Linux SRv6 实战

（第三篇）

多云环境下 Overlay(VPP) 和 Underlay 整合测试

李嘉明 思科系统工程师

苏远超 思科首席工程师

赵　勇 思科顾问工程师

摘要：本文基于 Linux 上开源的 VPP 软件以及 Cisco NCS5500 路由器，验证了通过 SRv6 实现多云环境下 Overlay 和 Underlay 整合、一体化调度以及快速收敛功能。

一、VPP 简介

关于 SRv6 原理及 Linux SRv6 常见功能(VPN、流量工程、服务链)的实现，请参见本系列文章的第一篇和第二篇，这里不再赘述。

VPP 全称 Vector Packet Processing（矢量数据包处理），最早是 Cisco 于 2002 年开发的商用代码。之后在 2016 年 Linux 基金会创建 FD.io 开源项目，Cisco 将 VPP 代码的开源版本加入该项目，目前已成为该项目的核心。现在一般来讲的 VPP 特指 FD.io VPP，即 Linux 基金会发起的开源项目。

这个项目在通用硬件平台上提供了具有灵活性、可扩展强、组件化等特点的高性能 IO 服务框架。该框架支持高吞吐量、低延迟、高资源利用率的 IO 服务，并可适用于多种硬件架构(x86/ARM/PowerPC)以及各种部署环境（裸机/VM/容器）。

VPP 底层使用了 Intel 数据平面开发工具包（DPDK：data plane development kit's）的轮询模式驱动程序（PMD：poll mode drivers）和环形缓冲区库，旨在通过减少数据流/转发表缓存的未命中数来增加转发平面吞吐量。

VPP 高度模块化，允许在不更改底层代码库的情况下轻松“插入”新的 Graph Node（图节点）。这使开发人员可以通过不同的转发图轻松构建任意数量的数据包处理解决方案，它通过 Graph Node（图节点）串联起来处理数据包，并可以通过插件的形式引入新的图节点或者重新排列数据包的图节点顺序，或者根据硬件情况通过某个节点直接连接硬件进行加速，因此可以用于构建任何类型的数据包处理应用。比如负载均衡、防火墙、IDS、主机栈等。目前业界已经有客户选用 VPP 来实现自己的数据包处理应用。

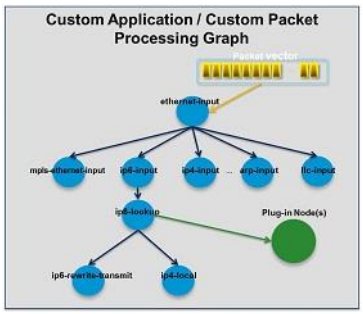

图 1 VPP 自定义程序包处理图

上图是 VPP 的自定义应用的处理图，除了 VPP 对一个包的默认处理流程外（图中蓝色的节点），可以在其中任意一个一个节点后，添加自定义的图节点，并在此实现自定义的操作或者扩展功能。

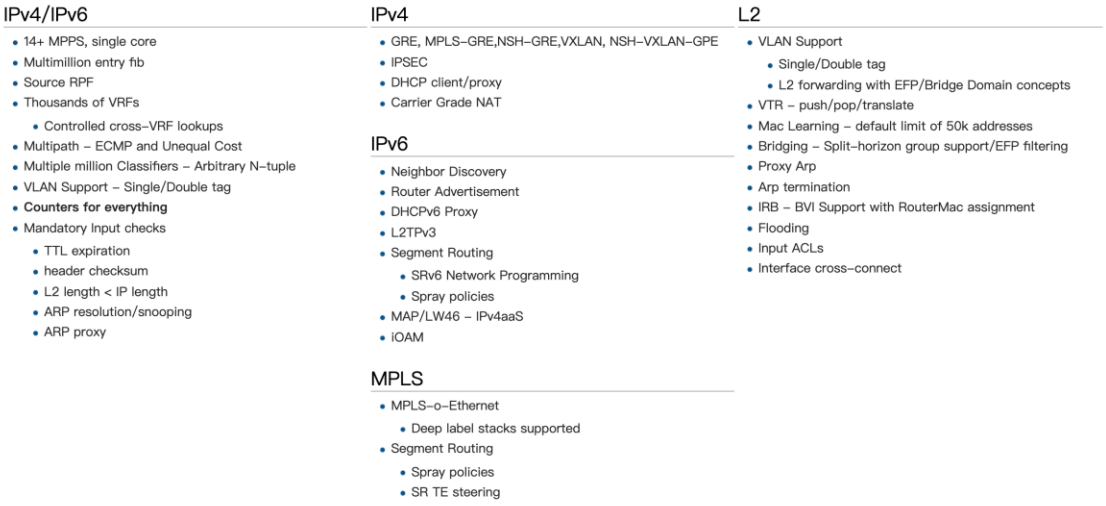

图 2 VPP 目前支持的功能一览

VPP 目前已经支持了很多的功能，具体如图 2 所示，很多功能还可以通过第三方开源软件+插件的方式进行扩展，比如动态路由协议 OSPF/BGP 等的支持。

VPP 对于 SRv6 有着非常好的支持，并且提供高性能。

二、多云环境下 Overlay 和 Underlay 的整合问题

一直以来，在多云环境下要实现 overlay 和 underlay 整合，都不太容易。一是如何将 Overlay 的 SLA 信息映射到 Underlay；二是 Underlay 能否以足够细的颗粒来满足 Overlay 基于流的 SLA 诉求；三是如何快速实现两者的协同和交接，因为这通常涉及两个不同的部门。

