (datapath -> controller). */

vlcp.protocol.openflow.defs.openflow10.**ofp_flow_mod_failed_code = ofp_flow_mod_failed_code**
/* ofp_error_msg 'code' values for OFPET_PORT_MOD_FAILED. 'data' contains * at least the first 64 bytes of the failed request. */

vlcp.protocol.openflow.defs.openflow10.**ofp_flow_mod_flags = ofp_flow_mod_flags**
/* Flow setup and teardown (controller -> datapath). */

vlcp.protocol.openflow.defs.openflow10.**ofp_flow_stats_request = ofp_flow_stats_request**
/* Body of reply to OFPST_FLOW request. */

vlcp.protocol.openflow.defs.openflow10.**ofp_packet_in = ofp_packet_in**
/* OFPAT10_ENQUEUE action struct: send packets to given queue on port. */

vlcp.protocol.openflow.defs.openflow10.**ofp_packet_out = ofp_packet_out**
/* Flow wildcards. */

vlcp.protocol.openflow.defs.openflow10.**ofp_packet_queue = ofp_packet_queue**
/* Query for port queue configuration. */

vlcp.protocol.openflow.defs.openflow10.**ofp_port_features = ofp_port_features**
/* Description of a physical port */

vlcp.protocol.openflow.defs.openflow10.**ofp_port_mod_failed_code = ofp_port_mod_failed_code**
/* ofp_error msg 'code' values for OFPET_QUEUE_OP_FAILED. 'data' contains * at least the first 64 bytes of the failed request */

vlcp.protocol.openflow.defs.openflow10.**ofp_port_stats_reply = ofp_port_stats_reply**
/* All ones is used to indicate all queues in a port (for stats retrieval). */

vlcp.protocol.openflow.defs.openflow10.**ofp_port_stats_request = ofp_port_stats_request**
/* Body of reply to OFPST_PORT request. If a counter is unsupported, set

- the field to all ones. */

vlcp.protocol.openflow.defs.openflow10.**ofp_port_status = ofp_port_status**
/* Statistics request or reply message. */

vlcp.protocol.openflow.defs.openflow10.**ofp_queue = ofp_queue**
/* Body for stats request of type OFPST_QUEUE. */

vlcp.protocol.openflow.defs.openflow10.**ofp_queue_get_config_reply = ofp_queue_get_config_re**
/* Packet received on port (datapath -> controller). */

vlcp.protocol.openflow.defs.openflow10.**ofp_queue_get_config_request = ofp_queue_get_config_**
/* Queue configuration for a given port. */

vlcp.protocol.openflow.defs.openflow10.**ofp_queue_stats_reply = ofp_queue_stats_reply**
/* Vendor extension stats message. */

vlcp.protocol.openflow.defs.openflow10.**ofp_queue_stats_request = ofp_queue_stats_request**
/* Body for stats reply of type OFPST_QUEUE consists of an array of this

- structure type. */

vlcp.protocol.openflow.defs.openflow10.**ofp_switch_config = ofp_switch_config**
/* OpenFlow 1.0 specific capabilities supported by the datapath (struct

- ofp_switch_features, member capabilities). */

vlcp.protocol.openflow.defs.openflow10.**ofp_switch_features = ofp_switch_features**
/* Modify behavior of the physical port */

vlcp.protocol.openflow.defs.openflow10.**ofp_table = ofp_table**
/* Body for ofp_stats_request of type OFPST_AGGREGATE. */

vlcp.protocol.openflow.defs.openflow10.**ofp_table_stats_reply = ofp_table_stats_reply**
/* Stats request of type OFPST_PORT. */

vlcp.protocol.openflow.defs.openflow13

Note: This document is generated from the source file.

View Source on GitHub

/* Copyright (c) 2008 The Board of Trustees of The Leland Stanford * Junior University * Copyright (c) 2011, 2012 Open Networking Foundation * * We are making the OpenFlow specification and associated documentation * (Software) available for public use and benefit with the expectation * that others will use, modify and enhance the Software and contribute * those enhancements back to the community. However, since we would * like to make the Software available for broadest use, with as few * restrictions as possible permission is hereby granted, free of * charge, to any person obtaining a copy of this Software to deal in * the Software under the copyrights without restriction, including * without limitation the rights to use, copy, modify, merge, publish, * distribute, sublicense, and/or sell copies of the Software, and to * permit persons to whom the Software is furnished to do so, subject to * the following conditions: * * The above copyright notice and this permission notice shall be * included in all copies or substantial portions of the Software. * * THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF * MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND * NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS * BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN * ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN * CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE * SOFTWARE. * * The

name and trademarks of copyright holder(s) may NOT be used in * advertising or publicity pertaining to the Software or any * derivatives without specific, written prior permission. */ Created on 2015/7/14

author hubo

vlcp.protocol.openflow.defs.openflow13.**OFPM_MAX = 4294901760**
/* Meter band types */

vlcp.protocol.openflow.defs.openflow13.**OFPQ_MAX_RATE_UNCFG = 65535**
/* Common description for a queue. */

vlcp.protocol.openflow.defs.openflow13.**OFPQ_MIN_RATE_UNCFG = 65535**
/* Max rate > 1000 means not configured. */

vlcp.protocol.openflow.defs.openflow13.**OFPXMT_OFB_ALL = 1099511627775**
/* The VLAN id is 12-bits, so we can use the entire 16 bits to indicate

- special conditions.

*/

vlcp.protocol.openflow.defs.openflow13.**OFP_FLOW_PERMANENT = 0**
/* By default, choose a priority in the middle. */

vlcp.protocol.openflow.defs.openflow13.**SERIAL_NUM_LEN = 32**
/* Body of reply to OFPMP_DESC request. Each entry is a NULL-terminated

- ASCII string. */

vlcp.protocol.openflow.defs.openflow13.**nicira_header = nicira_header**
/* Header for Nicira-defined actions. */

vlcp.protocol.openflow.defs.openflow13.**ofp_action_experimenter = ofp_action_experimenter**
/* ## ———————- ## // ## OpenFlow Instructions. ## // ## ———————- ## */

vlcp.protocol.openflow.defs.openflow13.**ofp_action_experimenter_desc = ofp_action_experiment**
/* Actions property */

vlcp.protocol.openflow.defs.openflow13.**ofp_action_group = ofp_action_group**
/* Action structure for OFPAT_SET_NW_TTL. */

vlcp.protocol.openflow.defs.openflow13.**ofp_action_mpls_ttl = ofp_action_mpls_ttl**
/* Action structure for OFPAT_PUSH_VLAN/MPLS/PBB. */

vlcp.protocol.openflow.defs.openflow13.**ofp_action_nw_ttl = ofp_action_nw_ttl**
/* Action structure for OFPAT_SET_FIELD. */

vlcp.protocol.openflow.defs.openflow13.**ofp_action_output = ofp_action_output**
/* Action structure for OFPAT_SET_MPLS_TTL. */

vlcp.protocol.openflow.defs.openflow13.**ofp_action_pop_mpls = ofp_action_pop_mpls**
/* Action structure for OFPAT_GROUP. */

vlcp.protocol.openflow.defs.openflow13.**ofp_action_push = ofp_action_push**
/* Action structure for OFPAT_POP_MPLS. */

vlcp.protocol.openflow.defs.openflow13.**ofp_action_set_field = ofp_action_set_field**
/* Action header for OFPAT_EXPERIMENTER.

- The rest of the body is experimenter-defined. */

vlcp.protocol.openflow.defs.openflow13.**ofp_action_type = ofp_action_type**
/* Action header that is common to all actions. The length includes the

- header and any padding used to make the action 64-bit aligned.

- NB: The length of an action *must* always be a multiple of eight. */

vlcp.protocol.openflow.defs.openflow13.**ofp_action_type_bitwise = ofp_action_type_bitwise**
/* Body of reply to OFPMP_GROUP_FEATURES request. Group features. */

vlcp.protocol.openflow.defs.openflow13.**ofp_aggregate_stats_reply = ofp_aggregate_stats_repl**
/* Table Feature property types.

- Low order bit cleared indicates a property for a regular Flow Entry.
- Low order bit set indicates a property for the Table-Miss Flow Entry.

*/

vlcp.protocol.openflow.defs.openflow13.**ofp_aggregate_stats_request = ofp_aggregate_stats_re**
/* Body of reply to OFPMP_AGGREGATE request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_bad_instruction_code = ofp_bad_instruction_code**
/* ofp_error_msg 'code' values for OFPET_BAD_MATCH. 'data' contains at least

- the first 64 bytes of the failed request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_bad_match_code = ofp_bad_match_code**
/* ofp_error_msg 'code' values for OFPET_FLOW_MOD_FAILED. 'data' contains

- at least the first 64 bytes of the failed request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_bucket = ofp_bucket**
/* Group types. Values in the range [128, 255] are reserved for experimental

- use. */

vlcp.protocol.openflow.defs.openflow13.**ofp_bucket_counter = ofp_bucket_counter**
/* Body of reply to OFPMP_GROUP request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_buffer_id = ofp_buffer_id**
/* Flow setup and teardown (controller -> datapath). */

vlcp.protocol.openflow.defs.openflow13.**ofp_capabilities = ofp_capabilities**
/* Current state of the physical port. These are not configurable from

- the controller.

*/

vlcp.protocol.openflow.defs.openflow13.**ofp_controller_max_len = ofp_controller_max_len**
/* Action structure for OFPAT_OUTPUT, which sends packets out 'port'.

- When the 'port' is the OFPP_CONTROLLER, 'max_len' indicates the max
- number of bytes to send. A 'max_len' of zero means no bytes of the
- packet should be sent. A 'max_len' of OFPCML_NO_BUFFER means that
- the packet is not buffered and the complete packet is to be sent to
- the controller. */

vlcp.protocol.openflow.defs.openflow13.**ofp_controller_role = ofp_controller_role**
/* Role request and reply message. */

vlcp.protocol.openflow.defs.openflow13.**ofp_desc_reply = ofp_desc_reply**
/* Body for ofp_multipart_request of type OFPMP_FLOW. */

vlcp.protocol.openflow.defs.openflow13.**ofp_experimenter = ofp_experimenter**
/* All ones is used to indicate all queues in a port (for stats retrieval). */

vlcp.protocol.openflow.defs.openflow13.**ofp_experimenter_multipart_reply = ofp_experimenter_**
/* Experimenter extension. */

vlcp.protocol.openflow.defs.openflow13.**ofp_flow_mod = ofp_flow_mod**
/* Group commands */

vlcp.protocol.openflow.defs.openflow13.**ofp_flow_mod_failed_code = ofp_flow_mod_failed_code**
/* ofp_error_msg 'code' values for OFPET_GROUP_MOD_FAILED. 'data' contains

- at least the first 64 bytes of the failed request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_flow_mod_flags = ofp_flow_mod_flags**
/* Special buffer-id to indicate 'no buffer' */

vlcp.protocol.openflow.defs.openflow13.**ofp_flow_removed = ofp_flow_removed**
/* Meter numbering. Flow meters can use any number up to OFPM_MAX. */

vlcp.protocol.openflow.defs.openflow13.**ofp_flow_removed_reason_bitwise = ofp_flow_removed_r**
/* Asynchronous message configuration. */

vlcp.protocol.openflow.defs.openflow13.**ofp_flow_stats_reply = ofp_flow_stats_reply**
/* Body for ofp_multipart_request of type OFPMP_AGGREGATE. */

vlcp.protocol.openflow.defs.openflow13.**ofp_flow_stats_request = ofp_flow_stats_request**
/* Body of reply to OFPMP_FLOW request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_group_desc_reply = ofp_group_desc_reply**
/* Backward compatibility with 1.3.1 - avoid breaking the API. */

vlcp.protocol.openflow.defs.openflow13.**ofp_group_desc_stats = ofp_group_desc**
/* Group configuration flags */

vlcp.protocol.openflow.defs.openflow13.**ofp_group_features_reply = ofp_group_features_reply**
/* Body of OFPMP_METER and OFPMP_METER_CONFIG requests. */

vlcp.protocol.openflow.defs.openflow13.**ofp_group_mod = ofp_group_mod**
/* Send packet (controller -> datapath). */

vlcp.protocol.openflow.defs.openflow13.**ofp_group_mod_command = ofp_group_mod_command**
/* Bucket for use in groups. */

vlcp.protocol.openflow.defs.openflow13.**ofp_group_mod_failed_code = ofp_group_mod_failed_cod**
/* ofp_error_msg 'code' values for OFPET_PORT_MOD_FAILED. 'data' contains

- at least the first 64 bytes of the failed request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_group_stats_reply = ofp_group_stats_reply**
/* Body of reply to OFPMP_GROUP_DESC request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_group_stats_request = ofp_group_stats_request**
/* Used in group stats replies. */

vlcp.protocol.openflow.defs.openflow13.**ofp_group_type = ofp_group_type**
/* Group setup and teardown (controller -> datapath). */

vlcp.protocol.openflow.defs.openflow13.**ofp_instruction = ofp_instruction**
/* Instruction structure for OFPIT_GOTO_TABLE */

vlcp.protocol.openflow.defs.openflow13.**ofp_instruction_actions = ofp_instruction_actions**
/* Instruction structure for OFPIT_METER */

vlcp.protocol.openflow.defs.openflow13.**ofp_instruction_experimenter = ofp_instruction_exper**
/* Value used in "idle_timeout" and "hard_timeout" to indicate that the entry

- is permanent. */

vlcp.protocol.openflow.defs.openflow13.**ofp_instruction_experimenter_feature = ofp_instructi**
/* Instructions property */

vlcp.protocol.openflow.defs.openflow13.**ofp_instruction_goto_table = ofp_instruction_goto_ta**
/* Instruction structure for OFPIT_WRITE_METADATA */

vlcp.protocol.openflow.defs.openflow13.**ofp_instruction_meter = ofp_instruction_meter**
/* Instruction structure for experimental instructions */

vlcp.protocol.openflow.defs.openflow13.**ofp_instruction_type = ofp_instruction_type**
/* Instruction header that is common to all instructions. The length includes

- the header and any padding used to make the instruction 64-bit aligned.
- NB: The length of an instruction *must* always be a multiple of eight. */

vlcp.protocol.openflow.defs.openflow13.**ofp_instruction_write_metadata = ofp_instruction_wri**
/* Instruction structure for OFPIT_WRITE/APPLY/CLEAR_ACTIONS */

vlcp.protocol.openflow.defs.openflow13.**ofp_match = ofp_match**
/* Components of a OXM TLV header.

- Those macros are not valid for the experimenter class, macros for the
- experimenter class will depend on the experimenter header used. */

vlcp.protocol.openflow.defs.openflow13.**ofp_meter_band = ofp_meter_band**
/* OFPMBT_DROP band - drop packets */

vlcp.protocol.openflow.defs.openflow13.**ofp_meter_band_drop = ofp_meter_band_drop**
/* OFPMBT_DSCP_REMARK band - Remark DSCP in the IP header */

vlcp.protocol.openflow.defs.openflow13.**ofp_meter_band_dscp_remark = ofp_meter_band_dscp_rem**
/* OFPMBT_EXPERIMENTER band - Experimenter type.

- The rest of the band is experimenter-defined. */

vlcp.protocol.openflow.defs.openflow13.**ofp_meter_band_experimenter = ofp_meter_band_experim**
/* Meter commands */

vlcp.protocol.openflow.defs.openflow13.**ofp_meter_band_stats = ofp_meter_band_stats**
/* Body of reply to OFPMP_METER request. Meter statistics. */

vlcp.protocol.openflow.defs.openflow13.**ofp_meter_band_type = ofp_meter_band_type**
/* Common header for all meter bands */

vlcp.protocol.openflow.defs.openflow13.**ofp_meter_band_type_bitwise = ofp_meter_band_type_bi**
/* Body of reply to OFPMP_METER_FEATURES request. Meter features. */

vlcp.protocol.openflow.defs.openflow13.**ofp_meter_features_reply = ofp_meter_features_reply**
/* Body for ofp_multipart_request/reply of type OFPMP_EXPERIMENTER. */

vlcp.protocol.openflow.defs.openflow13.**ofp_meter_flags = ofp_meter_flags**
/* Meter configuration. OFPT_METER_MOD. */

vlcp.protocol.openflow.defs.openflow13.**ofp_meter_mod = ofp_meter_mod**
/* ofp_error_msg 'code' values for OFPET_BAD_INSTRUCTION. 'data' contains at least

- the first 64 bytes of the failed request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_meter_mod_command = ofp_meter_mod_command**
/* Meter configuration flags */

vlcp.protocol.openflow.defs.openflow13.**ofp_meter_mod_failed_code = ofp_meter_mod_failed_cod**
/* ofp_error_msg 'code' values for OFPET_TABLE_FEATURES_FAILED. 'data' contains

- at least the first 64 bytes of the failed request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_meter_multipart_request = ofp_meter_multipart_re**
/* Statistics for each meter band */

vlcp.protocol.openflow.defs.openflow13.**ofp_meter_stats_reply = ofp_meter_stats_reply**
/* Body of reply to OFPMP_METER_CONFIG request. Meter configuration. */

vlcp.protocol.openflow.defs.openflow13.**ofp_msg = ofp_msg**
/* Switch configuration. */

vlcp.protocol.openflow.defs.openflow13.**ofp_multipart_type = ofp_multipart_type**
/* Backward compatibility with 1.3.1 - avoid breaking the API. */

vlcp.protocol.openflow.defs.openflow13.**ofp_oxm = ofp_oxm**
/* Header for OXM experimenter match fields.

- The experimenter class should not use OXM_HEADER() macros for defining
- fields due to this extra header. */

vlcp.protocol.openflow.defs.openflow13.**ofp_oxm_class = ofp_oxm_class**
/* OXM Flow match field types for OpenFlow basic class. */

vlcp.protocol.openflow.defs.openflow13.**ofp_oxm_header = ofp_oxm_header**
/* Bit definitions for IPv6 Extension Header pseudo-field. */

vlcp.protocol.openflow.defs.openflow13.**ofp_oxm_mask_ipv6 = ofp_oxm_mask_ipv6**
/* ## ———————— ## // ## OpenFlow Actions. ## // ## ———————— ## */

vlcp.protocol.openflow.defs.openflow13.**ofp_packet_out = ofp_packet_out**
/* Packet received on port (datapath -> controller). */

vlcp.protocol.openflow.defs.openflow13.**ofp_packet_queue = ofp_packet_queue**
/* Query for port queue configuration. */

vlcp.protocol.openflow.defs.openflow13.**ofp_port = ofp_port**
/* Switch features. */

vlcp.protocol.openflow.defs.openflow13.**ofp_port_desc_reply = ofp_port_desc_reply**
/* Body of OFPMP_GROUP request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_port_features = ofp_port_features**
/* Description of a port */

vlcp.protocol.openflow.defs.openflow13.**ofp_port_mod = ofp_port_mod**
/* ## ———————————— ## // ## OpenFlow Extensible Match. ## // ## ———————————— ## */
/* The match type indicates the match structure (set of fields that compose the

- match) in use. The match type is placed in the type field at the beginning
- of all match structures. The “OpenFlow Extensible Match” type corresponds
- to OXM TLV format described below and must be supported by all OpenFlow
- switches. Extensions that define other match types may be published on the
- ONF wiki. Support for extensions is optional.

