Ryu应用开发------基于跳数/时延/带宽的最短/优路径和负载均衡

对于SDN初学者而言，最短路径转发应用和负载均衡应用是最常见，也是最适合学习的经典应用。根据链路权重参数的不同，主要有基于跳数、时延和带宽的几种最短\最优路径转发应用。根据链路可用带宽实现的最优路径转发本质上也是一种网络流量负载均衡的简单实现。本文将介绍笔者在学习过程中开发的网络感知模块和基于网络感知模块提供的网络信息，实现的基于跳数、时延和带宽三种最优路径转发应用。

基于跳数的最短路径转发

基于跳数的最短路径转发是最简单的最优路径转发应用。我们通过network_awareness应用来实现网络拓扑资源的感知并计算最短路径。首先控制器通过下发LLDP报文来获取网络链路信息，然后再利用网络信息，生成网络拓扑图。网络感知应用使用networkx的有向图数据结构存储拓扑信息，使用networkx提供的shortestsimplepaths函数来计算最短路径。shortest_simple_paths函数支持在图上找出源交换机到目的交换机的K条最短路径，其函数参数信息如下：

```python
shortest_simple_paths(G,source,target,weight=None)
```

在给定图G，源交换机source，目的交换机target以及链路权重类型weight的情况下，会返回一个路径生成器。通过K次调用生成器可以生成K条最短路径。
获得最短路径之后，shortest_forwarding应用将完成流表下发等工作，实现基于跳数的最短路径转发应用。

Python | 复制代码

```python
# Copyright (C) 2016 Li Cheng at Beijing University of Posts
# and Telecommunications. www.muzixing.com
#
# 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.

# conding=utf-8
import logging
import struct
import networkx as nx
from operator import attrgetter
from ryu import cfg
from ryu.base import app_manager
from ryu.controller import ofp_event
from ryu.controller.handler import MAIN_DISPATCHER, DEAD_DISPATCHER
from ryu.controller.handler import CONFIG_DISPATCHER
from ryu.controller.handler import set_ev_cls
from ryu.ofproto import ofproto_v1_3
from ryu.lib.packet import packet
from ryu.lib.packet import ethernet
from ryu.lib.packet import ipv4
from ryu.lib.packet import arp

from ryu.topology import event, switches
from ryu.topology.api import get_switch, get_link

import network_awareness
import network_monitor
import network_delay_detector


CONF = cfg.CONF


class ShortestForwarding(app_manager.RyuApp):
```

```
    """
        ShortestForwarding is a Ryu app for forwarding packets in shortest
        path.
        The shortest path computation is done by module network awareness,
        network monitor and network delay detector.
    """

    OFP_VERSIONS = [ofproto_v1_3.OFP_VERSION]
    _CONTEXTS = {
        "network_awareness": network_awareness.NetworkAwareness,
        "network_monitor": network_monitor.NetworkMonitor,
        "network_delay_detector": network_delay_detector.NetworkDelayDetector}

    WEIGHT_MODEL = {'hop': 'weight', 'delay': "delay", "bw": "bw"}

    def __init__(self, *args, **kwargs):
        super(ShortestForwarding, self).__init__(*args, **kwargs)
        self.name = 'shortest_forwarding'
        self.awareness = kwargs["network_awareness"]
        self.monitor = kwargs["network_monitor"]
        self.delay_detector = kwargs["network_delay_detector"]
        self.datapaths = {}
        self.weight = self.WEIGHT_MODEL[CONF.weight]

    def set_weight_mode(self, weight):
        """
            set weight mode of path calculating.
        """
        self.weight = weight
        if self.weight == self.WEIGHT_MODEL['hop']:
            self.awareness.get_shortest_paths(weight=self.weight)
        return True

    @set_ev_cls(ofp_event.EventOFPStateChange,
                [MAIN_DISPATCHER, DEAD_DISPATCHER])
    def _state_change_handler(self, ev):
        """
            Collect datapath information.
        """
```