常见的方案有以下两种：

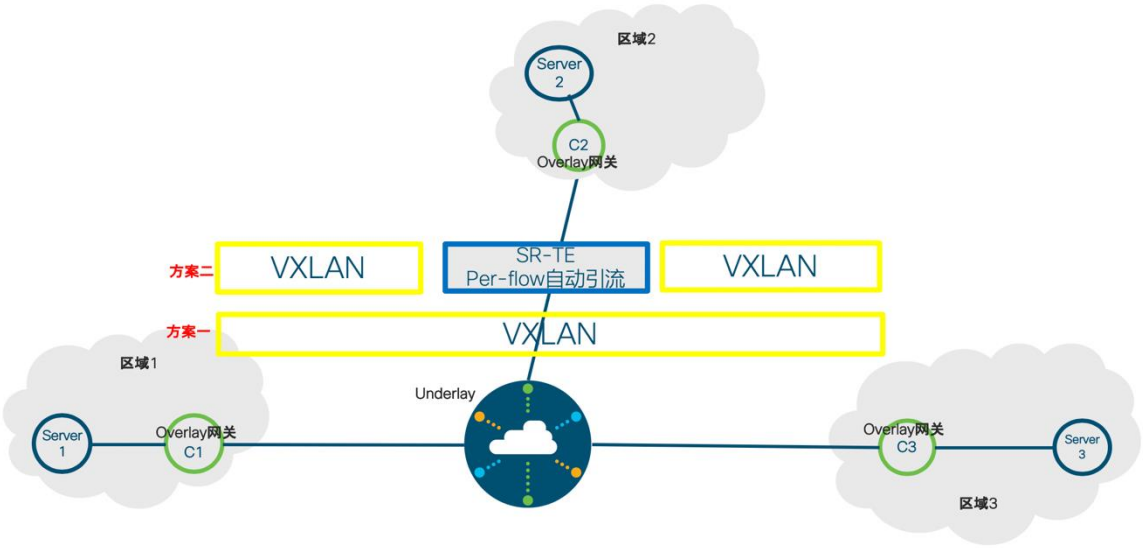

图 3 多云环境下 Overlay 和 Underlay 目前的整合方案

方案一：多云之间之间通过 VXLAN 打通，Underlay 就是纯粹的透传，对设备要求低。这种方案简单易行，不涉及到两个部门管理边界的问题；但缺点也很明显，对于需要 Underlay 提供 SLA 或多云间流量工程的情况(例如要求流量从 Overlay 网关 C1->C2->C3)，实现起来很困难。

方案二：每个云本地通过 IP+DSCP 的方式交接给 Underlay 的 SRTE PE，PE 设备需要支持基于流的自动引流功能。这种方案通过 SR-TE 基于流的自动引流实现了精细的映射，同时 SR-TE 自身的特性也保证了方案的可扩展性；需要指出的是，这种方案管理边界虽然清晰，但需要 Overlay 侧把 IP+DSCP 对应的 SLA 信息实时同步给 Underlay 侧；另外对 Underlay 设备要求较高，可能面临着软硬件升级的问题。

本文将讨论并验证第三种方案：基于 SRv6，在 Overlay 网关上利用 VPP 及 Underlay 的 SLA 信息生成 SRH(即发起 SR-TE)，这样可以充分利用前面谈到的方案 1 和方案 2 的优点：由于 Underlay 只需要执行简单的 END 操作，不需要作为基于流的 SR-TE 头端，因此无须对 Underlay 做大的改造，容易在现有网络中部署；Overlay 和 Underlay 交互的是 SLA 路径信息（这里可以利用 Flex-Algo 功能做进一步简化），边界清晰。同时最重要的是，本方案是把 Overlay 的 SLA 交给应用来负责（通过编程 Overlay 网关上的 SRH），因此可以提供高度的灵活性。

本方案用到的关键组件是 VPP。如图 2 所示，VPP 原生已经支持了 SRv6，包括常见的 SRv6 操作，下面我们简单的对比一下目前 Linux 上几种 SRv6 实现方式：

项目	Linux 内核原生	VPP	智能网卡(SmartNIC)
是否开源	是	是	否
支持的 END 操作	较全（部分需借助内核模块 srext）	全	取决于厂商实现，部分厂商实现较全
支持新/自定义的 END 操作	较容易，通过 END.BPF 实现（内核 4.18 以上）	一般，需要自行修改源代码	较难，需要等待厂商实现
性能	较低	高	最高
成本	免费	免费	需要投资
SRv6 功能支持与内核关联程度	直接相关，可能需要经常升级内核	运行在用户空间，VPP 升级对内核影响小	内核版本只要能驱动智能网卡本身即可，SRv6 功能升级对内核影响小