*/

/* Fields to match against flows */

vlcp.protocol.openflow.defs.openflow13.**ofp_port_mod_failed_code = ofp_port_mod_failed_code**
/* ofp_error_msg ‘code’ values for OFPET_TABLE_MOD_FAILED. ‘data’ contains

- at least the first 64 bytes of the failed request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_port_state = ofp_port_state**
/* Features of ports available in a datapath. */

vlcp.protocol.openflow.defs.openflow13.**ofp_port_stats_request = ofp_port_stats_request**
/* Body of reply to OFPMP_PORT request. If a counter is unsupported, set

- the field to all ones. */

vlcp.protocol.openflow.defs.openflow13.**ofp_port_status = ofp_port_status**
/* Modify behavior of the physical port */

vlcp.protocol.openflow.defs.openflow13.**ofp_queue = ofp_queue**
/* Min rate > 1000 means not configured. */

vlcp.protocol.openflow.defs.openflow13.**ofp_queue_get_config_reply = ofp_queue_get_config_re**
/* OFPAT_SET_QUEUE action struct: send packets to given queue on port. */

vlcp.protocol.openflow.defs.openflow13.**ofp_queue_get_config_request = ofp_queue_get_config_**
/* Queue configuration for a given port. */

vlcp.protocol.openflow.defs.openflow13.**ofp_queue_op_failed_code = ofp_queue_op_failed_code**
/* ofp_error_msg 'code' values for OFPET_SWITCH_CONFIG_FAILED. 'data' contains

- at least the first 64 bytes of the failed request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_queue_prop = ofp_queue_prop**
/* Min-Rate queue property description. */

vlcp.protocol.openflow.defs.openflow13.**ofp_queue_prop_experimenter = ofp_queue_prop_experi**
/* Full description for a queue. */

vlcp.protocol.openflow.defs.openflow13.**ofp_queue_prop_max_rate = ofp_queue_prop_max_rate**
/* Experimenter queue property description. */

vlcp.protocol.openflow.defs.openflow13.**ofp_queue_prop_min_rate = ofp_queue_prop_min_rate**
/* Max-Rate queue property description. */

vlcp.protocol.openflow.defs.openflow13.**ofp_queue_stats_reply = ofp_queue_stats_reply**
/* Configures the "role" of the sending controller. The default role is:

-
- – Equal (OFPCR_ROLE_EQUAL), which allows the controller access to all
- OpenFlow features. All controllers have equal responsibility.
-
- The other possible roles are a related pair:
-
- – Master (OFPCR_ROLE_MASTER) is equivalent to Equal, except that there
- may be at most one Master controller at a time: when a controller
- configures itself as Master, any existing Master is demoted to the
- Slave role.
-
- – Slave (OFPCR_ROLE_SLAVE) allows the controller read-only access to
- OpenFlow features. In particular attempts to modify the flow table
- will be rejected with an OFPBRC_EPERM error.

-
- Slave controllers do not receive OFPT_PACKET_IN or OFPT_FLOW_REMOVED
- messages, but they do receive OFPT_PORT_STATUS messages.

*/

vlcp.protocol.openflow.defs.openflow13.**ofp_role_request_failed_code = ofp_role_request_fai**
/* ofp_error_msg 'code' values for OFPET_METER_MOD_FAILED. 'data' contains

- at least the first 64 bytes of the failed request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_switch_config = ofp_switch_config**
/* Configure/Modify behavior of a flow table */

vlcp.protocol.openflow.defs.openflow13.**ofp_switch_config_failed_code = ofp_switch_config_fa**
/* ofp_error_msg 'code' values for OFPET_ROLE_REQUEST_FAILED. 'data' contains

- at least the first 64 bytes of the failed request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_switch_features = ofp_switch_features**
/* A physical port has changed in the datapath */

vlcp.protocol.openflow.defs.openflow13.**ofp_table_feature_prop_actions = ofp_table_feature_p**
/* Match, Wildcard or Set-Field property */

vlcp.protocol.openflow.defs.openflow13.**ofp_table_feature_prop_experimenter = ofp_table_feat**
/* Body for ofp_multipart_request of type OFPMP_TABLE_FEATURES./

- Body of reply to OFPMP_TABLE_FEATURES request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_table_feature_prop_instructions = ofp_table_feat**
/* Next Tables property */

vlcp.protocol.openflow.defs.openflow13.**ofp_table_feature_prop_oxm = ofp_table_feature_prop_**
/* Experimenter table feature property */

vlcp.protocol.openflow.defs.openflow13.**ofp_table_feature_prop_type = ofp_table_feature_prop**
/* Common header for all Table Feature Properties */

vlcp.protocol.openflow.defs.openflow13.**ofp_table_features_reply = ofp_table_features_reply**
/* Body of reply to OFPMP_TABLE request. */

vlcp.protocol.openflow.defs.openflow13.**ofp_table_mod = ofp_table_mod**
/* Capabilities supported by the datapath. */

vlcp.protocol.openflow.defs.openflow13.**ofp_table_mod_failed_code = ofp_table_mod_failed_co**
/* ofp_error msg 'code' values for OFPET_QUEUE_OP_FAILED. 'data' contains

- at least the first 64 bytes of the failed request */

vlcp.protocol.openflow.defs.openflow13.**ofp_table_stats_reply = ofp_table_stats_reply**
/* Body for ofp_multipart_request of type OFPMP_PORT. */

vlcp.protocol.openflow.defs.openflow13.**ofp_vlan_id = ofp_vlan_id**
/* Define for compatibility */

vlcp.protocol.openflow.openflow

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/7/8

author hubo

class vlcp.protocol.openflow.openflow.**Openflow**(*allowedVersions=None*)
Openflow control protocol

__init__(*allowedVersions=None*)
Constructor

Parameters **allowedVersions** – if specified, should be a tuple of allowed OpenFlow versions.

batch(*requests*, *connection*, *container*, *raiseonerror=True*)
Send multiple requests, return when all the requests are done. Requests can have no responses. The attributes are set even if an OpenflowErrorResultException is raised.

Returns (openflow_reply, openflow_replydict) in which *openflow_reply* is the list of messages in receiving order. *openflow_replydict* is the dictionary *{request:reply}*.

Raises ***OpenflowErrorResultException*** – when some replies are errors. *exc.result* returns (openflow_reply, openflow_replydict)

closed(*connection*)
routine for connection closed

error(*connection*)
routine for connection error

init(*connection*)
routine for connection initialization

keepalive(*connection*)
routine executed when there has been a long time since last data arrival. Check if the connection is down.

parse(*connection*, *data*, *laststart*)
Parse input data into events

Parameters

- **connection** – connection object
- **data** – view for input data
- **laststart** – last parsed position

Returns (events, keep) where events are parsed events to send, keep is the unused data length to be keeped for next parse.

querymultipart(*request*, *connection*, *container=None*, *raiseonerror=True*)
Send a multipart request, wait for all the responses. Return a list of reply messages

querywithreply(*request*, *connection*, *container=None*, *raiseonerror=True*)
Send an OpenFlow normal request, wait for the response of this request. The request must have exactly one response.

reconnect_init(*connection*)
routine for reconnect

replymatcher(*request*, *connection*, *iserror=None*)
Create an event matcher to match a reply to this request

statematcher(*connection*, *state='down'*, *currentconn=True*)
Create an event matcher to match the connection state of this connection

class vlcp.protocol.openflow.openflow.**OpenflowAsyncMessageEvent**(**args*, ***kwargs*)

Event for an async message is received

class vlcp.protocol.openflow.openflow.**OpenflowConnectionStateEvent**(**args*, ***kwargs*)

Event when connection state changes

exception vlcp.protocol.openflow.openflow.**OpenflowErrorResultException**(*errormsg*, *prompt='An error message is returned: '*, *result=None*)

OpenFlow returns error

__init__(*errormsg*, *prompt='An error message is returned: '*, *result=None*)

Initialize self. See help(type(self)) for accurate signature.

class vlcp.protocol.openflow.openflow.**OpenflowExperimenterMessageEvent**(**args*, ***kwargs*)

Event for experimenter messages

class vlcp.protocol.openflow.openflow.**OpenflowPresetupMessageEvent**(**args*, ***kwargs*)

Event for messages before connection setup

exception vlcp.protocol.openflow.openflow.**OpenflowProtocolException**

Critical protocol break exception

class vlcp.protocol.openflow.openflow.**OpenflowResponseEvent**(**args*, ***kwargs*)

Event for an OpenFlow response is received

5.3.2 vlcp.protocol.http

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/8/18

author hubo

class vlcp.protocol.http.**Http**(*server=True*, *defaultversion=None*)

Basic HTTP/1.1 protocol Base on RFC723x, which are more strict than RFC2616

__init__(*server=True*, *defaultversion=None*)

Constructor

beforelisten(*tcpserver*, *newsock*)

routine before a socket entering listen mode

closed(*connection*)
routine for connection closed

error(*connection*)
routine for connection error

final(*connection*)
routine for a connection finally ends: all connections are closed and not retrying

init(*connection*)
routine for connection initialization

notconnected(*connection*)
routine for connect failed and not retrying

parse(*connection*, *data*, *laststart*)
Parse input data into events

Parameters

- **connection** – connection object
- **data** – view for input data
- **laststart** – last parsed position

Returns (events, keep) where events are parsed events to send, keep is the unused data length to be keeped for next parse.

reconnect_init(*connection*)
routine for reconnect

request_with_response(*container*, *connection*, *host*, *path=b'/'*, *method=b'GET'*, *headers=[]*, *stream=None*, *keepalive=True*)
Send a HTTP request, and wait for the response. The last (usually wanted) response is stored in *http_finalresponse*. There may be multiple responses (1xx) for this request, they are stored in *http_responses*

Returns (http_finalresponse, http_responses)

requestwithresponse(*container*, *connection*, *host*, *path=b'/'*, *method=b'GET'*, *headers=[]*, *stream=None*, *keepalive=True*)
Send a HTTP request, and wait for the response. The last (usually wanted) response is stored in *http_finalresponse*. There may be multiple responses (1xx) for this request, they are stored in *http_responses*

Returns (http_finalresponse, http_responses)

responseTo(*connection*, *xid*, *response*)
Return an event if notify is necessary

response_to(*connection*, *xid*, *response*)
Return an event if notify is necessary

responsematcher(*connection*, *xid*, *isfinal=None*, *iserror=None*)
Create an event matcher to match the response

sendRequest(*connection*, *host*, *path=b'/'*, *method=b'GET'*, *headers=[]*, *stream=None*, *keepalive=True*)
If you do not provide a content-length header, the stream will be transfer-encoded with chunked, and it is not always acceptable by servers.

You may provide a MemoryStream, and it will provide a content-length header automatically

Return xid

send_request(*connection*, *host*, *path=b'/'*, *method=b'GET'*, *headers=[]*, *stream=None*, *keepalive=True*)
If you do not provide a content-length header, the stream will be transfer-encoded with chunked, and it is not always acceptable by servers.

You may provide a MemoryStream, and it will provide a content-length header automatically

Return xid

serverfinal(*tcpserver*)
routine for a tcpserver finally shutdown or not connected

startResponse(*connection*, *xid*, *status*, *headers*, *outputstream*, *disabledeflate=False*)
Start to response to a request with the specified xid on the connection, with status code and headers. The output stream is used to output the response body.

start_response(*connection*, *xid*, *status*, *headers*, *outputstream*, *disabledeflate=False*)
Start to response to a request with the specified xid on the connection, with status code and headers. The output stream is used to output the response body.

statematcher(*connection*, *state='clientclose'*, *currentconn=True*)
Create an event matcher to match the connection state

exception vlcp.protocol.http.**HttpConnectionClosedException**
Connection is closed

class vlcp.protocol.http.**HttpConnectionStateEvent**(**args*, ***kwargs*)
HTTP connection state changed

exception vlcp.protocol.http.**HttpProtocolException**
Critical protocol break on HTTP connections

class vlcp.protocol.http.**HttpRequestEvent**(**args*, ***kwargs*)
A HTTP request is received from the connection

canignorenow()
Extra criteria for an event with canignore = False. When this event returns True, the event is safely ignored.

class vlcp.protocol.http.**HttpResponseEndEvent**(**args*, ***kwargs*)
A HTTP response is fully received

class vlcp.protocol.http.**HttpResponseEvent**(**args*, ***kwargs*)
A HTTP response is received from the connection

class vlcp.protocol.http.**HttpStateChange**(**args*, ***kwargs*)

class vlcp.protocol.http.**HttpTrailersReceived**(**args*, ***kwargs*)
Trailers are received on an HTTPResponseStream

vlcp.protocol.http.**date_time_string**(*timestamp=None*)
Return the current date and time formatted for a message header.

vlcp.protocol.http.**escape**(*s*, *quote=True*)
Replace special characters "&", "<" and ">" to HTML-safe sequences. If the optional flag quote is true, the quotation mark character (") is also translated.

vlcp.protocol.http.**escape_b**(*s*, *quote=True*)
Replace special characters "&", "<" and ">" to HTML-safe sequences. If the optional flag quote is true, the quotation mark character (") is also translated.

5.3.3 vlcp.protocol.jsonrpc

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/8/12

author hubo

exception vlcp.protocol.jsonrpc.**JsonFormatException**

class vlcp.protocol.jsonrpc.**JsonRPC**

JSON-RPC 1.0 Protocol

__init__()

Constructor

closed(*connection*)

routine for connection closed

error(*connection*)

routine for connection error

init(*connection*)

routine for connection initialization

notificationmatcher(*method*, *connection*)

Create an event matcher to match specified notifications

parse(*connection*, *data*, *laststart*)

Parse input data into events

Parameters

- **connection** – connection object
- **data** – view for input data
- **laststart** – last parsed position

Returns (events, keep) where events are parsed events to send, keep is the unused data length to be keeped for next parse.

querywithreply(*method*, *params*, *connection*, *container=None*, *raiseonerror=True*)

Send a JSON-RPC request and wait for the reply.

Returns (result, error) tuple

reconnect_init(*connection*)

routine for reconnect

replymatcher(*requestid*, *connection*, *iserror=None*)

Create a matcher to match a reply

statematcher(*connection*, *state='down'*, *currentconn=True*)

Create an event matcher to match the connection state

waitfornotify(*method*, *connection*, *container*)

Wait for next notification

Returns (method, params) from the notification

class vlcp.protocol.jsonrpc.**JsonRPCConnectionStateEvent**(**args*, ***kwargs*)
Connection state change

exception vlcp.protocol.jsonrpc.**JsonRPCErrorResultException**(*error*, *result=None*)

__init__(*error*, *result=None*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.protocol.jsonrpc.**JsonRPCNotificationEvent**(**args*, ***kwargs*)
Notification received from the connection

exception vlcp.protocol.jsonrpc.**JsonRPCProtocolException**

class vlcp.protocol.jsonrpc.**JsonRPCRequestEvent**(**args*, ***kwargs*)
Request received from the connection

canignorenow()
Extra criteria for an event with canignore = False. When this event returns True, the event is safely ignored.

class vlcp.protocol.jsonrpc.**JsonRPCResponseEvent**(**args*, ***kwargs*)
Response received from the connection

5.3.4 vlcp.protocol.ovsdb

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/2/19

author hubo

class vlcp.protocol.ovsdb.**OVSDB**
OVSDB protocol, this is a specialized JSON-RPC 1.0 protocol

keepalive(*connection*)
routine executed when there has been a long time since last data arrival. Check if the connection is down.

reconnect_init(*connection*)
routine for reconnect

5.3.5 vlcp.protocol.protocol

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/6/29

author hubo

class vlcp.protocol.protocol.**Protocol**
Protocol base class

__init__()
Constructor

accept(*server*, *newaddr*, *newsocket*)
server accept :returns: new protocol object

beforelisten(*tcpserver*, *newsocket*)
routine before a socket entering listen mode

closed(*connection*)
routine for connection closed

error(*connection*)
routine for connection error

final(*connection*)
routine for a connection finally ends: all connections are closed and not retrying

init(*connection*)
routine for connection initialization

keepalive(*connection*)
routine executed when there has been a long time since last data arrival. Check if the connection is down.

notconnected(*connection*)
routine for connect failed and not retrying

parse(*connection*, *data*, *laststart*)
Parse input data into events

Parameters

- **connection** – connection object
- **data** – view for input data
- **laststart** – last parsed position

Returns (events, keep) where events are parsed events to send, keep is the unused data length to be keeped for next parse.

reconnect_init(*connection*)
routine for reconnect

serialize(*connection*, *event*)
Serialize a write event to bytes, and return if it is EOF

Parameters

- **connection** – connection object
- **event** – write event

Returns (bytes, EOF)

serverfinal(*tcpserver*)
routine for a tcpserver finally shutdown or not connected

5.3.6 vlcp.protocol.raw

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/12/25

author hubo

class vlcp.protocol.raw.**Raw**

Raw protocol, provide two streams for input and output

__init__()

Constructor

client_connect(*container*, *url*, **args*, ***kwargs*)

Create a connection with raw protocol

Parameters

- **container** – current routine container
- **url** – url to connect to (see Client)
- ****kwargs** (**args,*) – other parameters to create a Client (except url, protocol and scheduler)

Returns *(connection, inputstream, outputstream)* where client is the created connection, inputstream is the stream to read from the socket, outputstream is the stream to write to socket

closed(*connection*)

routine for connection closed

error(*connection*)

routine for connection error

init(*connection*)

routine for connection initialization

notconnected(*connection*)

routine for connect failed and not retrying

parse(*connection*, *data*, *laststart*)

Parse input data into events

Parameters

- **connection** – connection object
- **data** – view for input data
- **laststart** – last parsed position

Returns (events, keep) where events are parsed events to send, keep is the unused data length to be keeped for next parse.

reconnect_init(*connection*)

routine for reconnect

redirect_outputstream(*connection*, *stream*)

Close current outputstream and output from the new stream

class vlcp.protocol.raw.**RawConnectionStateEvent**(**args*, ***kwargs*)

5.3.7 vlcp.protocol.redis

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/1/5

author hubo

class vlcp.protocol.redis.**Redis**
Redis (RESP) Protocol

__init__()
Constructor

batch_execute(*connection*, *container*, **cmds*, *raise_first_exception=False*)
Send multiple commands to redis server at once, and get responses

Parameters

- **connection** – redis connection
- **container** – routine container
- ***cmds** – commands to send. Each command is a tuple/list of bytes/str.
- **raise_first_exception** – if True, the first exception is raised. if False, exceptions are returned in the list.

Returns list of replies.

closed(*connection*)
routine for connection closed

error(*connection*)
routine for connection error

execute_command(*connection*, *container*, **args*)
Send command to Redis server and wait for response

Parameters

- **connection** – Redis connection
- **container** – routine container
- ***args** – command paramters, begin with command name, e.g. *'SET'*,''key',''value''

Returns Response from Redis server

Raises ***RedisReplyException*** – Redis server returns an error (e.g. "-ERR . . . ")

init(*connection*)
routine for connection initialization

keepalive(*connection*)
routine executed when there has been a long time since last data arrival. Check if the connection is down.

notconnected(*connection*)
routine for connect failed and not retrying

parse(*connection*, *data*, *laststart*)
Parse input data into events

Parameters

- **connection** – connection object
- **data** – view for input data
- **laststart** – last parsed position

Returns (events, keep) where events are parsed events to send, keep is the unused data length to be keeped for next parse.

reconnect_init(*connection*)
routine for reconnect

replymatcher(*requestid*, *connection*, *iserror=None*)
Create an event matcher to match

send_batch(*connection*, *container*, **cmds*)
Send multiple commands to redis server at once

Parameters

- **connection** – redis connection
- **container** – routine container
- ***cmds** – commands to send. Each command is a tuple/list of bytes/str.

Returns list of reply event matchers

send_command(*connection*, *container*, **args*)
Send command to Redis server.