基于时延的最优路径转发

基于时延的最优路径转发应用原理和基于跳数的最短路径转发应用类似，只是链路权重类型变成了时延。关于计算链路时延的原理，读者可以阅读Ryu:网络时延探测应用。NetworkDelayDetector

是一个网络时延探测应用，其在获取到链路时延之后，将时延数据存储到Networkx的图数据结构中，以供其他模块使用。
通过设置链路权重参数，Shortest_forwarding应用可以基于时延数据计算最优的转发路径。

基于带宽的最优路径转发/负载均衡

基于带宽的最优路径相比以上两种应用相对要复杂一些。为了降低计算复杂度，我们采用先计算基于跳数的K条最短路径，再从中选取可用带宽最大的那条路径最为最优解。链路可用带宽的数据由nework_monitor应用提供。该应用周期地获取链路的剩余带宽，并将带宽数据存储到networkx的图结构中，提供给其他模块使用。此外，network_monitor模块还实现了基于链路可用带宽的最优转发路径的计算，为其他模块提供最优路径信息。
通过设置链路权重参数，Shortest_forwarding应用可以基于带宽数据计算最优的转发路径。本质上，network_monitor基于当前流量，为新数据流选择最佳转发路径，也是一种网络流量负载均衡的实现，只是其调度算法相对简单。

使用方法

为解析权重和最短K路径的参数，还需要在Ryu中注册全局的启动参数。注册参数的方法十分简单，只需要在Ryu顶级目录下的flags.py文件中添加如下的代码即可：

```python
CONF.register_cli_opts([
        # k_shortest_forwarding
        cfg.IntOpt('k-paths',default=1,help='number for k shortest paths'),
        cfg.StrOpt('weight',default='hop',
                   help='type of computing shortest path.')])
```

完成以上修改后，将Github仓库中的代码下载到本地，然后放置到Ryu目录下合适的位置，比如Ryu/app目录下。
最后还需要重新安装Ryu：进入到ryu/的根目录，运行setup.py文件，并添加install参数。

```shell
sudo python setup.py install
```

重新安装完成之后，启动shortest_forwarding应用，并添加observe-links，链路权重和最短路径条数等重要参数,示例如下：

```shell
ryu-manager ryu/app/network_awareness/shortest_forwarding --observe-links --k-paths=2 --weight=bw
```

启动Ryu之后，启动任意的SDN网络，如Mininet模拟的网络，并连接到Ryu控制器。最后可以在Mininet输入框中输入pingall进行测试。

```shell
sudo mn--controller=remote --topo=tree,3,3 --mac
```

为了方便使用，读者可以通过修改setting.py中的信息来修改应用的重要参数，比如获取链路信息的周期，是否打印网络信息等等。setting信息具体如下所示：

```shell
DISCOVERY_PERIOD=10
MONITOR_PERIOD=10
DELAY_DETECTING_PERIOD=10

TOSHOW=True
MAX_CAPACITY=281474976710655L
```

读者可以通过修改对应的周期数值，来修改对应模块获取信息的周期，其单位为秒。TOSHOW是一个布尔值，用于设置是否在终端打印网络信息。MAX_CAPACITY值为链路最大可用带宽值，可根据实际情况进行修改。

总结

本文介绍了基于跳数、时延和带宽三种权重类型的最优转发应用，同时，基于带宽的最优转发也是一种简单的网络流量负载均衡应用。以上的代码其实在很久以前就已经写出来了，其是OXP（Open eXchange Protocol）应用的基础，但是由于某些原因，一直无法公开发布。前段时间在博客上发布了时延应用的原理，并没有把代码公布。但后来有若干读者发邮件询问代码，所以就趁着6月份的尾巴，把压在箱底的陈年应用发表出来，希望给大家带来一些帮助。在使用过程中建议读者先仔细阅读本文或README。如果遇到问题，可以通过电子邮件的方式和我沟通，我会很快把BUG修改好，不影响程序的使用体验。