表 1 Linux SRv6、VPP 和 SmartNIC 的 SRv6 支持情况对比

如上表所示，VPP 是目前对 SRv6 支持最完备的开源项目。在性能方面，VPP 同样出色，比 Linux 内核原生支持的 SRv6 吞吐能力要超出很多；成本方面相比智能网卡则有着巨大优势。目前很多开源项目以及商业项目也选用了 VPP，其整体的稳定性和性能得到了广泛的验证。

因此本次实验中，我们选用 VPP 作为虚拟 SRv6 路由器，用于模拟需要支持 IPv6、高性能和高灵活性的 Host Overlay 网关。VPP 模拟的 SRv6 Overlay 网关结合 Cisco NCS5500 模拟的 SRv6 Underlay，验证了通过 SRv6 实现 Overlay 和 Underlay 的无缝/高性能整合以及一体化调度的强大能力。

三、实验准备工作

3.1 拓扑说明

1. 总体架构

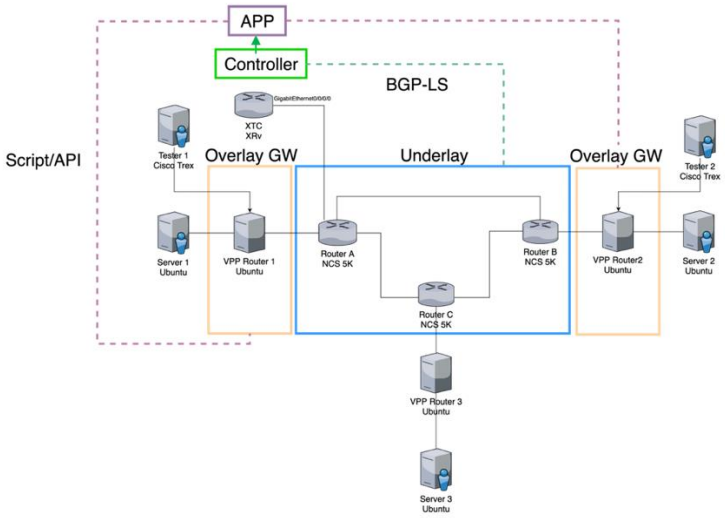

图 4 实验架构图

本次实验的总体架构如图所示，Underlay 部分主要由三台 Cisco NCS5500 构成，组成一个环状拓扑结构，负责 IPv6 报文的转发以及执行简单的 END 操作。Overlay 网关由 VPP 实现，VPP 作为 Overlay 的网关，负责将测试仪（这里我们使用开源的 Cisco Trex）以及服务器的 IPv4 流量进行封装，发往对端，以及执行 End.DX4 操作（去掉 IPv6 报头，转发所封装的 IPv4 数据包）。

控制器 Cisco XTC 通过 BGP-LS 得到全网拓扑信息，计算出最优路径，上层控制器应用（自己编写的示例程序）通过 Rest API 从控制器拿到最优路径信息，计算出新的 SRv6 Policy，并通过 VPP API 下发给 Overlay 网关设备，实现路径的实时更新。

2. 主要软硬件列表

组件	角色	版本	备注
Cisco NCS5500	Underlay 路由器	IOS XR 6.6	
VPP	Overlay 网关	18.07	
Cisco Trex	测试仪	2.56	
Cisco XTC	Underlay 控制器	IOS XR 6.6	以 VM 方式运行。对 XTC 的使用详见章节 4.2
控制器应用程序	示例程序，模拟控制 Overlay SLA 的上层应用		自行编写，源码见 https://github.com/ljm625/srv6_vpp_demo_controller
Ubuntu	Linux 操作系统	16.04	
Cisco UCS	硬件服务器		服务器均未配置智能网卡