Parameters

- **connection** – Redis connection
- **container** – routine container
- ***args** – command paramters, begin with command name, e.g. *'SET'*,''key',''value''

Returns Event matcher to wait for reply

class vlcp.protocol.redis.**RedisConnectionStateEvent**(**args*, ***kwargs*)

class vlcp.protocol.redis.**RedisParser**
Python implemented hiredis.Reader()

__init__()
Initialize self. See help(type(self)) for accurate signature.

exception vlcp.protocol.redis.**RedisProtocolException**

exception vlcp.protocol.redis.**RedisReplyException**(**args*, ***kwargs*)

__init__(**args*, ***kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.protocol.redis.**RedisResponseEvent**(**args*, ***kwargs*)

class vlcp.protocol.redis.**RedisSubscribeEvent**(**args*, ***kwargs*)

class vlcp.protocol.redis.**RedisSubscribeMessageEvent**(**args*, ***kwargs*)

5.3.8 vlcp.protocol.zookeeper

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/9/13

author think

class vlcp.protocol.zookeeper.**ZooKeeper**
ZooKeeper protocol

__init__()
Constructor

async_requests(*connection*, *requests*, *container=None*, *priority=0*)

Returns (matchers, sendall), where matchers are event matchers for the requests; sendall is an async function to send to requests. Use *await sendall()* to send the requests.

closed(*connection*)
routine for connection closed

error(*connection*)
routine for connection error

init(*connection*)
routine for connection initialization

keepalive(*connection*)
routine executed when there has been a long time since last data arrival. Check if the connection is down.

notconnected(*connection*)
routine for connect failed and not retrying

parse(*connection*, *data*, *laststart*)
Parse input data into events

Parameters

- **connection** – connection object
- **data** – view for input data
- **laststart** – last parsed position

Returns (events, keep) where events are parsed events to send, keep is the unused data length to be keeped for next parse.

reconnect_init(*connection*)
routine for reconnect

requests(*connection*, *requests*, *container=None*, *callback=None*, *priority=0*)
Send requests by sequence, return all the results (including the lost ones)

Params connection ZooKeeper connection

Params requests a sequence of ZooKeeper requests

Params container routine container of current routine

Params callback if not None, *callback(request, response)* is called immediately after each response received

Returns *(responses, lost_responses, retry_requests)*, where responses is a list of responses corresponded to the requests (None if response is not received); lost_responses is a list of requests that are sent but the responses are lost due to connection lost, it is the caller's responsibility to determine whether the call is succeeded or failed; retry_requests are the requests which are not sent and are safe to retry.

serialize(*connection*, *event*)
Serialize a write event to bytes, and return if it is EOF

Parameters

- **connection** – connection object
- **event** – write event

Returns (bytes, EOF)

class vlcp.protocol.zookeeper.**ZooKeeperConnectionStateEvent**(**args*, ***kwargs*)

exception vlcp.protocol.zookeeper.**ZooKeeperException**

class vlcp.protocol.zookeeper.**ZooKeeperHandshakeEvent**(**args*, ***kwargs*)

class vlcp.protocol.zookeeper.**ZooKeeperMessageEvent**(**args*, ***kwargs*)

exception vlcp.protocol.zookeeper.**ZooKeeperProtocolException**

exception vlcp.protocol.zookeeper.**ZooKeeperRequestTooLargeException**
Request is too large, which may break every thing, so we reject it

class vlcp.protocol.zookeeper.**ZooKeeperResponseEvent**(**args*, ***kwargs*)

exception vlcp.protocol.zookeeper.**ZooKeeperRetryException**
Connection lost or not connected on handshake

exception vlcp.protocol.zookeeper.**ZooKeeperSessionExpiredException**
Handshake reports the session is expired

class vlcp.protocol.zookeeper.**ZooKeeperWatcherEvent**(**args*, ***kwargs*)

class vlcp.protocol.zookeeper.**ZooKeeperWriteEvent**(**args*, ***kwargs*)

5.4 vlcp.scripts

Note: This document is generated from the source file.

View Source on GitHub

5.4.1 vlcp.scripts.migratedb

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/10/20

author hubo

class vlcp.scripts.migratedb.**MigrateDB**(*server*)
Migrate data from one database to another

migratedb.py -f <configfile> [–clean] src_module[:src_vhost] dst_module[:dst_vhost]

src_module, dst_module is one of: redisdb, zookeeperdb, defaultdb

5.4.2 vlcp.scripts.repairphymapdb

Note: This document is generated from the source file.

View Source on GitHub

class vlcp.scripts.repairphymapdb.**RepairPhyMapDB**(*server*)
before version 317b31130794650a9392eb15634a2aefaba35c28 have problem about physicalmap don't remove weakref(logicalnetwork) this script repair this problem in DB

5.4.3 vlcp.scripts.script

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/10/20

author hubo

class vlcp.scripts.script.**ScriptModule**(*server*)
Base script module

__init__(*server*)
Constructor

5.5 vlcp.server

Note: This document is generated from the source file.

View Source on GitHub

5.5.1 vlcp.server.module

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/9/30/

author hubo

class vlcp.server.module.**Module**(*server*)
A functional part which can be loaded or unloaded dynamically

__init__(*server*)
Constructor

changestate(*state*, *container*)
Change the current load state.

createAPI(**apidefs*)
Create API definitions on this module. This creates a ModuleAPIHandler and register these *apidefs* to it.

Parameters ***apidefs** – should be return values of *api()* or *publicapi()* functions.

create_api(**apidefs*)
Create API definitions on this module. This creates a ModuleAPIHandler and register these *apidefs* to it.

Parameters ***apidefs** – should be return values of *api()* or *publicapi()* functions.

getServiceName()
Return the targetname (or servicename) for this module

get_service_name()
Return the targetname (or servicename) for this module

load(*container*)
Load module

unload(*container*, *force=False*)
Unload module

class vlcp.server.module.**ModuleAPICall**(**args*, ***kwargs*)

class vlcp.server.module.**ModuleAPIHandler**(*moduleinst*, *apidefs=None*, *allowdiscover=True*, *rejectunknown=True*)
API Handler for modules

__init__(*moduleinst*, *apidefs=None*, *allowdiscover=True*, *rejectunknown=True*)
Create the routine container.

Parameters

- **scheduler** – The scheduler. This must be set; if None is used, it must be set with *container.bind(scheduler)* before using.
- **daemon** – If daemon = True, the main routine *container.main* is set to be a daemon routine. A daemon routine does not stop the scheduler from quitting; if all non-daemon routines are quit, the scheduler stops.

close()
Same as *terminate()*

discover(*details=False*)
Discover API definitions. Set details=true to show details

registerAPI(*name*, *handler*, *container=None*, *discoverinfo=None*, *criteria=None*)
Append new API to this handler

registerAPIs(*apidefs*)
API definition is in format: *(name, handler, container, discoverinfo)*

if the handler is a generator, container should be specified handler should accept two arguments:

```
def handler(name, params):
    ...
```

name is the method name, *params* is a dictionary contains the parameters.

the handler can either return the result directly, or be a generator (async-api), and write the result to container.retvalue on exit. e.g:

```
('method1', self.method1),     # method1 directly returns the result
('method2', self.method2, self) # method2 is an async-api
```

Use api() to automatically generate API definitions.

start(*asyncStart=False*)
Start *container.main* as the main routine.

Parameters **asyncStart** – if True, start the routine in background. By default, the routine starts in foreground, which means it is executed to the first *yield* statement before returning. If the started routine raises an exception, the exception is re-raised to the caller of *start*

unregisterAPI(*name*)
Remove an API from this handler

class vlcp.server.module.**ModuleAPIReply**(**args*, ***kwargs*)

exception vlcp.server.module.**ModuleLoadException**
Raised when module loading failed.

class vlcp.server.module.**ModuleLoadStateChanged**(**args*, ***kwargs*)

class vlcp.server.module.**ModuleLoader**(*server*)
Module loader to load modules. The server object creates this instance automatically, usually you can retrieve the pre-created object from *server.moduleloader*

__init__(*server*)
Create the routine container.

Parameters

- **scheduler** – The scheduler. This must be set; if None is used, it must be set with *container.bind(scheduler)* before using.
- **daemon** – If daemon = True, the main routine *container.main* is set to be a daemon routine. A daemon routine does not stop the scheduler from quitting; if all non-daemon routines are quit, the scheduler stops.

getModuleByName(*targetname*)
Return the module instance for a target name.

get_module_by_name(*targetname*)
Return the module instance for a target name.

loadByPath(*path*)
Load a module by full path. If there are dependencies, they are also loaded.

load_by_path(*path*)
Load a module by full path. If there are dependencies, they are also loaded.

loadmodule(*module*)
Load a module class

main()
The main routine method, should be rewritten to an async method

reloadModules(*pathlist*)
Reload modules with a full path in the pathlist

reload_modules(*pathlist*)
Reload modules with a full path in the pathlist

unloadByPath(*path*)
Unload a module by full path. Dependencies are automatically unloaded if they are marked to be services.

unload_by_path(*path*)
Unload a module by full path. Dependencies are automatically unloaded if they are marked to be services.

unloadmodule(*module*, *ignoreDependencies=False*)
Unload a module class

class vlcp.server.module.**ModuleNotification**(**args*, ***kwargs*)

vlcp.server.module.**api**(*func*, *container=None*, *criteria=None*)
Return an API def for a generic function
Parameters

- **func** – a function or bounded method
- **container** – if None, this is used as a synchronous method, the return value of the method is used for the return value. If not None, this is used as an asynchronous method, the return value should be a generator, and it is executed in *container* as a routine. The return value should be set to *container.retvalue*.
- **criteria** – An extra function used to test whether this function should process the API. This allows multiple API definitions to use the same API method name.

vlcp.server.module.**batchCallAPI**(*container*, *apis*, *timeout=120.0*)
DEPRECATED - use execute_all instead

vlcp.server.module.**batch_call_api**(*container*, *apis*, *timeout=120.0*)
DEPRECATED - use execute_all instead

vlcp.server.module.**callAPI**(*container*, *targetname*, *name*, *params={}*, *timeout=120.0*)
Call module API *targetname/name* with parameters.
Parameters

- **targetname** – module targetname. Usually the lower-cased name of the module class, or 'public' for public APIs.
- **name** – method name
- **params** – module API parameters, should be a dictionary of *{parameter: value}*
- **timeout** – raise an exception if the API call is not returned for a long time

Returns API return value

vlcp.server.module.**call_api**(*container*, *targetname*, *name*, *params={}*, *timeout=120.0*)
Call module API *targetname/name* with parameters.
Parameters

- **targetname** – module targetname. Usually the lower-cased name of the module class, or 'public' for public APIs.
- **name** – method name
- **params** – module API parameters, should be a dictionary of *{parameter: value}*

- **timeout** – raise an exception if the API call is not returned for a long time

Returns API return value

vlcp.server.module.**depend**(**args*)
Decorator to declare dependencies to other modules. Recommended usage is:

```
import other_module

@depend(other_module.ModuleClass)
class MyModule(Module):
    ...
```

Parameters ***args** – depended module classes.

vlcp.server.module.**proxy**(*name*, *default=None*)
Create a proxy module. A proxy module has a default implementation, but can be redirected to other implementations with configurations. Other modules can depend on proxy modules.

vlcp.server.module.**publicapi**(*func*, *container=None*, *criteria=None*)
Create an API def for public API processing. Target name of a public API is *'public'*.

Parameters

- **func** – a function or bounded method
- **container** – if None, this is used as a synchronous method, the return value of the method is used for the return value. If not None, this is used as an asynchronous method, the return value should be a generator, and it is executed in *container* as a routine. The return value should be set to *container.retvalue*.
- **criteria** – An extra function used to test whether this function should process the API. This allows multiple API definitions to use the same API method name.

vlcp.server.module.**send_api**(*container*, *targetname*, *name*, *params={}*)
Send API and discard the result

5.5.2 vlcp.server.server

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/7/24

author hubo

class vlcp.server.server.**Server**
Create a server with all necessary parts

__init__()
Constructor

serve()
Start the server

vlcp.server.server.**main**(*configpath=None*, *startup=None*, *daemon=False*, *pidfile=None*, *fork=None*)
The most simple way to start the VLCP framework

Parameters

- **configpath** – path of a configuration file to be loaded
- **startup** – startup modules list. If None, *server.startup* in the configuration files is used; if *server.startup* is not configured, any module defined or imported into __main__ is loaded.
- **daemon** – if True, use python-daemon to fork and start at background. *python-daemon* must be installed:

```
pip install python-daemon
```

- **pidfile** – if daemon=True, this file is used for the pidfile.
- **fork** – use extra fork to start multiple instances

5.6 vlcp.service

Note: This document is generated from the source file.

View Source on GitHub

5.6.1 vlcp.service.connection

Note: This document is generated from the source file.

View Source on GitHub

vlcp.service.connection.httpserver

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/10/19

author hubo

class vlcp.service.connection.httpserver.**HttpServer**(*server*)

Create HTTP server on specified URLs, vHosts are supported.

__init__(*server*)

Constructor

vlcp.service.connection.jsonrpcserver

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/12/25

author hubo

class vlcp.service.connection.jsonrpcserver.**JsonRPCServer**(*server*)
Create JsonRPC server on specified URLs, vHosts are supported.

__init__(*server*)
Constructor

vlcp.service.connection.openflowserver

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/12/25

author hubo

class vlcp.service.connection.openflowserver.**OpenflowServer**(*server*)
Create OpenFlow server on specified URLs, vHosts are supported.

__init__(*server*)
Constructor

vlcp.service.connection.redisdb

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/3/8

author hubo

class vlcp.service.connection.redisdb.**RedisDB**(*server*)
Create redis clients to connect to redis server

__init__(*server*)
Constructor

delete(*key*, *vhost=''*)
Delete a key from the storage

get(*key*, *timeout=None*, *vhost=''*)
Get value from key

getclient(*vhost=''*)
Return a tuple of `(redisclient, encoder, decoder)` for specified vhost

listallkeys(*vhost=''*)
Return all keys in the KVDB. For management purpose.

mget(*keys*, *vhost=''*)
Get multiple values from multiple keys

mgetwithcache(*keys*, *vhost=''*, *cache=None*)
Get multiple values, cached when possible

mset(*kvpairs*, *timeout=None*, *vhost=''*)
Set multiple values on multiple keys

mupdate(*keys*, *updater*, *timeout=None*, *vhost=''*)
Update multiple keys in-place with a custom function, see update. Either all success, or all fail.

set(*key*, *value*, *timeout=None*, *vhost=''*)
Set value to key, with an optional timeout

update(*key*, *updater*, *timeout=None*, *vhost=''*)
Update in-place with a custom function

Parameters

- **key** – key to update
- **updater** – `func(k,v)`, should return a new value to update, or return None to delete. The function may be call more than once.
- **timeout** – new timeout

Returns the updated value, or None if deleted

updateall(*keys*, *updater*, *timeout=None*, *vhost=''*)
Update multiple keys in-place, with a function `updater(keys, values)` which returns `(updated_keys, updated_values)`.

Either all success or all fail

updateallwithtime(*keys*, *updater*, *timeout=None*, *vhost=''*)
Update multiple keys in-place, with a function `updater(keys, values, timestamp)` which returns `(updated_keys, updated_values)`.

Either all success or all fail.

Timestamp is a integer standing for current time in microseconds.

exception vlcp.service.connection.redisdb.**RedisWriteConflictException**

vlcp.service.connection.tcpserver

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/10/19

author hubo

class vlcp.service.connection.tcpserver.**TcpServerBase**(*server*, *protocolclass*)
Generic tcp server on specified URLs, vHosts are supported.

__init__(*server*, *protocolclass*)
Constructor

getconnections(*vhost=None*)
Return accepted connections, optionally filtered by vhost

getservers(*vhost=None*)
Return current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

startlisten(*vhost=None*)
Start listen on current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

stoplisten(*vhost=None*)
Stop listen on current servers

Parameters **vhost** – return only servers of vhost if specified. '' to return only default servers. None for all servers.

unload(*container*, *force=False*)
Unload module

updateconfig()
Reload configurations, remove non-exist servers, add new servers, and leave others unchanged

vlcp.service.connection.zookeeperdb

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/10/9

author hubo

class vlcp.service.connection.zookeeperdb.**UpdateNotification**(**args*, ***kwargs*)

class vlcp.service.connection.zookeeperdb.**ZooKeeperDB**(*server*)
Create zookeeper clients to connect to redis server

__init__(*server*)
Constructor

createnotifier(*vhost=None*)
Create a new notifier object

delete(*key*, *vhost=''*)
Delete a key from the storage

get(*key*, *timeout=None*, *vhost=''*)
Get value from key

getclient(*vhost=''*)
Return a tuple of `(zookeeperclient, encoder, decoder)` for specified vhost

listallkeys(*vhost=''*)
Return all keys in the KVDB. For management purpose.

mget(*keys*, *vhost=''*)
Get multiple values from multiple keys

mgetwithcache(*keys*, *vhost=''*, *cache=None*)
Get multiple values, cached when possible

mset(*kvpairs*, *timeout=None*, *vhost=''*)
Set multiple values on multiple keys

mupdate(*keys*, *updater*, *timeout=None*, *vhost=''*)
Update multiple keys in-place with a custom function, see update.

Either all success, or all fail.

recycle(*keys*, *vhost=''*)
Recycle extra versions from the specified keys.

set(*key*, *value*, *timeout=None*, *vhost=''*)
Set value to key, with an optional timeout

update(*key*, *updater*, *timeout=None*, *vhost=''*)
Update in-place with a custom function

Parameters

- **key** – key to update
- **updater** – `func(k,v)`, should return a new value to update, or return None to delete. The function may be call more than once.
- **timeout** – new timeout

Returns the updated value, or None if deleted

updateall(*keys*, *updater*, *timeout=None*, *vhost=''*)
Update multiple keys in-place, with a function `updater(keys, values)` which returns `(updated_keys, updated_values)`.

Either all success or all fail

updateallwithtime(*keys*, *updater*, *timeout=None*, *vhost=''*)
Update multiple keys in-place, with a function `updater(keys, values, timestamp)` which returns `(updated_keys, updated_values)`.

Either all success or all fail.

Timestamp is a integer standing for current time in microseconds.

exception vlcp.service.connection.zookeeperdb.**ZooKeeperResultException**

5.6.2 vlcp.service.debugging

Note: This document is generated from the source file.

View Source on GitHub

vlcp.service.debugging.console

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/12/29

author hubo

class vlcp.service.debugging.console.**Console**(*server*)
VLCP debugging console.

Besides the normal functions of Python interactive console, Following variables are provided for debugging purpose:
server, manager, container
Following functions can be used to control VLCP running:
callapi, capture, sendevent, subroutine, execute, breakpoint, syscall, resume, debug, restore_console, console_help
For details call console_help()

__init__(*server*)
Constructor

class vlcp.service.debugging.console.**ConsoleEvent**(**args*, ***kwargs*)

class vlcp.service.debugging.console.**ConsoleServiceCall**(**args*, ***kwargs*)

class vlcp.service.debugging.console.**ConsoleServiceCancel**(**args*, ***kwargs*)

class vlcp.service.debugging.console.**InterruptPoller**(**args*, ***kwargs*)

class vlcp.service.debugging.console.**SocketInjectDone**(**args*, ***kwargs*)

5.6.3 vlcp.service.kvdb

Note: This document is generated from the source file.

View Source on GitHub

vlcp.service.kvdb.objectdb

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/3/24

author hubo

class vlcp.service.kvdb.objectdb.**ObjectDB**(*server*)
Abstract transaction layer for KVDB

__init__(*server*)
Constructor

asynctransact(*asyncupdater*, *withtime=False*, *maxretry=None*, *maxtime=60*)
Read-Write transaction with asynchronous operations.