表 2 本实验主要软硬件列表

3. 详细拓扑图

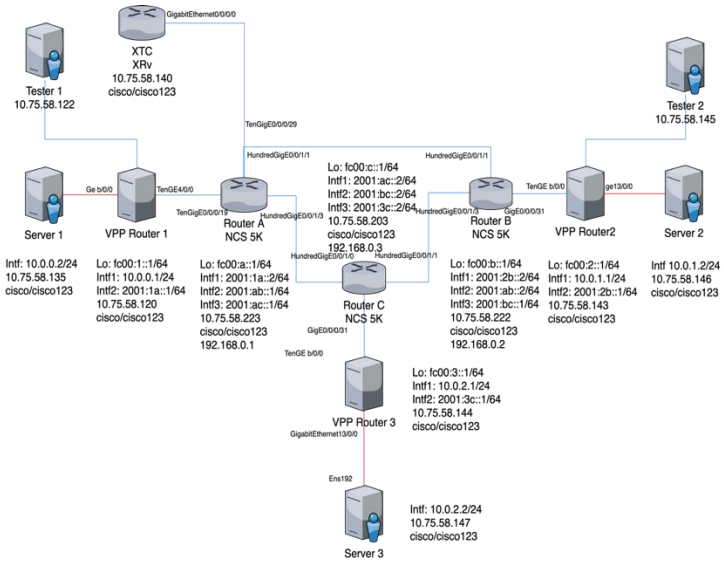

图 5 实验详细拓扑图

本次实验的详细拓扑如上图，中间为 3 台 Cisco NCS5500 路由器，构成了环形结构（Router A/B/C）。3 台 VPP 路由器(Overlay 网关)均安装在 ESXi 环境下，通过 PCI 直通分别与每台 NCS5500 直连。图中所有的蓝色链路均为物理连接，红色链路为通过 vSwitch 虚拟连接的链路。

三台服务器 Server1/Server2/Server3 通过 vSwitch 和 VPP 路由器分别连接，测试仪 1 和 2 为思科开源的 Trex 软件，安装在 Ubuntu 虚拟机上，通过 PCI 直通和 VPP 路由器 1 与 2 直连。

XTC 控制器为 IOS XRv 的虚拟机，通过 PCI 直通连接到 Router A。

三台 NCS5500 之间运行纯 IPv6。VPP 路由器为 IPv4+IPv6 双栈，VPP 和服务器/测试仪通过 IPv4 连接，和 NCS5500 通过 IPv6 连接。

测试仪 1/2 为基于 Ubuntu 16.04 安装的 Cisco Trex 开源测试仪软件。该测试软件有 2 种模式，Stateless（无状态）和 Advanced Stateful Mode（高级带状态模式），在这里我们使用的是后者，在后文中我们会详细说明。

3.2 安装 VPP

版本信息如图：

```
root@VPP:~# uname -sr; tail -2 /etc/lsb-release
Linux 4.4.0-116-generic
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04.4 LTS"
root@VPP:~#
```

图 6 VPP 安装的宿主 Ubuntu 版本信息

```
1. vi /etc/apt/sources.list.d/99fd.io.list
2.
3. deb [trusted=yes] https://nexus.fd.io/content/repositories/fd.io.stable.1807.ubu
   ntu.xenial.main/ ./
```

```
deb [trusted=yes] https://nexus.fd.io/content/repositories/fd.io.stable.1807.ubuntu.xenial.main/ ./
~
~
~
~
```

图 7 修改后的 APT Source 文件

```
1. apt update
2. apt dist-upgrade -y
3. apt install -y vpp vpp-lib vpp-plugins vpp-dpdk-dkms
```

安装完成之后，执行：

```
1. systemctl status vpp
```

如果能显示出 vpp 服务的详情，则代表安装完成。

```
root@VPP:~# systemctl status vpp
● vpp.service - vector packet processing engine
   Loaded: loaded (/lib/systemd/system/vpp.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2019-04-22 22:17:21 PDT; 5 days ago
  Process: 5288 ExecStopPost=/bin/rm -f /dev/shm/db /dev/shm/global_vm /dev/shm/vpe-api (code=exited, status=0/SUCCESS)
  Process: 5296 ExecStartPre=/sbin/modprobe uio_pci_generic (code=exited, status=0/SUCCESS)
  Process: 5293 ExecStartPre=/bin/rm -f /dev/shm/db /dev/shm/global_vm /dev/shm/vpe-api (code=exited, status=0/SUCCESS)
 Main PID: 5300 (vpp_main)
    Tasks: 5
   Memory: 116.0M
      CPU: 5d 1h 47min 13.101s
   CGroup: /system.slice/vpp.service
           └─5300 /usr/bin/vpp -c /etc/vpp/startup.conf

Apr 23 02:44:16 VPP vnet[5300]: sr policy: There is already a FIB entry for the BindingSID address.
                                The SR policy could not be created
Apr 23 02:44:46 VPP vnet[5300]: sr policy: There is already a FIB entry for the BindingSID address.
                                The SR policy could not be created
Apr 23 02:46:17 VPP vnet[5300]: sr policy: There is already a FIB entry for the BindingSID address.
                                The SR policy could not be created
Apr 23 02:46:17 VPP vnet[5300]: sr policy: There is already a FIB entry for the BindingSID address.
                                The SR policy could not be created
Apr 23 03:07:43 VPP vnet[5300]: sr policy: BUG: sr policy returns -
Apr 23 03:07:43 VPP vnet[5300]: sr policy: BUG: sr policy returns -
Apr 23 19:37:05 VPP vnet[5300]: ip route: parse error `via 2001:1a::2[D
Apr 23 20:34:33 VPP vnet[5300]: unknown input `| ?
Apr 24 01:29:18 VPP vnet[5300]: show: unknown input `intf addr
Apr 24 20:44:11 VPP vnet[5300]: unknown input `exit
root@VPP:~#
```