First, the *asyncupdater* is called with *asyncupdater(last_info, container)*. *last_info* is the info from last *AsyncTransactionLockException*. When *asyncupdater* is called for the first time, last_info = None.

The async updater should be an async function, and return *(updater, keys)*. The *updater* should be a valid updater function used in *transaction* API. *keys* will be the keys used in the transaction.

The async updater can return None to terminate the transaction without exception.

After the call, a transaction is automatically started with the return values of *asyncupdater*.

updater can raise *AsyncTransactionLockException* to restart the transaction from *asyncupdater*.

Parameters

- **asyncupdater** – An async updater *asyncupdater(last_info, container)* which returns *(updater, keys)*
- **withtime** – Whether the returned updater need a timestamp
- **maxretry** – Limit the max retried times
- **maxtime** – Limit the execution time. The transaction is abandoned if still not completed after *maxtime* seconds.

asyncwritewalk(*asyncwalker*, *withtime=False*, *maxtime=60*)
A read-write transaction with walker factory

Parameters

- **asyncwalker** – an async function called as *asyncwalker(last_info, container)* and returns (keys, walker), which are the same as parameters of *writewalk*

  param keys initial keys used in walk

  param walker A walker should be *walker(walk, write)*, where *walk* is a function *walk(key)->value* to get a value from the database, and *write* is a function *write(key, value)* to save value to the database.

  A value can be write to a database any times. A *walk* called after *write* is guaranteed to retrieve the previously written value.

  raise AsyncTransactionLockException in walkers to restart the transaction
- **withtime** – if withtime=True, an extra timestamp parameter is given to walkers, so walkers should be *walker(key, value, walk, write, timestamp)*
- **maxtime** – max execution time of this transaction

get(*key*, *requestid*, *nostale=False*)
Get an object from specified key, and manage the object. Return a reference to the object or None if not exists.

getonce(*key*, *nostale=False*)
Get a object without manage it. Return a copy of the object, or None if not exists. Referenced objects are not retrieved.

gettimestamp()
Get a timestamp from database server

load(*container*)
Load module

mget(*keys*, *requestid*, *nostale=False*)
Get multiple objects and manage them. Return references to the objects.

mgetonce(*keys*, *nostale=False*)
Get multiple objects, return copies of them. Referenced objects are not retrieved.

munwatch(*keys*, *requestid*)
Cancel management of keys

mwatch(*keys*, *requestid*, *nostale=False*)
Try to return all the references, see `watch()`

transact(*keys*, *updater*, *withtime=False*, *maxtime=60*)
Try to update keys in a transact, with an `updater(keys, values)`, which returns `(updated_keys, updated_values)`.

The updater may be called more than once. If `withtime = True`, the updater should take three parameters: `(keys, values, timestamp)` with timestamp as the server time

unload(*container*, *force=False*)
Unload module

unwatch(*key*, *requestid*)
Cancel management of a key

unwatchall(*requestid*)
Cancel management for all keys that are managed by requestid

walk(*keys*, *walkerdict*, *requestid*, *nostale=False*)
Recursively retrieve keys with customized functions. walkerdict is a dictionary `key->walker(key, obj, walk, save)`.

watch(*key*, *requestid*, *nostale=False*)
Try to find an object and return a reference. Use `reference.isdeleted()` to test whether the object exists. Use `reference.wait(container)` to wait for the object to be existed.

watchlist(*requestid=None*)
Return a dictionary whose keys are database keys, and values are lists of request ids. Optionally filtered by request id

writewalk(*keys*, *walker*, *withtime=False*, *maxtime=60*)
A read-write transaction with walkers

Parameters

- **keys** – initial keys used in walk. Provide keys already known to be necessary to optimize the transaction.
- **walker** – A walker should be *walker(walk, write)*, where *walk* is a function *walk(key)->value* to get a value from the database, and *write* is a function *write(key, value)* to save value to the database.

  A value can be write to a database any times. A *walk* called after *write* is guaranteed to retrieve the previously written value.
- **withtime** – if withtime=True, an extra timestamp parameter is given to walkers, so walker should be *walker(walk, write, timestamp)*
- **maxtime** – max execution time of this transaction

class vlcp.service.kvdb.objectdb.**RetrieveReply**(**args*, ***kwargs*)

class vlcp.service.kvdb.objectdb.**RetrieveRequestSend**(**args*, ***kwargs*)

vlcp.service.kvdb.redisnotifier

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/3/21

author hubo

class vlcp.service.kvdb.redisnotifier.**ModifyListen**(**args*, ***kwargs*)

class vlcp.service.kvdb.redisnotifier.**RedisNotifier**(*server*)
Update notification with Redis Pub/Sub

__init__(*server*)
Constructor

createnotifier()
Create a new notifier object

load(*container*)
Load module

unload(*container*, *force=False*)
Unload module

class vlcp.service.kvdb.redisnotifier.**UpdateNotification**(**args*, ***kwargs*)

class vlcp.service.kvdb.redisnotifier.**UpdateNotifier**(*server*)

vlcp.service.kvdb.storage

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/3/22

author hubo

class vlcp.service.kvdb.storage.**KVStorage**(*server*)

5.6.4 vlcp.service.manage

Note: This document is generated from the source file.

View Source on GitHub

vlcp.service.manage.modulemanager

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/12/2

author hubo

class vlcp.service.manage.modulemanager.**Manager**(*server*)
Manage module loading/unloading. Optionally reload a module when modified.

__init__(*server*)
Constructor

activeModules()
Return current loaded modules

enableAutoReload(*enabled=True*)
Enable or disable auto reload.

Parameters **enabled** – enable if True, disable if False

loadmodule(*path*)
Load specified module

Parameters **path** – module path (e.g. vlcp.service.connection.httpserver.HttpServer)

reloadmodules(*pathlist*)
Reload specified modules.

Parameters **pathlist** – list of module path

unloadmodule(*path*)
Unload specified module

Parameters **path** – module path (e.g. vlcp.service.connection.httpserver.HttpServer)

vlcp.service.manage.webapi

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/12/2

author hubo

class vlcp.service.manage.webapi.**WebAPI**(*server*)
Call module API from web. Free access to any module APIs may create serious security problems, make sure to configure this module properly.

__init__(*server*)
Constructor

class vlcp.service.manage.webapi.**WebAPIHandler**(*parent*)

__init__(*parent*)
Create the routine container.

Parameters

- **scheduler** – The scheduler. This must be set; if None is used, it must be set with *container.bind(scheduler)* before using.

- **daemon** – If daemon = True, the main routine *container.main* is set to be a daemon routine. A daemon routine does not stop the scheduler from quitting; if all non-daemon routines are quit, the scheduler stops.

start(*asyncStart=False*)
Start *container.main* as the main routine.

Parameters **asyncStart** – if True, start the routine in background. By default, the routine starts in foreground, which means it is executed to the first *yield* statement before returning. If the started routine raises an exception, the exception is re-raised to the caller of *start*

5.6.5 vlcp.service.sdn

Note: This document is generated from the source file.

View Source on GitHub

vlcp.service.sdn.plugins

Note: This document is generated from the source file.

View Source on GitHub

vlcp.service.sdn.plugins.networklocaldriver

Note: This document is generated from the source file.

View Source on GitHub

Physical network with type=local can have unlimited logical networks, but it cannot have physical ports. So, logical ports in the same logical network can access each other only when they are on the same host.

class vlcp.service.sdn.plugins.networklocaldriver.**NetworkLocalDriver**(*server*)
Network driver for local networks. Local networks cannot have physical ports; logical networks in local networks do not have external connectivities, only endpoints on the same server can access each other.

__init__(*server*)
Constructor

vlcp.service.sdn.plugins.networknativedriver

Note: This document is generated from the source file.

View Source on GitHub

Physical network with type=native does not have isolation technique, so there can be only one logical network in each physical network.

class vlcp.service.sdn.plugins.networknativedriver.**NetworkNativeDriver**(*server*)
Network driver for native networks. Native network is a physical network provides only one logical network capacity. Packets from the logical network is directly forwarded to the physical network.

__init__(*server*)
Constructor

vlcp.service.sdn.plugins.networkvlandriver

Note: This document is generated from the source file.

View Source on GitHub

Physical network

class vlcp.service.sdn.plugins.networkvlandriver.**NetworkVlanDriver**(*server*)
Network driver for VXLAN networks. When creating a VXLAN type physical network, you must specify an extra option `vlanrange`.

__init__(*server*)
Constructor

vlcp.service.sdn.plugins.networkvxlandriver

Note: This document is generated from the source file.

View Source on GitHub

class vlcp.service.sdn.plugins.networkvxlandriver.**NetworkVxlanDriver**(*server*)
Network driver for VXLAN networks. When creating a VXLAN type physical network, you must specify an extra option `vnirange`.

__init__(*server*)
Constructor

vlcp.service.sdn.arpresponder

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/7/4

author hubo

class vlcp.service.sdn.arpresponder.**ARPResponder**(*server*)
Send ARP respond

__init__(*server*)
Constructor

createproxyarp(*connection*, *arpentries*)
Create ARP respond flow for specified ARP entries, each is a tuple `(ip_address, mac_address, logical_network_id, local)`. When local is True, only respond to ARP request from logical port; when local is False, only respond to ARP request from physical port; respond to both else.

removeproxyarp(*connection*, *arpentries*)
Remove specified ARP entries.

class vlcp.service.sdn.arpresponder.**ARPUpdater**(*connection*, *parent*)

__init__(*connection*, *parent*)
Retrieve data objects from ObjectDB and use them to generate flows

The module starts ObjectDB.walk from initial keys and walkers. After the walk completes, the retrieved data objects are used by *updateflow()* to generate flows and send them to the OpenFlow connection. When the retrieved objects are updated, FlowUpdater either restart the walk process (re-walk) or directly call another *updateflow()*, according to the objects that are updated.

A subclass should re-initialize *self._initialkeys* and *self._walkerdict* before *main()* coroutine starts to customize the process.

updateflow() is guaranteed for no re-entrance i.e. will not be called until the last call returns. Multiple changes may be merged into the same call.

Parameters

- **connection** – OpenFlow connection
- **initialkeys** – DEPRECATED The key list that triggers a re-walk
- **requestid** – request id to retrieve data objects from ObjectDB
- **logger** – inherit a logger from a module

main()
Main coroutine

updateflow(*connection*, *addvalues*, *removevalues*, *updatedvalues*)
Update flow callback. When data objects are updated (either by a re-walk or by a direct update), this method is called with the modification, after the last *updateflow()* call ends.

vlcp.service.sdn.dhcpserver

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/7/19

author hubo

class vlcp.service.sdn.dhcpserver.**DHCPServer**(*server*)
DHCP server that responds the DHCP discover/request with static IP address settings

__init__(*server*)
Constructor

class vlcp.service.sdn.dhcpserver.**DHCPUpdater**(*connection*, *parent*)

__init__(*connection*, *parent*)
Retrieve data objects from ObjectDB and use them to generate flows

The module starts ObjectDB.walk from initial keys and walkers. After the walk completes, the retrieved data objects are used by *updateflow()* to generate flows and send them to the OpenFlow connection. When the retrieved objects are updated, FlowUpdater either restart the walk process (re-walk) or directly call another *updateflow()*, according to the objects that are updated.

A subclass should re-initialize *self._initialkeys* and *self._walkerdict* before *main()* coroutine starts to customize the process.

updateflow() is guaranteed for no re-entrance i.e. will not be called until the last call returns. Multiple changes may be merged into the same call.

Parameters

- **connection** – OpenFlow connection
- **initialkeys** – DEPRECATED The key list that triggers a re-walk
- **requestid** – request id to retrieve data objects from ObjectDB
- **logger** – inherit a logger from a module

main()
Main coroutine

updateflow(*connection*, *addvalues*, *removevalues*, *updatedvalues*)
Update flow callback. When data objects are updated (either by a re-walk or by a direct update), this method is called with the modification, after the last *updateflow()* call ends.

vlcp.service.sdn.flowbase

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/4/11

author hubo

class vlcp.service.sdn.flowbase.**FlowBase**(*server*)

__init__(*server*)
Constructor

gettablerequest()
Table requirement for this module

vlcp.service.sdn.icmpresponder

Note: This document is generated from the source file.

View Source on GitHub

class vlcp.service.sdn.icmpresponder.**ICMPResponder**(*server*)
Respond ICMP echo (ping) requests to the gateway

__init__(*server*)
Constructor

class vlcp.service.sdn.icmpresponder.**ICMPResponderUpdater**(*connection*, *parent*)

__init__(*connection*, *parent*)
Retrieve data objects from ObjectDB and use them to generate flows

The module starts ObjectDB.walk from initial keys and walkers. After the walk completes, the retrieved data objects are used by *updateflow()* to generate flows and send them to the OpenFlow connection. When the retrieved objects are updated, FlowUpdater either restart the walk process (re-walk) or directly call another *updateflow()*, according to the objects that are updated.

A subclass should re-initialize *self._initialkeys* and *self._walkerdict* before *main()* coroutine starts to customize the process.

updateflow() is guaranteed for no re-entrance i.e. will not be called until the last call returns. Multiple changes may be merged into the same call.

Parameters

- **connection** – OpenFlow connection
- **initialkeys** – DEPRECATED The key list that triggers a re-walk
- **requestid** – request id to retrieve data objects from ObjectDB
- **logger** – inherit a logger from a module

main()
Main coroutine

reset_initialkeys(*keys*, *values*)
Callback after walk complete, can be used to update *self._initialkeys*.

updateflow(*connection*, *addvalues*, *removevalues*, *updatedvalues*)
Update flow callback. When data objects are updated (either by a re-walk or by a direct update), this method is called with the modification, after the last *updateflow()* call ends.

vlcp.service.sdn.ioprocessing

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/4/13

author hubo

class vlcp.service.sdn.ioprocessing.**DataObjectChanged**(**args*, ***kwargs*)

class vlcp.service.sdn.ioprocessing.**IOFlowUpdater**(*connection*, *systemid*, *bridgename*, *parent*)

__init__(*connection*, *systemid*, *bridgename*, *parent*)
Retrieve data objects from ObjectDB and use them to generate flows

The module starts ObjectDB.walk from initial keys and walkers. After the walk completes, the retrieved data objects are used by *updateflow()* to generate flows and send them to the OpenFlow connection. When

the retrieved objects are updated, FlowUpdater either restart the walk process (re-walk) or directly call another *updateflow()*, according to the objects that are updated.

A subclass should re-initialize *self._initialkeys* and *self._walkerdict* before *main()* coroutine starts to customize the process.

updateflow() is guaranteed for no re-entrance i.e. will not be called until the last call returns. Multiple changes may be merged into the same call.

Parameters

- **connection** – OpenFlow connection
- **initialkeys** – DEPRECATED The key list that triggers a re-walk
- **requestid** – request id to retrieve data objects from ObjectDB
- **logger** – inherit a logger from a module

reset_initialkeys(*keys*, *values*)
Callback after walk complete, can be used to update *self._initialkeys*.

update_ports(*ports*, *ovsdb_ports*)
Called from main module to update port information

updateflow(*connection*, *addvalues*, *removevalues*, *updatedvalues*)
Update flow callback. When data objects are updated (either by a re-walk or by a direct update), this method is called with the modification, after the last *updateflow()* call ends.

walkcomplete(*keys*, *values*)
Async callback after walk complete, before flow update

class vlcp.service.sdn.ioprocessing.**IOProcessing**(*server*)
Ingress and Egress processing

__init__(*server*)
Constructor

vlcp.service.sdn.l2switch

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/5/24

author hubo

class vlcp.service.sdn.l2switch.**L2FlowUpdater**(*connection*, *parent*)

__init__(*connection*, *parent*)
Retrieve data objects from ObjectDB and use them to generate flows

The module starts ObjectDB.walk from initial keys and walkers. After the walk completes, the retrieved data objects are used by *updateflow()* to generate flows and send them to the OpenFlow connection. When the retrieved objects are updated, FlowUpdater either restart the walk process (re-walk) or directly call another *updateflow()*, according to the objects that are updated.

A subclass should re-initialize *self._initialkeys* and *self._walkerdict* before *main()* coroutine starts to customize the process.

updateflow() is guaranteed for no re-entrance i.e. will not be called until the last call returns. Multiple changes may be merged into the same call.

Parameters

- **connection** – OpenFlow connection
- **initialkeys** – DEPRECATED The key list that triggers a re-walk
- **requestid** – request id to retrieve data objects from ObjectDB
- **logger** – inherit a logger from a module

main()
Main coroutine

updateflow(*conn*, *addvalues*, *removevalues*, *updatedvalues*)
Update flow callback. When data objects are updated (either by a re-walk or by a direct update), this method is called with the modification, after the last *updateflow()* call ends.

class vlcp.service.sdn.l2switch.**L2Switch**(*server*)
L2 switch functions

__init__(*server*)
Constructor

vlcp.service.sdn.l3router

Note: This document is generated from the source file.

View Source on GitHub

class vlcp.service.sdn.l3router.**ARPRequest**(**args*, ***kwargs*)

class vlcp.service.sdn.l3router.**L3Router**(*server*)
L3 connectivities with virtual router.

__init__(*server*)
Constructor

class vlcp.service.sdn.l3router.**RouterUpdater**(*connection*, *parent*)

__init__(*connection*, *parent*)
Retrieve data objects from ObjectDB and use them to generate flows

The module starts ObjectDB.walk from initial keys and walkers. After the walk completes, the retrieved data objects are used by *updateflow()* to generate flows and send them to the OpenFlow connection. When the retrieved objects are updated, FlowUpdater either restart the walk process (re-walk) or directly call another *updateflow()*, according to the objects that are updated.

A subclass should re-initialize *self._initialkeys* and *self._walkerdict* before *main()* coroutine starts to customize the process.

updateflow() is guaranteed for no re-entrance i.e. will not be called until the last call returns. Multiple changes may be merged into the same call.

Parameters

- **connection** – OpenFlow connection
- **initialkeys** – DEPRECATED The key list that triggers a re-walk

- **requestid** – request id to retrieve data objects from ObjectDB
- **logger** – inherit a logger from a module

main()
Main coroutine

reset_initialkeys(*keys*, *values*)
Callback after walk complete, can be used to update *self._initialkeys*.

updateflow(*connection*, *addvalues*, *removevalues*, *updatedvalues*)
Update flow callback. When data objects are updated (either by a re-walk or by a direct update), this method is called with the modification, after the last *updateflow()* call ends.

vlcp.service.sdn.ofpmanager

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/2/19

author hubo

class vlcp.service.sdn.ofpmanager.**FlowInitialize**(**args*, ***kwargs*)

class vlcp.service.sdn.ofpmanager.**OpenflowManager**(*server*)
Manage Openflow Connections

__init__(*server*)
Constructor

acquiretable(*modulename*)
Start to acquire tables for a module on module loading.

getallconnections(*vhost=''*)
Get all connections from vhost. If vhost is None, return all connections from any host

getalldatapathids()
Get all datapath IDs from any vhost. Return `(vhost, datapathid)` pair.

getallendpoints()
Get all endpoints from any vhost. Return `(vhost, endpoint)` pairs.

getconnection(*datapathid*, *auxiliaryid=0*, *vhost=''*)
Get current connection of datapath

getconnections(*datapathid*, *vhost=''*)
Return all connections of datapath

getconnectionsbyendpoint(*endpoint*, *vhost=''*)
Get connection by endpoint address (IP, IPv6 or UNIX socket address)

getconnectionsbyendpointname(*name*, *vhost=''*, *timeout=30*)
Get connection by endpoint name (Domain name, IP or IPv6 address)

getdatapathids(*vhost=''*)
Get All datapath IDs

getendpoints(*vhost=''*)
Get all endpoints for vhost

lastacquiredtables(*vhost=''*)
Get acquired table IDs

load(*container*)
Load module

unacquiretable(*modulename*)
When module is unloaded, stop acquiring tables for this module.

unload(*container*, *force=False*)
Unload module

waitconnection(*datapathid*, *auxiliaryid=0*, *timeout=30*, *vhost=''*)
Wait for a datapath connection

class vlcp.service.sdn.ofpmanager.**TableAcquireDelayEvent**(**args*, ***kwargs*)

class vlcp.service.sdn.ofpmanager.**TableAcquireUpdate**(**args*, ***kwargs*)

vlcp.service.sdn.ofpportmanager

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/2/23

author hubo

class vlcp.service.sdn.ofpportmanager.**OpenflowPortManager**(*server*)
Manage Ports from Openflow Protocol

__init__(*server*)
Constructor

getallports(*vhost=None*)
Return all `(datapathid, port, vhost)` tuples, optionally filterd by vhost

getportbyname(*datapathid*, *name*, *vhost=''*)
Return port with specified port name

getportbyno(*datapathid*, *portno*, *vhost=''*)
Return port with specified OpenFlow portno

getports(*datapathid*, *vhost=''*)
Return all ports of a specifed datapath

resync(*datapathid*, *vhost=''*)
Resync with current ports

waitportbyname(*datapathid*, *name*, *timeout=30*, *vhost=''*)
Wait for a port with the specified port name to appear, or until timeout

waitportbyno(*datapathid*, *portno*, *timeout=30*, *vhost=''*)
Wait for the specified OpenFlow portno to appear, or until timeout.

exception vlcp.service.sdn.ofpportmanager.**OpenflowPortNotAppearException**

class vlcp.service.sdn.ofpportmanager.**OpenflowPortSynchronized**(**args*, ***kwargs*)

vlcp.service.sdn.ovsdbmanager

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/2/19

author hubo

exception vlcp.service.sdn.ovsdbmanager.**OVSDBBridgeNotAppearException**

class vlcp.service.sdn.ovsdbmanager.**OVSDBBridgeSetup**(**args*, ***kwargs*)

class vlcp.service.sdn.ovsdbmanager.**OVSDBConnectionSetup**(**args*, ***kwargs*)

class vlcp.service.sdn.ovsdbmanager.**OVSDBManager**(*server*)

Manage Openflow Connections

__init__(*server*)

Constructor

getallbridges(*vhost=None*)

Get all `(dpid, name, _uuid)` tuple for all connections, optionally filtered by vhost

getallconnections(*vhost=''*)

Get all connections from vhost. If vhost is None, return all connections from any host

getalldatapathids()

Get all datapath IDs from any vhost. Return `(vhost, datapathid)` pair.

getallendpoints()

Get all endpoints from any vhost. Return `(vhost, endpoint)` pairs.

getallsystemids()

Get all system-ids from any vhost. Return `(vhost, system-id)` pair.

getbridge(*connection*, *name*)

Get datapath ID on this connection with specified name

getbridgebyuuid(*connection*, *uuid*)

Get datapath ID of bridge on this connection with specified _uuid

getbridgeinfo(*datapathid*, *vhost=''*)

Get `(bridgename, systemid, bridge_uuid)` tuple from bridge datapathid

getbridges(*connection*)

Get all `(dpid, name, _uuid)` tuple on this connection

getconnection(*datapathid*, *vhost=''*)

Get current connection of datapath

getconnectionsbyendpoint(*endpoint*, *vhost=''*)

Get connection by endpoint address (IP, IPv6 or UNIX socket address)

getconnectionsbyendpointname(*name*, *vhost=''*, *timeout=30*)

Get connection by endpoint name (Domain name, IP or IPv6 address)

getdatapathids(*vhost=''*)

Get All datapath IDs

getendpoints(*vhost=''*)

Get all endpoints for vhost

getsystemids(*vhost=''*)
Get All system-ids

waitbridge(*connection*, *name*, *timeout=30*)
Wait for bridge with specified name appears and return the datapath-id

waitbridgebyuuid(*connection*, *uuid*, *timeout=30*)
Wait for bridge with specified _uuid appears and return the datapath-id

waitbridgeinfo(*datapathid*, *timeout=30*, *vhost=''*)
Wait for bridge with datapathid, and return `(bridgename, systemid, bridge_uuid)` tuple

waitconnection(*datapathid*, *timeout=30*, *vhost=''*)
Wait for a datapath connection

waitconnectionbysystemid(*systemid*, *timeout=30*, *vhost=''*)
Wait for a connection with specified system-id

vlcp.service.sdn.ovsdbportmanager

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/2/26

author think

class vlcp.service.sdn.ovsdbportmanager.**OVSDBConnectionPortsSynchronized**(**args*, ***kwargs*)

class vlcp.service.sdn.ovsdbportmanager.**OVSDBPortManager**(*server*)
Manage Ports from OVSDB Protocol

__init__(*server*)
Constructor

getallports(*vhost=None*)
Return all `(datapathid, port, vhost)` tuples, optionally filterd by vhost

getportbyid(*id*, *vhost=''*)
Return port with the specified id. The return value is a pair: `(datapath_id, port)`

getportbyname(*datapathid*, *name*, *vhost=''*)
Return port with specified name

getportbyno(*datapathid*, *portno*, *vhost=''*)
Return port with specified portno

getports(*datapathid*, *vhost=''*)
Return all ports of a specifed datapath

resync(*datapathid*, *vhost=''*)
Resync with current ports

waitportbyid(*id*, *timeout=30*, *vhost=''*)
Wait for port with the specified id. The return value is a pair `(datapath_id, port)`

waitportbyname(*datapathid*, *name*, *timeout=30*, *vhost=''*)
Wait for port with specified name

waitportbyno(*datapathid*, *portno*, *timeout=30*, *vhost=''*)
Wait for port with specified portno

exception vlcp.service.sdn.ovsdbportmanager.**OVSDBPortNotAppearException**

class vlcp.service.sdn.ovsdbportmanager.**OVSDBPortUpNotification**(**args*, ***kwargs*)

vlcp.service.sdn.viperflow

Note: This document is generated from the source file.

View Source on GitHub

exception vlcp.service.sdn.viperflow.**UpdateConflictException**(*desc='db update conflict'*)

__init__(*desc='db update conflict'*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.service.sdn.viperflow.**ViperFlow**(*server*)
Standard network model for L2 SDN

__init__(*server*)
Constructor

createlogicalnetwork(*physicalnetwork: str*, *id: (<class 'str'>, None) = None*, ***kwargs*)
Create logical network

Parameters

- **physicalnetwork** – physical network ID that contains this logical network
- **id** – logical network ID. If ommited an UUID is generated.
- ****kwargs** – customized options for logical network creation. Common options include:

  vni/vxlan Specify VNI / VLAN tag for VXLAN / VLAN network. If omitted, an unused VNI / VLAN tag is picked automatically.

  mtu MTU value for this network. You can use 1450 for VXLAN networks.

Returns A dictionary of information of the created logical port

createlogicalnetworks(*networks: [{'?mtu': (<class 'int'>, extra(<class 'str'>)), '?vni': (<class 'int'>, extra(<class 'str'>)), 'physicalnetwork': <class 'str'>, '?dns_nameservers': ([extra(<class 'str'>)], None), '?vxlan': (<class 'int'>, extra(<class 'str'>)), '?domain_name': (<class 'str'>, None), '?id': <class 'str'>, '?lease_time': ((<class 'int'>, extra(<class 'str'>)), extra(<class 'str'>), None), '?ntp_servers': ([extra(<class 'str'>)], None), '?extra_dhcp_options': extra([tuple_(((<class 'int'>, extra(<class 'str'>)), <class 'object'>))])}]*)
Create multiple logical networks in a transaction.

Parameters **networks** – a list of `createlogicalnetwork` parameters.

Returns a list of dictionaries for the created logical networks.

createlogicalport (*logicalnetwork: str*, *id: (<class 'str'>*, *None) = None*, *subnet: (<class 'str'>*, *None) = None*, ***kwargs*)

Create logical port

Parameters

- **logicalnetwork** – logical network containing this port
- **id** – logical port id. If omitted an UUID is created.
- **subnet** – subnet containing this port
- ****kwargs** – customized options for creating logical ports. Common options are:

  mac_address port MAC address

  ip_address port IP address

Returns a dictionary for the logical port

createlogicalports (*ports: [{'?mac_address': extra(<class 'str'>), '?extra_dhcp_options': extra([tuple_(((<class 'int'>, extra(<class 'str'>)), <class 'object'>))]), 'logicalnetwork': <class 'str'>, '?hostname': (<class 'str'>, None), '?id': <class 'str'>, '?subnet': <class 'str'>, '?ip_address': extra(<class 'str'>)}]*)

Create multiple logical ports in a transaction

createphysicalnetwork (*type: str = 'vlan'*, *id: (<class 'str'>*, *None) = None*, ***kwargs*)

Create physical network.

Parameters

- **type** – Network type, usually one of *vlan*, *vxlan*, *local*, *native*
- **id** – Specify the created physical network ID. If omitted or None, an UUID is generated.
- ****kwargs** – extended creation parameters. Look for the document of the corresponding driver. Common options include:

  vnirange list of `[start,end]` ranges like `[[1000,2000]]`. Both *start* and *end* are included. It specifies the usable VNI ranges for VXLAN network.

  vlanrange list of `[start,end]` ranges like `[[1000,2000]]`. Both *start* and *end* are included. It specifies the usable VLAN tag ranges for VLAN network.

Returns A dictionary of information of the created physical network.

createphysicalnetworks (*networks: [{'?id': <class 'str'>, 'type': <class 'str'>, '?vlanrange': [tuple_((<class 'int'>, <class 'int'>))], '?vnirange': [tuple_((<class 'int'>, <class 'int'>))]}]*)

Create multiple physical networks in a transaction.

Parameters **networks** – each should be a dictionary contains all the parameters in `createphysicalnetwork`

Returns A list of dictionaries of information of the created physical networks.

createphysicalport (*physicalnetwork: str*, *name: str*, *vhost: str = ''*, *systemid: str = '%'*, *bridge: str = '%'*, ***kwargs*)

Create physical port

Parameters

- **physicalnetwork** – physical network this port is in.
- **name** – port name of the physical port, should match the name in OVSDB

- **vhost** – only match ports for the specified vHost
- **systemid** – only match ports on this systemid; or '%' to match all systemids.
- **bridge** – only match ports on bridges with this name; or '%' to match all bridges.
- ****kwargs** – customized creation options, check the driver document

Returns A dictionary containing information of the created physical port.

createphysicalports(*ports: [{'?vhost': <class 'str'>, 'physicalnetwork': <class 'str'>, '?systemid': <class 'str'>, '?bridge': <class 'str'>, 'name': <class 'str'>}]*)

Create multiple physical ports in a transaction

Parameters **ports** – A list of dictionaries, each contains all parameters for `createphysicalport`

Returns A list of dictionaries of information of the created physical ports

createsubnet(*logicalnetwork: str*, *cidr: extra(<class 'str'>)*, *id: (<class 'str'>, None) = None*, ***kwargs*)

Create a subnet for the logical network.

Parameters

- **logicalnetwork** – The logical network is subnet is in.
- **cidr** – CIDR of this subnet like `"10.0.1.0/24"`
- **id** – subnet ID. If omitted, an UUID is generated.
- ****kwargs** – customized creating options. Common options are:

  gateway Gateway address for this subnet

  allocated_start First IP of the allowed IP range.

  allocated_end Last IP of the allowed IP range.

  host_routes A list of `[dest_cidr, via]` like `[["192.168.1.0/24", "192.168.2.3"],["192.168.3.0/24","192.168.2.4"]]`. This creates static routes on the subnet.

  isexternal This subnet can forward packet to external physical network

  pre_host_config A list of `[{systemid, bridge, cidr, local_address, gateway, ...}]` Per host configuration, will union with public info when used

Returns A dictionary of information of the subnet.

createsubnets(*subnets: [{'?mtu': (<class 'int'>, extra(<class 'str'>)), '?pre_host_config': [{'?vhost': <class 'str'>, '?systemid': <class 'str'>, '?gateway': extra(<class 'str'>), '?cidr': extra(<class 'str'>), '?bridge': <class 'str'>, '?local_address': extra(<class 'str'>)}], 'logicalnetwork': <class 'str'>, '?isexternal': <class 'bool'>, '?id': <class 'str'>, '?ntp_servers': ([extra(<class 'str'>)], None), '?domain_name': (<class 'str'>, None), '?allocated_end': extra(<class 'str'>), '?extra_dhcp_options': extra([tuple_(((<class 'int'>, extra(<class 'str'>)), <class 'object'>))]), '?gateway': extra(<class 'str'>), 'cidr': extra(<class 'str'>), '?allocated_start': extra(<class 'str'>), '?dns_nameservers': ([extra(<class 'str'>)], None), '?lease_time': ((<class 'int'>, extra(<class 'str'>)), extra(<class 'str'>), None), '?host_routes': [tuple_((extra(<class 'str'>), extra(<class 'str'>)))]}]*)

Create multiple subnets in a transaction.

deletelogicalnetwork(*id: str*)
Delete logical network

deletelogicalnetworks(*networks: [{'id': <class 'str'>}]*)
Delete logical networks

Parameters **networks** – a list of `{"id":id}`

Returns `{"status": "OK"}`

deletelogicalport(*id: str*)
Delete logical port

deletelogicalports(*ports: [{'id': <class 'str'>}]*)
Delete multiple logical ports

deletephysicalnetwork(*id: str*)
Delete physical network with specified ID

Parameters **id** – Physical network ID

Returns `{"status": "OK"}`

deletephysicalnetworks(*networks: [{'id': <class 'str'>}]*)
Delete multiple physical networks with a transaction

Parameters **networks** – a list of `{"id": <id>}` dictionaries.

Returns `{"status": "OK"}`

deletephysicalport(*name: str*, *vhost: str = ''*, *systemid: str = '%'*, *bridge: str = '%'*)
Delete a physical port

Parameters

- **name** – physical port name.
- **vhost** – physical port vHost.
- **systemid** – physical port systemid.
- **bridge** – physcial port bridge.

Returns `{"status": "OK"}`

deletephysicalports(*ports: [{'?vhost': <class 'str'>, '?systemid': <class 'str'>, '?bridge': <class 'str'>, 'name': <class 'str'>}]*)
Delete multiple physical ports in a transaction

Delete a physical port

Parameters **ports** – a list of `deletephysicalport` parameters

Returns `{"status": "OK"}`

deletesubnet(*id: str*)
Delete subnet

deletesubnets(*subnets: [{'id': <class 'str'>}]*)
Delete multiple subnets

listlogicalnetworks(*id=None*, *physicalnetwork=None*, ***kwargs*)
Query logical network information

Parameters

- **id** – If specified, only return the logical network with the specified ID.

- **physicalnetwork** – If specified, only return logical networks in this physical network.
- ****kwargs** – customized filters, only return a logical network if the attribute value of this logical network matches the specified value.

Returns A list of dictionaries each stands for a matched logical network

listlogicalports(*id=None*, *logicalnetwork=None*, ***kwargs*)
Query logical port

Parameters

- **id** – If specified, returns only logical port with this ID.
- **logicalnetwork** – If specified, returns only logical ports in this network.
- ****kwargs** – customzied filters

Returns return matched logical ports

listphysicalnetworks(*id=None*, ***kwargs*)
Query physical network information

Parameters

- **id** – If specified, only return the physical network with the specified ID.
- ****kwargs** – customized filters, only return a physical network if the attribute value of this physical network matches the specified value.

Returns A list of dictionaries each stands for a matched physical network

listphysicalports(*name=None*, *physicalnetwork=None*, *vhost=''*, *systemid='%'*, *bridge='%'*, ***kwargs*)
Query physical port information

Parameters

- **name** – If specified, only return the physical port with the specified name.
- **physicalnetwork** – If specified, only return physical ports in that physical network
- **vhost** – If specified, only return physical ports for that vHost.
- **systemid** – If specified, only return physical ports for that systemid.
- **bridge** – If specified, only return physical ports for that bridge.
- ****kwargs** – customized filters, only return a physical network if the attribute value of this physical network matches the specified value.

Returns A list of dictionaries each stands for a matched physical network

listsubnets(*id=None*, *logicalnetwork=None*, ***kwargs*)
Query subnets

Parameters

- **id** – if specified, only return subnet with this ID
- **logicalnetwork** – if specified, only return subnet in the network
- ****kwargs** – customized filters

Returns A list of dictionaries each stands for a matched subnet.

load(*container*)
Load module

updatelogicalnetwork(*id: str*, ***kwargs*)
Update logical network attributes of the ID

updatelogicalnetworks(*networks: [{'?mtu': (<class 'int'>, extra(<class 'str'>)), 'id': <class 'str'>, '?vxlan': (<class 'int'>, extra(<class 'str'>)), '?dns_nameservers': ([extra(<class 'str'>)], None), '?domain_name': (<class 'str'>, None), '?vni': (<class 'int'>, extra(<class 'str'>)), '?lease_time': ((<class 'int'>, extra(<class 'str'>)), extra(<class 'str'>), None), '?ntp_servers': ([extra(<class 'str'>)], None), '?extra_dhcp_options': extra([tuple_(((<class 'int'>, extra(<class 'str'>)), <class 'object'>))])}]*)
Update multiple logical networks in a transaction

updatelogicalport(*id: str*, ***kwargs*)
Update attributes of the specified logical port

updatelogicalports(*ports: [{'?mac_address': extra(<class 'str'>), 'id': <class 'str'>, '?extra_dhcp_options': extra([tuple_(((<class 'int'>, extra(<class 'str'>)), <class 'object'>))]), '?hostname': (<class 'str'>, None), '?ip_address': extra(<class 'str'>)}]*)
Update multiple logcial ports

updatephysicalnetwork(*id: str*, ***kwargs*)
Update physical network with the specified ID.

Parameters

- **id** – physical network ID
- ****kwargs** – attributes to be updated, usually the same attributes for creating.

Returns A dictionary of information of the updated physical network.

updatephysicalnetworks(*networks: [{'id': <class 'str'>, '?vlanrange': [tuple_((<class 'int'>, <class 'int'>))], '?vnirange': [tuple_((<class 'int'>, <class 'int'>))]}]*)
Update multiple physical networks in a transaction

Parameters **networks** – a list of dictionaries, each contains parameters of `updatephysicalnetwork`

Returns A list of dictionaries of information of the updated physical network.

updatephysicalport(*name: str*, *vhost: str = ''*, *systemid: str = '%'*, *bridge: str = '%'*, ***args*)
Update physical port

Parameters

- **name** – Update physical port with this name.
- **vhost** – Update physical port with this vHost.
- **systemid** – Update physical port with this systemid.
- **bridge** – Update physical port with this bridge name.
- ****kwargs** – Attributes to be updated

Returns Updated result as a dictionary.

updatephysicalports(*ports: [{'?vhost': <class 'str'>, '?systemid': <class 'str'>, '?bridge': <class 'str'>, 'name': <class 'str'>}]*)
Update multiple physical ports with a transaction

Parameters **ports** – a list of `updatephysicalport` parameters

Returns Updated result as a list of dictionaries.

updatesubnet(*id: str*, ***kwargs*)
Update subnet attributes

updatesubnets(*subnets: [{'?mtu': (<class 'int'>, extra(<class 'str'>)), '?pre_host_config': [{'?vhost': <class 'str'>, '?systemid': <class 'str'>, '?gateway': extra(<class 'str'>), '?cidr': extra(<class 'str'>), '?bridge': <class 'str'>, '?local_address': extra(<class 'str'>)}], '?allocated_end': extra(<class 'str'>), '?gateway': extra(<class 'str'>), '?domain_name': (<class 'str'>, None), '?ntp_servers': ([extra(<class 'str'>)], None), '?extra_dhcp_options': extra([tuple_(((<class 'int'>, extra(<class 'str'>)), <class 'object'>))]), 'id': <class 'str'>, '?allocated_start': extra(<class 'str'>), '?dns_nameservers': ([extra(<class 'str'>)], None), '?lease_time': ((<class 'int'>, extra(<class 'str'>)), extra(<class 'str'>), None), '?cidr': extra(<class 'str'>), '?host_routes': [tuple_((extra(<class 'str'>), extra(<class 'str'>)))]}]*)
Update multiple subnets

vlcp.service.sdn.vrouterapi

Note: This document is generated from the source file.