图 8 查看 VPP 的运行状态

3.3 配置 VPP

首先需要列出机器连接的 PCI 网络设备的 ID：

```
1.  lshw -class network -businfo
```

```
root@VPP:~# lshw -class network -businfo
Bus info          Device      Class      Description
=========================================================
pci@0000:03:00.0  ens160      network    VMXNET3 Ethernet Controller
pci@0000:04:00.0              network    Ethernet Controller X710 for 10GbE SFP+
pci@0000:0b:00.0              network    VMXNET3 Ethernet Controller
pci@0000:13:00.0              network    Ethernet Controller X710 for 10GbE SFP+
```

图 9 查看 Linux 上的网络 PCI 设备

如上图，这个是在 VPP1 上列出的结果，其中包括 4 块网卡，可以看到 2 块网卡是 VMXNET3 的，即 VMWare 虚拟机的虚拟网卡，剩下 2 块网卡则是 PCI 直通到虚拟机的网卡，为 X710 网卡的 2 个端口。在这里我们把后 3 块网卡交给 VPP 管理（**VPP 管理的网卡，不能再被系统使用，因此请预留一个管理口，如图的 ens160 给系统使用，以及 SSH 访问等**）。

在这里我们记录下来后 3 块网卡的 PCI id，分别为 0000:04:00.0，0000:0b:00.0 以及 0000:13:00.0。

接着修改 vpp 的配置文件：

```
1.  vi /etc/vpp/startup.conf
```

需要修改里面的 dpdk 部分：

```
1.  dpdk {
2.      dev 0000:04:00.0
3.      dev 0000:0b:00.0
4.      dev 0000:13:00.0
5.  }
```

```
dpdk {
        ## Change default settings for all intefaces
        # dev default {
                ## Number of receive queues, enables RSS
                ## Default is 1
                # num-rx-queues 3

                ## Number of transmit queues, Default is equal
                ## to number of worker threads or 1 if no workers treads
                # num-tx-queues 3

                ## Number of descriptors in transmit and receive rings
                ## increasing or reducing number can impact performance
                ## Default is 1024 for both rx and tx
                # num-rx-desc 512
                # num-tx-desc 512

                ## VLAN strip offload mode for interface
                ## Default is off
                # vlan-strip-offload on
        # }

        ## Whitelist specific interface by specifying PCI address
        # dev 0000:02:00.0
        dev 0000:04:00.0
        dev 0000:0b:00.0
        dev 0000:13:00.0
        ## Whitelist specific interface by specifying PCI address and in
```

图 10 修改后的 VPP 配置文件

保存文件，重启 VPP：

```
1.  systemctl restart vpp
2.  vppctl show pci
3.  vppctl show interface addr
```

```
vpp# show pci
Address      Sock VID:PID     Link Speed   Driver          Product Name                    Vital Product Data
0000:03:00.0      15ad:07b0   5.0 GT/s x32 vmxnet3
0000:04:00.0      8086:1572   5.0 GT/s x32 uio_pci_generic Cisco(R) Ethernet Converged NIC  PN: X710-DA4
                                                                                           V0: 0x 4d 41 50 20 35 2e 31 36 ...
                                                                                           MN: 1137
                                                                                           RV: 0x 2e
0000:0b:00.0      15ad:07b0   5.0 GT/s x32 uio_pci_generic
0000:13:00.0      8086:1572   5.0 GT/s x32 uio_pci_generic Cisco(R) Ethernet Converged NIC  PN: X710-DA4
                                                                                           V0: 0x 4d 41 50 20 35 2e 31 36 ...
                                                                                           MN: 1137
                                                                                           RV: 0x 2e
```

图 11 VPP 中的 PCI 信息

```
vpp# show interface addr
GigabitEthernetb/0/0 (dn):
TenGigabitEthernet13/0/0 (dn):
TenGigabitEthernet4/0/0 (dn):
local0 (dn):
vpp#
```

图 12 VPP 接管的端口和 IP 地址

可以看到刚刚添加的几个 Interface，说明配置成功：

0000:04:00.0 -> TenGigabitEthernet4/0/0（"04:00.0"所映射端口的编号"4/0/0"）

0000:0b:00.0 -> GigabitEthernetb/0/0（"0b:00.0"所映射端口的编号"b/0/0"）

0000:13:00.0 -> TenGigabitEthernet13/0/0（"13:00.0"所映射端口的编号"13/0/0"）

VPP 在每次重启之后，之前的配置就会丢失，因为 VPP 在设计之初就是希望其他程序通过 API 与其进行交互。在本次实验中我们通过 Day0 配置文件来保存配置（读者也可以通过插件集成到 VPP 中实现类似功能，这里不再赘述）。

接着进行 VPP 的初始配置，VPP 的初始配置由/etc/vpp/startup.conf 里面的配置决定：

```
unix {
  nodaemon
  log /var/log/vpp/vpp.log
  full-coredump
  cli-listen /run/vpp/cli.sock
  gid vpp
  startup-config /usr/share/vpp/scripts/interface-up.txt
}
```