View Source on GitHub

class vlcp.service.sdn.vrouterapi.**VRouterApi**(*server*)
Standard network model for L3 SDN

__init__(*server*)
Constructor

addrouterinterface(*router: str*, *subnet: str*, *id: (<class 'str'>, None) = None*, ***kwargs*)
Connect virtual router to a subnet

Parameters

- **router** – virtual router ID
- **subnet** – subnet ID
- **id** – router port ID
- ****kwargs** – customized options

Returns A dictionary of information of the created router port

addrouterinterfaces(*interfaces: [{'router': <class 'str'>, 'subnet': <class 'str'>, '?id': <class 'str'>, '?ip_address': extra(<class 'str'>)}]*)
Create multiple router interfaces

createvirtualrouter(*id: (<class 'str'>, None) = None*, ***kwargs*)
Create a virtual router

Parameters

- **id** – Virtual router id. If omitted, an UUID is generated.

- ****kwargs** – extra attributes for creation.

Returns A dictionary of information of the virtual router.

createvirtualrouters(*routers: [{'?id': <class 'str'>, '?routes': [tuple_((extra(<class 'str'>), extra(<class 'str'>)))]}]*)
Create multiple virtual routers in a transaction

deletevirtualrouter(*id: str*)
Delete virtual router

deletevirtualrouters(*routers: [{'id': <class 'str'>}]*)
Delete multiple virtual routers

listrouterinterfaces(*id: str*, ***kwargs*)
Query router ports from a virtual router

Parameters

- **id** – virtual router ID
- ****kwargs** – customized filters on router interfaces

Returns a list of dictionaries each stands for a matched router interface

listvirtualrouters(*id: (<class 'str'>, None) = None*, ***kwargs*)
Query virtual router

Parameters

- **id** – if specified, only return virtual router with this ID
- ****kwargs** – customized filter

Returns a list of dictionaries each stands for a matched virtual router.

load(*container*)
Load module

removerouterinterface(*router: str*, *subnet: str*)
Remote a subnet from the router

Parameters

- **router** – virtual router ID
- **subnet** – subnet ID

Returns `{"status": "OK"}`

removerouterinterfaces(*interfaces: [{'router': <class 'str'>, 'subnet': <class 'str'>}]*)
Remote multiple subnets from routers

updatevirtualrouter(*id: str*, ***kwargs*)
Update virtual router

updatevirtualrouters(*routers: [{'id': <class 'str'>, '?routes': [tuple_((extra(<class 'str'>), extra(<class 'str'>)))]}]*)
Update multiple virtual routers

vlcp.service.sdn.vtepcontroller

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/12/1

author hubo

class vlcp.service.sdn.vtepcontroller.**VtepConnectionSynchronized**(**args*, ***kwargs*)

class vlcp.service.sdn.vtepcontroller.**VtepController**(*server*)

Controll a physical switch which supports OVSDB hardware_vtep protocol.

__init__(*server*)

Constructor

listphysicalports(*physicalswitch=None*)

Get physical ports list from this controller, grouped by physical switch name

Parameters **physicalswitch** – physicalswitch name. Return all switches if is None.

Returns dictionary: {physicalswitch: [physicalports]} e.g. {'ps1': ['port1', 'port2']}

listphysicalswitches(*physicalswitch=None*)

Get physical switch info

Parameters **physicalswitch** – physicalswitch name. Return all switches if is None.

Returns dictionary: {physicalswitch: {key: value}} keys include: management_ips, tunnel_ips, description, switch_fault_status

unbindlogicalswitch(*physicalswitch*, *physicalport*, *vlanid*, *logicalnetwork*)

Remove bind of a physical port

Parameters

- **physicalswitch** – physical switch name, should be the name in PhysicalSwitch table of OVSDB vtep database
- **physicalport** – physical port name, should be the name in OVSDB vtep database
- **vlanid** – the vlan tag used for this logicalswitch
- **logicalnetwork** – the logical network id, will also be the logical switch id

unbindphysicalport(*physicalswitch*, *physicalport*)

Remove all bindings for a physical port

Parameters

- **physicalswitch** – physical switch name, should be the name in PhysicalSwitch table of OVSDB vtep database
- **physicalport** – physical port name, should be the name in OVSDB vtep database

updatelogicalswitch(*physicalswitch*, *physicalport*, *vlanid*, *logicalnetwork*, *vni*, *logicalports*)

Bind VLAN on physicalport to specified logical network, and update logical port vxlan info

Parameters

- **physicalswitch** – physical switch name, should be the name in PhysicalSwitch table of OVSDB vtep database
- **physicalport** – physical port name, should be the name in OVSDB vtep database

- **vlanid** – the vlan tag used for this logicalswitch
- **logicalnetwork** – the logical network id, will also be the logical switch id
- **vni** – the VXLAN VNI of the logical network
- **logicalports** – a list of logical port IDs. The VXLAN info of these ports will be updated.

class vlcp.service.sdn.vtepcontroller.**VtepPhysicalSwitchStateChanged**(**args*, ***kwargs*)

class vlcp.service.sdn.vtepcontroller.**_DataUpdateEvent**(**args*, ***kwargs*)

vlcp.service.sdn.vxlancast

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/5/30

author hubo

class vlcp.service.sdn.vxlancast.**VXLANCast**(*server*)

VXLAN single-cast and broadcast functions

__init__(*server*)

Constructor

class vlcp.service.sdn.vxlancast.**VXLANGroupChanged**(**args*, ***kwargs*)

class vlcp.service.sdn.vxlancast.**VXLANUpdater**(*connection*, *parent*)

__init__(*connection*, *parent*)

Retrieve data objects from ObjectDB and use them to generate flows

The module starts ObjectDB.walk from initial keys and walkers. After the walk completes, the retrieved data objects are used by *updateflow()* to generate flows and send them to the OpenFlow connection. When the retrieved objects are updated, FlowUpdater either restart the walk process (re-walk) or directly call another *updateflow()*, according to the objects that are updated.

A subclass should re-initialize *self._initialkeys* and *self._walkerdict* before *main()* coroutine starts to customize the process.

updateflow() is guaranteed for no re-entrance i.e. will not be called until the last call returns. Multiple changes may be merged into the same call.

Parameters

- **connection** – OpenFlow connection
- **initialkeys** – DEPRECATED The key list that triggers a re-walk
- **requestid** – request id to retrieve data objects from ObjectDB
- **logger** – inherit a logger from a module

main()

Main coroutine

reset_initialkeys(*keys*, *values*)
Callback after walk complete, can be used to update *self._initialkeys*.

updateflow(*conn*, *addvalues*, *removevalues*, *updatedvalues*)
Update flow callback. When data objects are updated (either by a re-walk or by a direct update), this method is called with the modification, after the last *updateflow()* call ends.

wait_for_group(*container*, *networkid*, *timeout=120*)
Wait for a VXLAN group to be created

vlcp.service.sdn.vxlanvtep

Note: This document is generated from the source file.

View Source on GitHub

class vlcp.service.sdn.vxlanvtep.**VXLANHandler**(*connection*, *parent*)

__init__(*connection*, *parent*)
Create the routine container.

Parameters

- **scheduler** – The scheduler. This must be set; if None is used, it must be set with *container.bind(scheduler)* before using.
- **daemon** – If daemon = True, the main routine *container.main* is set to be a daemon routine. A daemon routine does not stop the scheduler from quitting; if all non-daemon routines are quit, the scheduler stops.

action_handler()
Call vtep controller in sequence, merge mutiple calls if possible

When a bind relationship is updated, we always send all logical ports to a logicalswitch, to make sure it recovers from some failed updates (so called idempotency). When multiple calls are pending, we only need to send the last of them.

main()
The main routine method, should be rewritten to an async method

class vlcp.service.sdn.vxlanvtep.**VXLANMapChanged**(**args*, ***kwargs*)

class vlcp.service.sdn.vxlanvtep.**VXLANVtep**(*server*)
Use hardware_vtep instead of software VXLAN

__init__(*server*)
Constructor

get_vxlan_bind_info(*systemid=None*)
get vxlan -> vlan , bind info

class vlcp.service.sdn.vxlanvtep.**VtepControllerCall**(**args*, ***kwargs*)

5.6.6 vlcp.service.utils

Note: This document is generated from the source file.

View Source on GitHub

vlcp.service.utils.autoload

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/5/27

author hubo

class vlcp.service.utils.autoload.**AutoLoad**(*server*)
Auto load some modules from a package. Usually used to load network plugins.

__init__(*server*)
Constructor

load(*container*)
Load module

unload(*container*, *force=False*)
Unload module

vlcp.service.utils.knowledge

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/11/9

author hubo

class vlcp.service.utils.knowledge.**Knowledge**(*server*)
Simple KV-cache in memory. A base for other KV-DB. Data is automatically removed after timeout. Use knowledge instead of local storage in modules so data is not lost on module restarting.

__init__(*server*)
Constructor

delete(*key*)
Delete a key

get(*key*, *timeout=None*)
Get value from key

mget(*keys*)
Get multiple values from multiple keys

mgetwithcache(*keys*, *cache=None*)
Get multiple values, cached when possible

mset(*kvpairs*, *timeout=None*)
Set multiple values on multiple keys

mupdate(*keys*, *updater*, *timeout=None*)
Update multiple keys in-place one by one with a custom function, see update. Either all success, or all fail.

set(*key*, *value*, *timeout=None*)
Set value to key, with an optional timeout

update(*key*, *updater*, *timeout=None*)
Update in-place with a custom function

Parameters

- **key** – key to update
- **updater** – `func(k,v)`, should return a new value to update, or return None to delete
- **timeout** – new timeout

Returns the updated value, or None if deleted

updateall(*keys*, *updater*, *timeout=None*)
Update multiple keys in-place, with a function `updater(keys, values)` which returns `(updated_keys, updated_values)`. Either all success or all fail

updateallwithtime(*keys*, *updater*, *timeout=None*)
Update multiple keys in-place, with a function `updater(keys, values, timestamp)` which returns `(updated_keys, updated_values)`. Either all success or all fail.

Timestamp is a integer standing for current time in microseconds.

class vlcp.service.utils.knowledge.**MemoryStorage**(*server*)

vlcp.service.utils.knowledge.**escape_key**(*k*)
Escape k, ensuring there is not a '.' in the string.

vlcp.service.utils.knowledge.**return_self_updater**(*func*)
Run func, but still return v. Useful for using knowledge.update with operates like append, extend, etc. e.g. return_self(lambda k,v: v.append('newobj'))

vlcp.service.utils.knowledge.**unescape_key**(*k*)
Unescape key to get '.' back

vlcp.service.utils.remoteapi

Note: This document is generated from the source file.

View Source on GitHub

class vlcp.service.utils.remoteapi.**RemoteCall**(*server*)
Route local API calls to remote management API.

__init__(*server*)
Constructor

call(*remote_module*, *method*, *timeout*, *params*)
Call remote API

Parameters

- **remote_module** – target name for the remote module
- **method** – method name of the API
- **timeout** – timeout for the call
- **params** – A dictionary contains all the parameters need for the call

Returns Return result from the remote call

vlcp.service.utils.session

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/11/9

author hubo

class vlcp.service.utils.session.**Session**(*server*)

HTTP Session with cookies

__init__(*server*)

Constructor

create()

Create a new session object

Returns Session handle for the created session object.

destroy(*sessionid*)

Destroy a session

Parameters **sessionid** – session ID

Returns a list of Set-Cookie headers to be sent to the client

get(*sessionid*, *refresh=True*)

Get the seesion object of the session id

Parameters

- **sessionid** – a session ID
- **refresh** – if True, refresh the expiration time of this session

Returns Session object or None if not exists

start(*cookies*, *cookieopts=None*)

Session start operation. First check among the cookies to find existed sessions; if there is not an existed session, create a new one.

Parameters

- **cookies** – cookie header from the client
- **cookieopts** – extra options used when creating a new cookie

Returns `(session_handle, cookies)` where session_handle is a SessionHandle object, and cookies is a list of created Set-Cookie headers (may be empty)

5.6.7 vlcp.service.web

Note: This document is generated from the source file.

View Source on GitHub

vlcp.service.web.static

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/11/18

author hubo

class vlcp.service.web.static.**Static**(*server*)
Map specified path to local files

__init__(*server*)
Constructor

updateconfig()
Reload configurations, remove non-exist servers, add new servers, and leave others unchanged

5.7 vlcp.utils

Note: This document is generated from the source file.

View Source on GitHub

5.7.1 vlcp.utils.connector

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/11/30

author hubo

Process events multi-threaded or multi-processed

class vlcp.utils.connector.**Connector**(*worker_start*, *matchers=()*, *scheduler=None*, *mp=True*, *inputlimit=0*, *allowcontrol=True*)

__init__(*worker_start*, *matchers=()*, *scheduler=None*, *mp=True*, *inputlimit=0*, *allowcontrol=True*)

Parameters

- **worker_start** – func(queuein, queueout), queuein is the input queue, queueout is the output queue. For queuein, each object is (event, matcher) tuple; For queueout, each object is a tuple of events to send. Every object in queuein must have a response in queueout.
- **matcheres** – match events to be processed by connector.
- **scheduler** – bind to specified scheduler
- **mp** – use multiprocessing if possible. For windows, multi-threading is always used.
- **inputlimit** – input queue size limit. 0 = infinite.
- **allowcontrol** – if True, the connector accepts ConnectorControlEvent for connector configuration.

main()
The main routine method, should be rewritten to an async method

class vlcp.utils.connector.**ConnectorControlEvent**(**args*, ***kwargs*)

class vlcp.utils.connector.**MoreResultEvent**(**args*, ***kwargs*)

class vlcp.utils.connector.**Resolver**(*scheduler=None*, *poolsize=256*)

__init__(*scheduler=None*, *poolsize=256*)

Parameters

- **worker_start** – func(queuein, queueout), queuein is the input queue, queueout is the output queue. For queuein, each object is (event, matcher) tuple; For queueout, each object is a tuple of events to send. Every object in queuein must have a response in queueout.
- **matcheres** – match events to be processed by connector.
- **scheduler** – bind to specified scheduler
- **mp** – use multiprocessing if possible. For windows, multi-threading is always used.
- **inputlimit** – input queue size limit. 0 = infinite.
- **allowcontrol** – if True, the connector accepts ConnectorControlEvent for connector configuration.

class vlcp.utils.connector.**TaskDoneEvent**(**args*, ***kwargs*)

class vlcp.utils.connector.**TaskEvent**(**args*, ***kwargs*)

class vlcp.utils.connector.**TaskPool**(*scheduler=None*, *poolsize=64*)
Thread pool for small tasks

__init__(*scheduler=None*, *poolsize=64*)

Parameters

- **worker_start** – func(queuein, queueout), queuein is the input queue, queueout is the output queue. For queuein, each object is (event, matcher) tuple; For queueout, each object is a tuple of events to send. Every object in queuein must have a response in queueout.
- **matcheres** – match events to be processed by connector.
- **scheduler** – bind to specified scheduler
- **mp** – use multiprocessing if possible. For windows, multi-threading is always used.

- **inputlimit** – input queue size limit. 0 = infinite.
- **allowcontrol** – if True, the connector accepts ConnectorControlEvent for connector configuration.

runAsyncTask(*container*, *asynctask*, *newthread=True*)
Run asynctask(sender) in task pool, call sender(events) to send customized events, return result

runGenTask(*container*, *gentask*, *newthread=True*)
Run generator gentask() in task pool, yield customized events

runTask(*container*, *task*, *newthread=False*)
Run task() in task pool. Raise an exception or return the return value

run_async_task(*container*, *asynctask*, *newthread=True*)
Run asynctask(sender) in task pool, call sender(events) to send customized events, return result

run_gen_task(*container*, *gentask*, *newthread=True*)
Run generator gentask() in task pool, yield customized events

run_task(*container*, *task*, *newthread=False*)
Run task() in task pool. Raise an exception or return the return value

5.7.2 vlcp.utils.dataobject

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/3/25

author think

exception vlcp.utils.dataobject.**AlreadyExistsException**

class vlcp.utils.dataobject.**DataObject**(*prefix=None*, *deleted=False*)
A base class to serialize data into KVDB

__eq__(*obj*)
Return self==value.

__hash__()
Return hash(self).

__init__(*prefix=None*, *deleted=False*)
Initialize self. See help(type(self)) for accurate signature.

__ne__(*obj*)
Return self!=value.

__repr__(**args*, ***kwargs*)
Return repr(self).

class vlcp.utils.dataobject.**DataObjectSet**
A set of data objects, usually of a same type. Allow weak references only.

__eq__(*obj*)
Return self==value.

__hash__()
Return hash(self).

__init__()
Initialize self. See help(type(self)) for accurate signature.

__ne__(*obj*)
Return self!=value.

__repr__(**args*, ***kwargs*)
Return repr(self).

class vlcp.utils.dataobject.**DataObjectUpdateEvent**(**args*, ***kwargs*)

class vlcp.utils.dataobject.**MultiKeyReference**(*prefix=None*, *deleted=False*)

class vlcp.utils.dataobject.**MultiKeySet**(*prefix=None*, *deleted=False*)

__init__(*prefix=None*, *deleted=False*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.dataobject.**ReferenceObject**(*key*, *refobj=None*)
A strong reference. The referenced object should be automatically retrieved from KVDB.

__eq__(*obj*)
Return self==value.

__hash__(**args*, ***kwargs*)
Return hash(self).

__init__(*key*, *refobj=None*)
Initialize self. See help(type(self)) for accurate signature.

__ne__(*obj*)
Return self!=value.

__repr__(**args*, ***kwargs*)
Return repr(self).

__str__(**args*, ***kwargs*)
Return str(self).

class vlcp.utils.dataobject.**UniqueKeyReference**(*prefix=None*, *deleted=False*)

class vlcp.utils.dataobject.**UniqueKeySet**(*prefix=None*, *deleted=False*)

__init__(*prefix=None*, *deleted=False*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.dataobject.**WeakReferenceObject**(*key*)
A weak reference. The referenced object must be retrieved manually.

__eq__(*obj*)
Return self==value.

__hash__(**args*, ***kwargs*)
Return hash(self).

__init__(*key*)
Initialize self. See help(type(self)) for accurate signature.

__ne__(*obj*)
Return self!=value.

__repr__(**args*, ***kwargs*)
Return repr(self).

__str__(**args*, ***kwargs*)
Return str(self).

vlcp.utils.dataobject.**create_from_key**(*cls*, *oldvalue*, *key*)
Raise if the old value already exists

vlcp.utils.dataobject.**create_new**(*cls*, *oldvalue*, **args*)
Raise if the old value already exists

vlcp.utils.dataobject.**dump**(*obj*, *attributes=True*, *_refset=None*)
Show full value of a data object

vlcp.utils.dataobject.**list_updater**(**args*)
Decorate a function with named lists into updater for transact.

Params *args parameter list sizes. -1 means all other items. None means a single item instead of a list. only one -1 is allowed.

vlcp.utils.dataobject.**updater**(*f*)
Decorate a function with named arguments into updater for transact

vlcp.utils.dataobject.**watch_context**(*keys*, *result*, *reqid*, *container*, *module='objectdb'*)
DEPRECATED - use request_context for most use cases

5.7.3 vlcp.utils.dhcp

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/6/22

author hubo

See https://tools.ietf.org/html/rfc2131 https://tools.ietf.org/html/rfc2132

vlcp.utils.dhcp.**build_options**(*payload*, *options*, *maxsize=576*, *overload=3*, *allowpartial=True*)
Split a list of options

This is the reverse operation of *reassemble_options*, it splits *dhcp_option* into *dhcp_option_partial* if necessary, and set overload option if field overloading is used.

Parameters

- **options** – a list of *dhcp_option*
- **maxsize** – Limit the maximum DHCP message size. If options cannot fit into the DHCP message, specified fields are overloaded for options. If options cannot fit after overloading, extra options are DROPPED if allowpartial = True.

  It is important to sort the dhcp options by priority.
- **overload** – fields that are allowed to be overloaded
- **allowpartial** – When options cannot fit into the DHCP message, allow the rest options to be dropped.

Returns Number of options that are dropped i.e. *options[:-return_value]* are dropped

vlcp.utils.dhcp.**create_dhcp_options**(*input_dict*, *ignoreError=False*, *generateNone=False*)
Try best to create dhcp_options from human friendly values, ignoring invalid values

vlcp.utils.dhcp.**create_option_from_value**(*tag*, *value*)
Set DHCP option with human friendly value

vlcp.utils.dhcp.**reassemble_options**(*payload*)
Reassemble partial options to options, returns a list of dhcp_option

DHCP options are basically |*tag*|*length*|*value*| structure. When an option is longer than 255 bytes, it can be splitted into multiple structures with the same tag. The splitted structures must be joined back to get the original option.

dhcp_option_partial is used to present the splitted options, and *dhcp_option* is used for reassembled option.

5.7.4 vlcp.utils.encoders

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/11/26

author hubo

5.7.5 vlcp.utils.ethernet

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/7/30

author hubo

vlcp.utils.ethernet.**ip_frag**(*packet*)
Not fragments: ip_frag(packet) == 0 not ip_frag(packet)
First packet of fragments: ip_frag(packet) == IP_FRAG_ANY
Not first packet of fragments: ip_frag(packet) & IP_FRAG_LATER
All fragments: ip_frag(packet) & IP_FRAG_ANY

5.7.6 vlcp.utils.exceptions

Note: This document is generated from the source file.

View Source on GitHub

Created on 2018/7/2

author hubo

exception vlcp.utils.exceptions.**APIRejectedException**

exception vlcp.utils.exceptions.**AsyncTransactionLockException**(*info=None*)

__init__(*info=None*)
Initialize self. See help(type(self)) for accurate signature.

exception vlcp.utils.exceptions.**ModuleAPICallTimeoutException**

exception vlcp.utils.exceptions.**StaleResultException**(*result*, *desc='Result is stale'*)

__init__(*result*, *desc='Result is stale'*)
Initialize self. See help(type(self)) for accurate signature.

exception vlcp.utils.exceptions.**TransactionFailedException**

exception vlcp.utils.exceptions.**TransactionRetryExceededException**

__init__()
Initialize self. See help(type(self)) for accurate signature.

exception vlcp.utils.exceptions.**TransactionTimeoutException**

__init__()
Initialize self. See help(type(self)) for accurate signature.

exception vlcp.utils.exceptions.**WalkKeyNotRetrieved**

5.7.7 vlcp.utils.flowupdater

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/4/11

author hubo

class vlcp.utils.flowupdater.**FlowUpdater**(*connection*, *initialkeys=()*, *requestid=None*, *logger=None*)

__init__(*connection*, *initialkeys=()*, *requestid=None*, *logger=None*)
Retrieve data objects from ObjectDB and use them to generate flows

The module starts ObjectDB.walk from initial keys and walkers. After the walk completes, the retrieved data objects are used by *updateflow()* to generate flows and send them to the OpenFlow connection. When the retrieved objects are updated, FlowUpdater either restart the walk process (re-walk) or directly call another *updateflow()*, according to the objects that are updated.

A subclass should re-initialize *self._initialkeys* and *self._walkerdict* before *main()* coroutine starts to customize the process.

updateflow() is guaranteed for no re-entrance i.e. will not be called until the last call returns. Multiple changes may be merged into the same call.

Parameters

- **connection** – OpenFlow connection
- **initialkeys** – DEPRECATED The key list that triggers a re-walk
- **requestid** – request id to retrieve data objects from ObjectDB

- **logger** – inherit a logger from a module

main()
Main coroutine

reset_initialkeys(*keys*, *values*)
Callback after walk complete, can be used to update *self._initialkeys*.

restart_walk()
Force a re-walk

shouldupdate(*newvalues*, *updatedvalues*)
Callback when upate. Rewrite this method to ignore some updates.

If this callback returns False, the update is ignored.

updateflow(*connection*, *addvalues*, *removevalues*, *updatedvalues*)
Update flow callback. When data objects are updated (either by a re-walk or by a direct update), this method is called with the modification, after the last *updateflow()* call ends.

updateobjects(*updatedvalues*)
Force a update notification on specified objects, even if they are not actually updated in ObjectDB

walkcomplete(*keys*, *values*)
Async callback after walk complete, before flow update

class vlcp.utils.flowupdater.**FlowUpdaterNotification**(**args*, ***kwargs*)
These notifications are edge triggered - they work with identifiers and set

5.7.8 vlcp.utils.gzipheader

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/11/26

author hubo

5.7.9 vlcp.utils.http

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/11/10

author hubo

class vlcp.utils.http.**Dispatcher**(*scheduler=None*, *daemon=False*, *vhost=''*)

__init__(*scheduler=None*, *daemon=False*, *vhost=''*)
Initialize self. See help(type(self)) for accurate signature.

classmethod expand(*match*, *expand*)
If use expand directly, the url-decoded context will be decoded again, which create a security issue. Hack expand to quote the text before expanding

redirect(*path, expand, status=302, host=None, vhost=None, method=[b'GET', b'HEAD'], keepquery=True*)
Automatically redirect a request to another location

rewrite(*path, expand, newmethod=None, host=None, vhost=None, method=[b'GET', b'HEAD'], keepquery=True*)
Automatically rewrite a request to another location

route(*path, routinemethod, container=None, host=None, vhost=None, method=[b'GET', b'HEAD']*)
Route specified path to a WSGI-styled routine factory

Parameters

- **path** – path to match, can be a regular expression
- **routinemethod** – factory function routinemethod(env), env is an Environment object see also utils.http.Environment
- **container** – routine container
- **host** – if specified, only response to request to specified host
- **vhost** – if specified, only response to request to specified vhost. If not specified, response to dispatcher default vhost.
- **method** – if specified, response to specified methods

routeargs(*path, routinemethod, container=None, host=None, vhost=None, method=[b'POST'], tostr=True, matchargs=(), fileargs=(), queryargs=(), cookieargs=(), sessionargs=(), csrfcheck=False, csrfarg='_csrf', formlimit=67108864*)
Convenient way to route a processor with arguments. Automatically parse arguments and pass them to the corresponding handler arguments. If required arguments are missing, HttpInputException is thrown which creates a 400 Bad Request response. If optional arguments are missing, they are replaced with default values just as normal Python call does. If handler accepts keyword arguments, extra arguments are sent with kwargs. If not, they are safely ignored.

Parameters

- **path** – path to match, can be a regular expression
- **routinemethod** – factory function routinemethod(env, arga, argb, argc...). env is an Environment object. form or querystring arguments 'arga', 'argb', 'argc' are passed to arga, argb, argc.
- **container** – routine container
- **host** – if specified, only response to request to specified host
- **vhost** – if specified, only response to request to specified vhost. If not specified, response to dispatcher default vhost.
- **method** – methods allowed. With POST method, arguments are extracted from form by default; With GET or HEAD method, arguments are extracted from querystring(args).
- **tostr** – In Python3, convert bytes to str before sending arguments to handler.
- **matchargs** – Instead of using form or args, extract arguments from path match. matchargs is a sequence of matcher group names. If specified a group name by

number, the argument is used as positional arguments; if specified a group name by name(str), the argument is used as a keyword argument.

- **fileargs** – Instead of using form or args, extract specified arguments from files.
- **queryargs** – Instead of using form, extract specified arguments from args. Notice that when GET is allowed, the arguments are always extracted from args by default.
- **cookieargs** – Instead of using form or args, extract specified arguments from cookies.
- **sessionargs** – Instead of using form or args, extract specified arguments from session. Notice that if sessionargs is not empty, env.sessionstart() is called, so vlcp.service.utils.session.Session module must be loaded.
- **csrfcheck** – If True, check <csrfarg> in input arguments against <csrfarg> in session. Notice that csrfcheck=True cause env.sessionstart() to be called, so vlcp.service.utils.session.Session module must be loaded.
- **csrfarg** – argument name to check, default to "_csrf"
- **formlimit** – limit on parseform, default to 64MB. None to no limit.

For example, if using:

```
async def handler(env, target, arga, argb, argc):
    ...

dispatcher.routeargs(b'/do/(.*)', handler, matchargs=(1,), queryargs=('argc
↪'))
```

And there is a HTTP POST:

```
POST /do/mytarget?argc=1 HTTP/1.1
Host: ...
...

arga=test&argb=test2
```

then handler accepts arguments: target="mytarget", arga="test", argb="test2", argc="1"

routeevent(*path, routinemethod, container=None, host=None, vhost=None, method=[b'GET', b'HEAD']*)

Route specified path to a routine factory

Parameters

- **path** – path to match, can be a regular expression
- **routinemethod** – factory function routinemethod(event), event is the HttpRequestEvent
- **container** – routine container. If None, default to self for bound method, or event.connection if not
- **host** – if specified, only response to request to specified host
- **vhost** – if specified, only response to request to specified vhost. If not specified, response to dispatcher default vhost.
- **method** – if specified, response to specified methods

class vlcp.utils.http.**Environment**(*event*, *container=None*, *defaultencoding='utf-8'*)
Environment object used in HTTP handlers

__init__(*event*, *container=None*, *defaultencoding='utf-8'*)
Initialize self. See help(type(self)) for accurate signature.

__repr__(**args*, ***kwargs*)
Return repr(self).

argstostr()
Query string arguments are bytes in Python3. This function Convert bytes to string with env.encoding(default to utf-8).

basicauth(*realm=b'all'*, *nofail=False*)
Try to get the basic authorize info, return (username, password) if succeeded, return 401 otherwise

basicauthfail(*realm=b'all'*)
Return 401 for authentication failure. This will end the handler.

bufferoutput()
Buffer the whole output until write EOF or flushed.

close()
Close this request, send all data. You can still run other operations in the handler.

cookietostr()
Cookie values are bytes in Python3. This function Convert bytes to string with env.encoding(default to utf-8).

createcsrf(*csrfarg='_csrf'*)
Create a anti-CSRF token in the session

error(*status=500*, *allowredirect=True*, *close=True*, *showerror=None*, *headers=[]*)
Show default error response

escape(*text*, *quote=True*)
Escape special characters in HTML

exit(*output=b''*)
Exit current HTTP processing

flush(*eof=False*)
Flush the current output stream buffer

getrealpath(*root*, *path*)
Return the real path on disk from the query path, from a root path. The input path from URL might be absolute '/abc', or point to parent '../test', or even with UNC or drive 'testbc', 'c: est.abc', which creates security issues when accessing file contents with the path. With getrealpath, these paths cannot point to files beyond the root path.

Parameters

- **root** – root path of disk files, any query is limited in root directory.
- **path** – query path from URL.

header(*key*, *value*, *replace=True*)
Send a new header

nl2br(*text*)

Replace '

' with '
n'

output(*stream*, *disabletransferencoding=None*)
Set output stream and send response immediately

outputdata(*data*)
Send output with fixed length data

outputjson(*obj*)
Serialize *obj* with JSON and output to the client

parseform(*limit=67108864*, *tostr=True*, *safename=True*)
Parse form-data with multipart/form-data or application/x-www-form-urlencoded In Python3, the keys of form and files are unicode, but values are bytes If the key ends with '[]', it is considered to be a list: a=1&b=2&b=3 => {'a':1,'b':3} a[]=1&b[]=2&b[]=3 => {'a':[1],'b':[2,3]} :param limit: limit total input size, default to 64MB. None = no limit. Note that all the form data is stored in memory (including upload files), so it is dangerous to accept a very large input. :param tostr: convert values to str in Python3. Only apply to form, files data are always bytes :param safename: if True, extra security checks are performed on filenames to reduce known security risks.

rawheader(*kv*, *replace=True*)
Add a header with "<Header>: Value" string

redirect(*path*, *status=302*)
Redirect this request with 3xx status

rewrite(*path*, *method=None*, *keepresponse=True*)
Rewrite this request to another processor. Must be called before header sent

sessiondestroy()
Destroy current session. The session object is discarded and can no longer be used in other requests.

sessionstart()
Start session. Must start service.utils.session.Session to use this method

setcookie(*key*, *value*, *max_age=None*, *expires=None*, *path='/'*, *domain=None*, *secure=None*, *httponly=False*)
Add a new cookie

startResponse(*status=200*, *headers=[]*, *clearheaders=True*, *disabletransferencoding=False*)
Start to send response

start_response(*status=200*, *headers=[]*, *clearheaders=True*, *disabletransferencoding=False*)
Start to send response

write(*data*, *eof=False*, *buffering=True*)
Write output to current output stream

writelines(*lines*, *eof=False*, *buffering=True*)
Write lines to current output stream

exception vlcp.utils.http.**HttpExitException**

class vlcp.utils.http.**HttpHandler**(*scheduler=None*, *daemon=False*, *vhost=''*)

__init__(*scheduler=None*, *daemon=False*, *vhost=''*)
Create the routine container.

Parameters

- **scheduler** – The scheduler. This must be set; if None is used, it must be set with *container.bind(scheduler)* before using.

- **daemon** – If daemon = True, the main routine *container.main* is set to be a daemon routine. A daemon routine does not stop the scheduler from quitting; if all non-daemon routines are quit, the scheduler stops.

close()
Same as *terminate()*

static routeargs(*path, host=None, vhost=None, method=[b'POST'], **kwargs*)
For extra arguments, see Dispatcher.routeargs. They must be specified by keyword arguments

start(*asyncStart=False*)
Start *container.main* as the main routine.

Parameters **asyncStart** – if True, start the routine in background. By default, the routine starts in foreground, which means it is executed to the first *yield* statement before returning. If the started routine raises an exception, the exception is re-raised to the caller of *start*

exception vlcp.utils.http.**HttpInputException**

exception vlcp.utils.http.**HttpRewriteLoopException**

vlcp.utils.http.**http**(*container=None*)
wrap a WSGI-style class method to a HTTPRequest event handler

vlcp.utils.http.**statichttp**(*container=None*)
wrap a WSGI-style function to a HTTPRequest event handler

5.7.10 vlcp.utils.indexedheap

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/10/18

author hubo

class vlcp.utils.indexedheap.**IndexedHeap**
A heap with indices

__init__()
Initialize self. See help(type(self)) for accurate signature.

5.7.11 vlcp.utils.jsonencoder

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/3/14

author hubo

class vlcp.utils.jsonencoder.**JsonFormat**
This is an extended JSON formatter used by WebAPI module

5.7.12 vlcp.utils.kvcache

Note: This document is generated from the source file.

View Source on GitHub

Created on 2018/4/17

author hubo

5.7.13 vlcp.utils.logger

Note: This document is generated from the source file.

View Source on GitHub

class vlcp.utils.logger.**ContextAdapter**(*logger*, *extra*)

process(*msg*, *kwargs*)
Process the logging message and keyword arguments passed in to a logging call to insert contextual information. You can either manipulate the message itself, the keyword args or both. Return the message and kwargs modified (or not) to suit your needs.

Normally, you'll only need to override this one method in a LoggerAdapter subclass for your specific needs.

5.7.14 vlcp.utils.netutils

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/7/26

author think

5.7.15 vlcp.utils.networkmodel

Note: This document is generated from the source file.

View Source on GitHub

class vlcp.utils.networkmodel.**DVRouterExternalAddressInfo**(*prefix=None*, *deleted=None*)

__init__(*prefix=None*, *deleted=None*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**DVRouterForwardInfo**(*prefix=None*, *deleted=None*)

__init__(*prefix=None*, *deleted=None*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**DVRouterForwardInfoRef**(*prefix=None*, *deleted=None*)

__init__(*prefix=None*, *deleted=None*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**DVRouterForwardSet**(*prefix=None*, *deleted=None*)

__init__(*prefix=None*, *deleted=None*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**LogicalNetwork**(*prefix=None*, *deleted=False*)

class vlcp.utils.networkmodel.**LogicalNetworkMap**(*prefix=None*, *deleted=None*)

__init__(*prefix=None*, *deleted=None*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**LogicalNetworkSet**(*prefix=None*, *deleted=None*)

__init__(*prefix=None*, *deleted=None*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**LogicalPort**(*prefix=None*, *deleted=False*)

class vlcp.utils.networkmodel.**LogicalPortSet**(*prefix=None*, *deleted=None*)

__init__(*prefix=None*, *deleted=None*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**LogicalPortVXLANInfo**(*prefix=None*, *deleted=False*)

__init__(*prefix=None*, *deleted=False*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**PhysicalNetwork**(*prefix=None*, *deleted=False*)

class vlcp.utils.networkmodel.**PhysicalNetworkMap**(*prefix=None*, *deleted=False*)

__init__(*prefix=None*, *deleted=False*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**PhysicalNetworkSet**(*prefix=None*, *deleted=None*)

__init__(*prefix=None*, *deleted=None*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**PhysicalPort**(*prefix=None*, *deleted=False*)

class vlcp.utils.networkmodel.**PhysicalPortSet**(*prefix=None*, *deleted=False*)

__init__(*prefix=None*, *deleted=False*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**RouterPort**(*prefix=None*, *deleted=False*)

class vlcp.utils.networkmodel.**SubNet**(*prefix=None*, *deleted=False*)

class vlcp.utils.networkmodel.**SubNetMap**(*prefix=None*, *deleted=None*)

__init__(*prefix=None*, *deleted=None*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**SubNetSet**(*prefix=None*, *deleted=None*)

__init__(*prefix=None*, *deleted=None*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**VRouter**(*prefix=None*, *deleted=None*)

__init__(*prefix=None*, *deleted=None*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**VRouterSet**(*prefix=None*, *deleted=None*)

__init__(*prefix=None*, *deleted=None*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.networkmodel.**VXLANEndpointSet**(*prefix=None*, *deleted=None*)

__init__(*prefix=None*, *deleted=None*)
Initialize self. See help(type(self)) for accurate signature.

5.7.16 vlcp.utils.networkplugin

Note: This document is generated from the source file.