图 13 VPP 的配置文件

添加一行：

```
1.  startup-config /path_to_config_file
```

在这里我们定义的 config 文件为/usr/share/vpp/scripts/interface-up.txt。

接下来编辑 interface-up.txt：

```
1.  # 配置 TenGigabitEthernet4/0/0 的 ip 地址和端口状态
2.  set interface state TenGigabitEthernet4/0/0 up
```

```
set interface ip address TenGigabitEthernet4/0/0 2001:1a::1/64
# 配置默认路由
ip route add ::/0 via 2001:1a::2
# 配置 GigabitEthernetb/0/0 的 ip 地址和端口状态
set interface state GigabitEthernetb/0/0 up
set interface ip address GigabitEthernetb/0/0 10.0.0.1/24
# 配置 TenGigabitEthernet13/0/0 的 ip 地址和端口状态
set interface state TenGigabitEthernet13/0/0 up
set interface ip address TenGigabitEthernet13/0/0 10.1.0.1/24
# 新建 Loopback 端口，配置 loopback 地址和端口状态
loopback create-interface
set interface ip address loop0 fc00:1::1/64
set interface state loop0 up
```

完成之后重启 vpp：

```
systemctl restart vpp
vppctl show interface addr
```

```
root@VPP:~# vppctl show interface addr
GigabitEthernetb/0/0 (up):
  L3 10.0.0.1/24
TenGigabitEthernet13/0/0 (up):
  L3 10.1.0.1/24
TenGigabitEthernet4/0/0 (up):
  L3 2001:1a::1/64
local0 (dn):
loop0 (up):
  L3 fc00:1::1/64
```

图 14 VPP 接管的端口和 IP 地址

可以看到已经完成了 day0 的接口、 IP 和路由配置。

3.4 配置 Cisco NCS5500

Cisco NCS5500 需要启用 SRv6 功能。在 Cisco NCS5500 启用 SRv6 功能并分配 Locator 之后，会自动获得 End 操作对应的 Segment，这点和 Linux 不一样。

需要配置：

```
hw-module profile segment-routing srv6
segment-routing
 srv6
  encapsulation
   source-address fc00:c::1
  !
  locators
   locator TEST
    prefix fc00:c:1::/64
```

```
10.    !
11.   !
12.  !
13. !
```

以上配置启用 SRv6 功能，并配置 SRv6 的 locator。如果配置之后 SRv6 没有生效，则需要执行：

```
1.  reset
```

该命令会重启 Cisco NCS5500，并启用新配置的 hw-module。

配置之后，可以通过下面的命令查看 locator 以及 Segment：

```
2.  show segment-routing srv6 locator
3.  show segment-routing srv6 sid
```

```
RP/0/RP0/CPU0:R33#show segment-routing srv6 locator
Wed Apr 24 07:13:38.023 UTC
Name                  ID       Prefix                   Status
--------------------  -------  -----------------------  -------
TEST*                 1        fc00:c:1::/64            Up
```

图 15 Cisco NCS5500 的 Locator 配置

可以看到 Locator 已经是 UP 状态。

```
RP/0/RP0/CPU0:R33#show segment-routing srv6 sid
Wed Apr 24 07:13:53.855 UTC

*** Locator: 'TEST' ***

SID                     Function     Context                      Owner              State  RW
----------------------  -----------  ---------------------------  -----------------  -----  --
fc00:c:1:0:1::          End (PSP)    'default':1                  sidmgr             InUse  Y
```

图 16 Cisco NCS5500 自动分配的 Segment 信息

自动分配与 End 操作对应的 Segment 为 fc00:c:1:0:1::

我们在三台 Cisco NCS5500 路由器上均进行该操作，对应的 End 操作 Segment 分别为：

路由器	Segment
Router A	fc00:a:1:0:1::
Router B	fc00:b:1:0:1::
Router C	fc00:c:1:0:1::

表 3 Cisco NCS5500 自动分配的 End 操作对应的 Segment

3.5 配置控制器应用程序运行环境

控制器应用程序为 Python 实现的示例程序，可以安装在任意平台，这里以 ubuntu 为例。

```
apt-get update
apt-get install python3 python3-pip git
python3 -m pip install requests paramiko
```

至此安装环境完成。

接着获取控制器的代码：

```
git clone https://github.com/ljm625/srv6_vpp_demo_controller.git
cd srv6_vpp_demo_controller
```

需要修改配置文件 config.json：

```
{
  "xtc_node":{ // XTC 控制器的连接信息，默认 REST API 端口为 8080
    "ip":"10.75.58.140",
    "username":"cisco",
    "password":"cisco123"
  },
  "vpp_node":{ // VPP 控制器应用程序的连接信息，默认使用 SSH 连接
    "ip":"10.75.58.120",
    "username":"root",
    "password":"cisco123"
  },
  "node_list":["node1","node2","node3"], // 节点列表和名称，要和下面的 table 一致
  "node_table":{ // NCS5500 路由器的 IPv4 Loopback 地址，用于算路
      "node1":"192.168.0.1",
      "node2":"192.168.0.2",
      "node3":"192.168.0.3"
  },
  "node_sid":{ // NCS5500 路由器的 End 操作对应的 Segment，由 NCS5500 自动生成
      "node1": "fc00:a:1:0:1::",
      "node2": "fc00:b:1:0:1::",
      "node3": "fc00:c:1:0:1::"
  },
  "node_prefix":["10.0.1.0/24","10.0.2.0/24"], // IPv4 的路由信息
  "node_dx4_sid":["fc00:2::a","fc00:3::a"] // DX4 操作对应的 Segment
}
```

按需修改以上配置文件，预置值为本实验环境的配置。

四、实验流程

4.1 Overlay 和 Underlay 整合下的性能测试

在这个测试里，我们将测试具有三个 Segment 的 SRv6 Policy 的端到端转发性能。

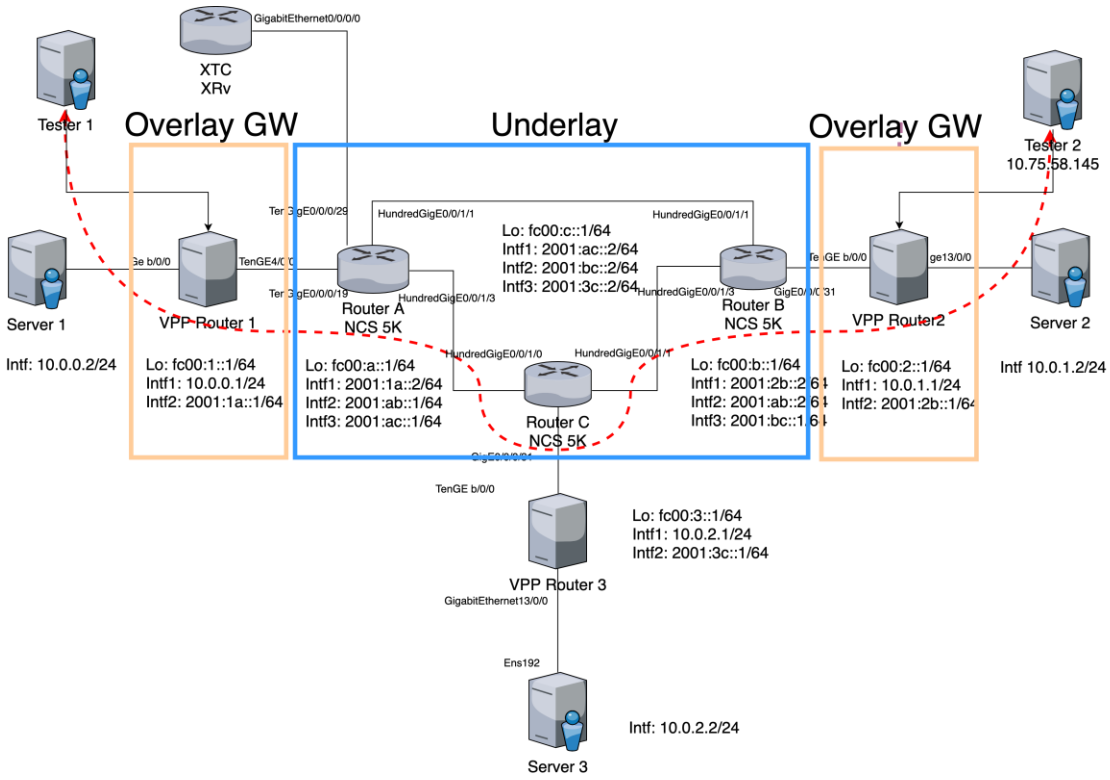

图 17 拓扑结构

拓扑结构如图所示，本次试验使用 2 台 Cisco 开源测试仪软件 Trex。Trex 使用 DPDK 作为底层驱动，可以达到很高的性能。Trex 分别部署在 Tester1 和 Tester2，本次测试从 Tester2 打流到 Tester1，并在 VPP2 上配置了一个包含三个 Segment 的 SRv6 Policy，其中的 2 个 Segment 为 NCS5500。Tester2 发送 594B 的 UDP 小包流量。

首先重新配置 VPP2：

```
1. sr localsid address fc00:2::1a behavior end.dx4 GigabitEthernet1b/0/0 10.1.1.2
2. # 定义 Segment fc00:2::1a 的操作为 End.DX4,将内部的 IPv4 包转发到
   GigabitEthernet1b/0/0 的 10.1.1.2 机器下(即 Tester2)
3. sr policy add bsid fc00:2::999:12 next fc00:b:1:0:1:: next fc00:a:1:0:1:: next f
   c00:1::1a encap
4. # 添加一个新的 Policy，包含三个 Segment，先到 Router B，再到 Router A，最后到 VPP1
5. sr steer l3 10.1.0.0/24 via bsid fc00:2::999:12
6. # 将去往 10.1.0.0/24 的包引导至新定义的 SRv6 Policy
```

```
vpp# show sr policies
SR policies:
[0].-   BSID: fc00:2::999:12
        Behavior: Encapsulation
        Type: Default
        FIB table: 0
        Segment Lists:
        [0].- < fc00:b:1:0:1::, fc00:a:1:0:1::, fc00:1::1a > weight: 1
-----------
```