View Source on GitHub

Created on 2018/7/2

author hubo

Base walkers for network plugins

vlcp.utils.networkplugin.**createlogicalnetwork**(*create_processor=functools.partial(<function default_processor>, excluding=('id', 'physicalnetwork'))*, *reorder_dict=<function default_iterate_dict>*)

Parameters **create_processor** – create_processor(logicalnetwork, logicalnetworkmap, physicalnetwork, physicalnetworkmap, walk, write, *, parameters)

vlcp.utils.networkplugin.**createphysicalnetwork**(*type*, *create_processor=functools.partial(<function default_processor>, excluding=('id', 'type'))*, *reorder_dict=<function default_iterate_dict>*)

Parameters

- **type** – physical network type
- **create_processor** – create_processor(physicalnetwork, walk, write, *, parameters)

vlcp.utils.networkplugin.**createphysicalport**(*create_processor=functools.partial(<function default_processor>, excluding=('vhost', 'systemid', 'bridge', 'name', 'physicalnetwork'))*, *reorder_dict=<function default_iterate_dict>*)

Parameters **create_processor** – create_processor(physicalport, physicalnetwork, physicalnetworkmap, walk, write, *, parameters)

vlcp.utils.networkplugin.**deletelogicalnetwork**(*check_processor=<function default_logicalnetwork_delete_check>*, *reorder_dict=<function default_iterate_dict>*)

Parameters **check_processor** – check_processor(logicalnetwork, logicalnetworkmap, physicalnetwork, physicalnetworkmap, walk, write, *, parameters)

vlcp.utils.networkplugin.**deletephysicalnetwork**(*check_processor=<function default_physicalnetwork_delete_check>*, *reorder_dict=<function default_iterate_dict>*)

Parameters **check_processor** – check_processor(physicalnetwork, physicalnetworkmap, walk, write, *, parameters)

vlcp.utils.networkplugin.**deletephysicalport**(*check_processor=<function _false_processor>*, *reorder_dict=<function default_iterate_dict>*)

Parameters **check_processor** – check_processor(physicalport, physicalnetwork, physicalnetworkmap, walk, write *, parameters)

vlcp.utils.networkplugin.**updatelogicalnetwork**(*update_processor=functools.partial(<function default_processor>, disabled=('physicalnetwork',), excluding=('id',))*, *reorder_dict=<function default_iterate_dict>*)

Parameters **update_processor** – update_processor(logicalnetwork, walk, write, *, parameters)

vlcp.utils.networkplugin.**updatephysicalnetwork**(*update_processor=functools.partial(<function default_processor>, disabled=('type',))*, *reorder_dict=<function default_iterate_dict>*)

Parameters **update_processor** – update_processor(physicalnetwork, walk, write, *, parameters)

vlcp.utils.networkplugin.**updatephysicalport**(*update_processor=functools.partial(<function default_processor>, disabled=('physicalnetwork',), excluding=('vhost', 'systemid', 'bridge', 'name')), reorder_dict=<function default_iterate_dict>*)

Parameters **update_processor** – update_processor(physcialport, walk, write, *, parameters)

5.7.17 vlcp.utils.ovsdb

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/2/22

OVSDB JSON structure helpers. Write less!

These are just simple wrappers, see https://tools.ietf.org/html/rfc7047#page-28 for details

author hubo

5.7.18 vlcp.utils.pycache

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/10/19

author hubo

Remove pycache files from a module, to ensure a successful reload

5.7.19 vlcp.utils.redisclient

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/1/8

author hubo

class vlcp.utils.redisclient.**RedisClient**(*url=None*, *db=None*, *protocol=None*)

__init__(*url=None*, *db=None*, *protocol=None*)

Redis client to communicate with Redis server. Several connections are created for different functions.

Parameters

- **url** – connectiom url, e.g. 'tcp://localhost/'. If not specified, redisclient.url in configuration is used

- **db** – default database. If not specified, redisclient.db in configuration is used, which defaults to 0.
- **protocol** – use a pre-created protocol instance instead of creating a new instance

execute_command(*container*, **args*)

Execute command on Redis server:

- For (P)SUBSCRIBE/(P)UNSUBSCRIBE, the command is sent to the subscribe connection. It is recommended to use (p)subscribe/(p)unsubscribe method instead of directly call the command
- For BLPOP, BRPOP, BRPOPLPUSH, the command is sent to a separated connection. The connection is recycled after command returns.
- For other commands, the command is sent to the default connection.

get_connection(*container*)

Get an exclusive connection, useful for blocked commands and transactions.

You must call release or shutdown (not recommanded) to return the connection after use.

Parameters **container** – routine container

Returns RedisClientBase object, with some commands same as RedisClient like execute_command, batch_execute, register_script etc.

make_connobj(*container*)

Return an object to be used like a connection. Put the connection-like object in module.connections to make RedisClient shutdown on module unloading.

psubscribe(*container*, **keys*)

Subscribe to specified globs

Parameters

- **container** – routine container
- ***keys** – subscribed globs

Returns list of event matchers for the specified globs

punsubscribe(*container*, **keys*)

Unsubscribe specified globs. Every subscribed glob should be unsubscribed exactly once, even if duplicated subscribed.

Parameters

- **container** – routine container
- ***keys** – subscribed globs

shutdown(*container*, *force=False*)

Shutdown all connections. Exclusive connections created by get_connection will shutdown after release()

subscribe(*container*, **keys*)

Subscribe to specified channels

Parameters

- **container** – routine container
- ***keys** – subscribed channels

Returns list of event matchers for the specified channels

subscribe_state_matcher(*container*, *connected=True*)
Return a matcher to match the subscribe connection status.

Parameters

- **container** – a routine container. NOTICE: this method is not a routine.
- **connected** – if True, the matcher matches connection up. If False, the matcher matches connection down.

Returns an event matcher.

unsubscribe(*container*, **keys*)
Unsubscribe specified channels. Every subscribed key should be unsubscribed exactly once, even if duplicated subscribed.

Parameters

- **container** – routine container
- ***keys** – subscribed channels

class vlcp.utils.redisclient.**RedisClientBase**(*conn=None*, *parent=None*, *protocol=None*)
Connect to Redis server

__init__(*conn=None*, *parent=None*, *protocol=None*)
Initialize self. See help(type(self)) for accurate signature.

batch_execute(*container*, **cmds*, *raise_first_exception=False*)
execute a batch of commands on current connection in pipeline mode

context(*container*, *release=True*, *lockconn=True*)
Use with statement to manage the connection

Params release if True(default), release the connection when leaving with scope

Params lockconn if True(default), do not allow reconnect during with scope; execute commands on a disconnected connection raises Exceptions.

ensure_registerd(*container*, **scripts*)
Ensure that these scripts are cached on the server. Important when using scripts with batch_execute.

Parameters

- **container** – routine container.
- ***scripts** – registered script tuples, return value of register_script

eval_registered(*container*, *registerd_script*, **args*)
eval a registered script. If the script is not cached on the server, it is automatically cached.

execute_command(*container*, **args*)
execute command on current connection

register_script(*container*, *script*)
register a script to this connection.

Returns registered script. This is a tuple (sha1, script). Pass the tuple to eval_registered, ensure_registerd as registerd_script parameter.

release(*container*)
Release the connection, leave it to be reused later.

shutdown(*container*, *force=False*)
Shutdown all connections to Redis server

exception vlcp.utils.redisclient.**RedisConnectionDown**

exception vlcp.utils.redisclient.**RedisConnectionRestarted**

5.7.20 vlcp.utils.typelib

Note: This document is generated from the source file.

View Source on GitHub

Created on 2018/7/13

author hubo

5.7.21 vlcp.utils.vxlandiscover

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/12/15

author hubo

Refactoring from vxlancast module, to share the logic with hardware-vtep

vlcp.utils.vxlandiscover.**get_broadcast_ips**(*vxlan_endpointset*, *local_ip*, *ovsdb_vhost*, *system_id*, *bridge*)

Get all IP addresses that are not local

Parameters

- **vxlan_endpointset** – a VXLANEndpointSet object
- **local_ips** – list of local IP address to exclude with
- **ovsdb_vhost** – identifier, vhost
- **system_id** – identifier, system-id
- **bridge** – identifier, bridge name

Returns *[(ip, ipnum)]* list where IPs are the original string of the IP address, and ipnum are 32-bit numeric IPv4 address.

vlcp.utils.vxlandiscover.**lognet_vxlan_walker**(*prepush=True*)

Return a walker function to retrieve necessary information from ObjectDB

vlcp.utils.vxlandiscover.**update_vxlaninfo**(*container*, *network_ip_dict*, *created_ports*, *removed_ports*, *ovsdb_vhost*, *system_id*, *bridge*, *allowedmigrationtime*, *refreshinterval*)

Do an ObjectDB transact to update all VXLAN informations

Parameters

- **container** – Routine container
- **network_ip_dict** – a {logicalnetwork_id: tunnel_ip} dictionary
- **created_ports** – logical ports to be added, a {logicalport_id: tunnel_ip} dictionary

- **removed_ports** – logical ports to be removed, a {logicalport_id: tunnel_ip} dictionary
- **ovsdb_vhost** – identifier for the bridge, vhost name
- **system_id** – identifier for the bridge, OVSDB systemid
- **bridge** – identifier for the bridge, bridge name
- **allowedmigrationtime** – time allowed for port migration, secondary endpoint info will be removed after this time
- **refreshinterval** – refreshinterval * 2 will be the timeout for network endpoint

5.7.22 vlcp.utils.walkerlib

Note: This document is generated from the source file.

View Source on GitHub

Created on 2018/7/12

author hubo

vlcp.utils.walkerlib.**ensure_keys**(*walk*, **keys*)
Use walk to try to retrieve all keys

5.7.23 vlcp.utils.webclient

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/11/24

author hubo

exception vlcp.utils.webclient.**CertificateException**

exception vlcp.utils.webclient.**ManualRedirectRequired**(*msg*, *response*, *request*, *kwargs*)

__init__(*msg*, *response*, *request*, *kwargs*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.webclient.**WebClient**(*allowcookies=False*, *cookiejar=None*)
Convenient HTTP request processing. Proxy is not supported in current version.

__init__(*allowcookies=False*, *cookiejar=None*)

Parameters

- **allowcookies** – Accept and store cookies, automatically use them on further requests
- **cookiejar** – Provide a customized cookiejar instead of the default CookieJar()

cleanup(*host=None*)
Cleaning disconnected connections

cleanup_task(*container*, *interval=None*)
If this client object is persist for a long time, and you are worrying about memory leak, create a routine with this method: myclient.cleanup_task(mycontainer, 60). But remember that if you have created at lease one task, you must call myclient.endtask() to completely release the webclient object.

manualredirect(*container*, *exc*, *data*, *datagen=None*)
If data is a stream, it cannot be used again on redirect. Catch the ManualRedirectException and call a manual redirect with a new stream.

open(*container*, *request*, *ignorewebexception=False*, *timeout=None*, *datagen=None*, *cafile=None*, *key=None*, *certificate=None*, *followredirect=True*, *autodecompress=False*, *allowcookies=None*)
Open http request with a Request object

Parameters

- **container** – a routine container hosting this routine
- **request** – vlcp.utils.webclient.Request object
- **ignorewebexception** – Do not raise exception on Web errors (4xx, 5xx), return a response normally
- **timeout** – timeout on connection and single http request. When following redirect, new request does not share the old timeout, which means if timeout=2: connect to host: (2s) wait for response: (2s) response is 302, redirect connect to redirected host: (2s) wait for response: (2s) ...
- **datagen** – if the request use a stream as the data parameter, you may provide a routine to generate data for the stream. If the request failed early, this routine is automatically terminated.
- **cafile** – provide a CA file for SSL certification check. If not provided, the SSL connection is NOT verified.
- **key** – provide a key file, for client certification (usually not necessary)
- **certificate** – provide a certificate file, for client certification (usually not necessary)
- **followredirect** – if True (default), automatically follow 3xx redirections
- **autodecompress** – if True, automatically detect Content-Encoding header and decode the body
- **allowcookies** – override default settings to disable the cookies

shutdown()
Shutdown free connections to release resources

urlgetcontent(*container*, *url*, *data=None*, *method=None*, *headers={}*, *tostr=False*, *encoding=None*, *rawurl=False*, **args*, ***kwargs*)
In Python2, bytes = str, so tostr and encoding has no effect. In Python3, bytes are decoded into unicode str with encoding. If encoding is not specified, charset in content-type is used if present, or default to utf-8 if not. See open for available arguments

Parameters **rawurl** – if True, assume the url is already url-encoded, do not encode it again.

urlopen(*container*, *url*, *data=None*, *method=None*, *headers={}*, *rawurl=False*, **args*, ***kwargs*)
Similar to urllib2.urlopen, but: 1. is a routine 2. data can be an instance of vlcp.event.stream.BaseStream, or str/bytes 3. can specify method 4. if datagen is not None, it is a routine which writes to <data>. It is automatically terminated if the connection is down. 5. can also specify key and certificate, for client

certification 6. certificates are verified with CA if provided. If there are keep-alived connections, they are automatically reused. See open for available arguments

Extra argument:

Parameters **rawurl** – if True, assume the url is already url-encoded, do not encode it again.

class vlcp.utils.webclient.**WebClientRequestDoneEvent**(**args*, ***kwargs*)

exception vlcp.utils.webclient.**WebException**

vlcp.utils.webclient.**match_hostname**(*cert*, *hostname*)

Verify that *cert* (in decoded format as returned by SSLSocket.getpeercert()) matches the *hostname*. RFC 2818 and RFC 6125 rules are followed, but IP addresses are not accepted for *hostname*.

CertificateError is raised on failure. On success, the function returns nothing.

5.7.24 vlcp.utils.zkclient

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/9/22

author hubo

class vlcp.utils.zkclient.**ZooKeeperClient**(*container*, *serverlist=None*, *chroot=None*, *protocol=None*, *readonly=False*, *restart_session=True*)

ZooKeeper client to send requests to a cluster

__init__(*container*, *serverlist=None*, *chroot=None*, *protocol=None*, *readonly=False*, *restart_session=True*)

Initialize self. See help(type(self)) for accurate signature.

get_last_watch_zxid()

Return the latest zxid seen from servers

get_last_zxid()

Return the latest zxid seen from servers

requests(*requests*, *container*, *timeout=None*, *session_lock=None*, *callback=None*, *priority=0*)

similar to vlcp.protocol.zookeeper.ZooKeeper.requests, but:

1. Returns an extra item *watchers*, which is a list of objects corresponding to each request. if the request has watch=True, the corresponding object is a RoutineFuture object; if the request has watch=False or does not support watch, the corresponding object is None. Use watcher.wait() to get the watch event. Use watcher.close() to discard the watcher.
2. If the connection is lost during requests, this method waits for reconnecting until timeout, session expires or the response of a request which is not read-only is lost.

Parameters

- **requests** – sequence of request
- **container** – container of current routine
- **timeout** – if not None, wait only for specified time. Notice that it is not an exact limit, it won't stop the execution unless the connection is lost

- **session_lock** – if not None, only execute if the session_id == session_lock
- **callback** – if not None, callback(request, response) is called immediately after any response is received

Returns (result, lost_responses, retry_requests, watchers) tuple, the first three are the same as ZooKeeper.requests, the last item *watchers* is a list of RoutineFuture objects

reset()
Discard current session and start a new one

watch_path(*path*, *watch_type*, *container=None*)
Watch the specified path as specified type

exception vlcp.utils.zkclient.**ZooKeeperIllegalPathException**

class vlcp.utils.zkclient.**ZooKeeperRestoreWatches**(**args*, ***kwargs*)

class vlcp.utils.zkclient.**ZooKeeperSessionStateChanged**(**args*, ***kwargs*)

exception vlcp.utils.zkclient.**ZooKeeperSessionUnavailable**(*state*)

__init__(*state*)
Initialize self. See help(type(self)) for accurate signature.

vlcp.utils.zkclient.**random**() → x in the interval [0, 1).

5.7.25 vlcp.utils.zookeeper

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/8/25

author hubo

class vlcp.utils.zookeeper.**UStringParser**
Jute ustring type.

__init__()
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.zookeeper.**VectorParser**(*innerparser*)
Jute vector type.

__init__(*innerparser*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp.utils.zookeeper.**ustringtype**(*displayname='ustring'*)
A int32 followed by variable length bytes

__init__(*displayname='ustring'*)
Initialize self. See help(type(self)) for accurate signature.

__repr__(**args*, ***kwargs*)
Return repr(self).

parser()
Get parser for this type. Create the parser on first call.

class vlcp.utils.zookeeper.**vector**(*innertype*)
Jute vector

__init__(*innertype*)
Initialize self. See help(type(self)) for accurate signature.

__repr__(**args*, ***kwargs*)
Return repr(self).

5.8 vlcp.start

Note: This document is generated from the source file.

View Source on GitHub

Created on 2015/10/19

author hubo

Command-line entry

vlcp.start.**default_start**()
Use *sys.argv* for starting parameters. This is the entry-point of *vlcp-start*

5.9 vlcp_docker.cleanup

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/11/3

author hubo

class vlcp_docker.cleanup.**Cleanup**(*server*)
Clean up unreleased veth devices, delete unreleased logical ports. Comparing current logical ports with docker API result

cleanup.py -f <configfile> [-H <endpoint>] [–skipovs] [–skipiplink] [–skiplogicalport] [–nodockerinfo]

-H or –host: specify docker API endpoint –skipovs: do not remove invalid ports from OpenvSwitch –skipiplink: do not remove extra veth devices –skiplogicalport: do not remove unreleased logical ports –nodockerinfo: do not detect docker info, always delete logical ports

5.10 vlcp_docker.dockerplugin

Note: This document is generated from the source file.

View Source on GitHub

Created on 2016/8/22

author hubo

class vlcp_docker.dockerplugin.**DockerInfo**(*prefix=None*, *deleted=False*)

class vlcp_docker.dockerplugin.**DockerPlugin**(*server*)
Integrate VLCP with Docker

__init__(*server*)
Constructor

getdockerinfo(*portid*)
Get docker info for specified port

load(*container*)
Load module

class vlcp_docker.dockerplugin.**IPAMPoolReserve**(*prefix=None*, *deleted=False*)

__init__(*prefix=None*, *deleted=False*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp_docker.dockerplugin.**IPAMReserve**(*prefix=None*, *deleted=False*)

__init__(*prefix=None*, *deleted=False*)
Initialize self. See help(type(self)) for accurate signature.

class vlcp_docker.dockerplugin.**IPAMReserveMarker**(*prefix=None*, *deleted=False*)

exception vlcp_docker.dockerplugin.**IPAMUsingException**

class vlcp_docker.dockerplugin.**NetworkPlugin**(*parent*)

__init__(*parent*)
Create the routine container.

Parameters

- **scheduler** – The scheduler. This must be set; if None is used, it must be set with *container.bind(scheduler)* before using.
- **daemon** – If daemon = True, the main routine *container.main* is set to be a daemon routine. A daemon routine does not stop the scheduler from quitting; if all non-daemon routines are quit, the scheduler stops.

exception vlcp_docker.dockerplugin.**RetryUpdateException**

CHAPTER 6

Articles

There are some introducing articles for VLCP or related technologies in Chinese:

VLCP

WebServer——VLCP

namedstruct:

CythonPyPyVLCP

——FutureChannelPub/Sub

VLCPDocker network

No-SQL——VLCPObjectDB

Note: The code examples in them are outdated because they are written for v1.x, but v2.0 changes the basic grammer and moved to Python 3.5+. But, most designs remain the same.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

v

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

S

T

U

V

W

Y

Z