图 18 VPP2 的 SRv6 Policy，包含 3 个 Segment

```
vpp# show sr localsids
SRv6 - My LocalSID Table:
=========================
        Address:        fc00:2::1a
        Behavior:       DX4 (Endpoint with decapsulation and IPv4 cross-connect)
        Iface:          GigabitEthernet1b/0/0
        Next hop:       10.1.1.2
        Good traffic:   [0 packets : 0 bytes]
        Bad traffic:    [0 packets : 0 bytes]
-------------------
```

图 19 VPP2 的本地 Segment 表

接着也需要在 VPP1 进行配置：

```
sr localsid address fc00:1::1a behavior end.dx4 TenGigabitEthernet13/0/0 10.1.0.2
# 定义 Segment fc00:1::1a 的操作为 End.DX4,将内部的 IPv4 包转发到 GigabitEthernet13/0/0 的 10.1.0.2 机器下(即 Tester1)
sr policy add bsid fc00:1::999:1a next fc00:2::1a encap
# 添加一个新的 Policy，回程直接到 VPP2
sr steer l3 10.1.1.0/24 via bsid fc00:1::999:1a
# 将去往 10.1.1.0/24 的包引导至新定义的 SRv6 Policy
```

```
-----------
[1].-   BSID: fc00:1::999:1a
        Behavior: Encapsulation
        Type: Default
        FIB table: 0
        Segment Lists:
        [1].- < fc00:2::1a > weight: 1
-----------
```

图 20 VPP1 的 SRv6 Policy，直接回到 VPP2

```
--------------------
        Address:        fc00:1::1a
        Behavior:       DX4 (Endpoint with decapsulation and IPv4 cross-connect)
        Iface:          TenGigabitEthernet13/0/0
        Next hop:       10.1.0.2
        Good traffic:   [0 packets : 0 bytes]
        Bad traffic:    [0 packets : 0 bytes]
--------------------
```

图 21 VPP1 的本地 Segment 表

Tester2 配置：

```
root@ubuntu:/opt/trex/v2.56# cat /etc/trex_cfg.yaml
- port_limit      : 2
  version         : 2
#List of interfaces. Change to suit your setup. Use ./dpdk_setup_ports.py -s to see available options
  interfaces    : ["0b:00.0","dummy"]
  port_info       :  # Port IPs. Change to suit your needs. In case of loopback, you can leave as is.
          - ip         : 10.1.1.2
            default_gw : 10.1.1.1
          - ip         : 1.1.1.1
            default_gw : 2.2.2.2
```

图 22 Tester2 的 Trex 配置

上图为 Trex 的配置文件，其中配置了 Trex 使用的网卡端口（通过 PCI ID 指定，如图中 interfaces 变量）以及网卡所配置的 IP，网关等（如图 port_info 下的信息）。在配置文件中，interfaces 变量对应的网卡端口数组中，第 1、3、5 个端口只能作为发送端（如上图的”0b:00:0”），数组第 2、4、6 个位置的端口只能作为接收端（如上图的 dummy，dummy 相当于一个假的接口），可以看到 Tester2 网卡被配置为发端。

Tester1 配置：

```
root@VPP3:/opt/trex/v2.56# cat /etc/trex_cfg.yaml
- port_limit      : 2
  version         : 2
#List of interfaces. Change to suit your setup. Use ./dpdk_setup_ports.py -s to see available options
  interfaces    : ["dummy","0b:00.0"]
  port_info       :  # Port IPs. Change to suit your needs. In case of loopback, you can leave as is.
          - ip         : 1.1.1.1
            default_gw : 2.2.2.2
          - ip         : 10.1.0.2
            default_gw : 10.1.0.1
```

图 23 Tester1 的 Trex 配置

上图为 Tester1 的配置文件，可以看到和 Tester2 的区别为，Tester1 的数组第一位为 dummy（即为空），第二位为 0b:00:0，即 Tester1 的网卡端口被配置为接收端。

```
root@ubuntu:/opt/trex/v2.56# cat astf/new.py
from trex.astf.api import *


class Prof1():
    def __init__(self):
        pass

    def get_profile(self, **kwargs):
        # ip generator
        ip_gen_c = ASTFIPGenDist(ip_range=["10.1.1.2", "10.1.1.2"], distribution="seq")
        print(ip_gen_c)
        ip_gen_s = ASTFIPGenDist(ip_range=["10.1.0.2", "10.1.0.2"], distribution="seq")
        ip_gen = ASTFIPGen(glob=ASTFIPGenGlobal(ip_offset="1.0.0.0"),
                           dist_client=ip_gen_c,
                           dist_server=ip_gen_s)

        return ASTFProfile(default_ip_gen=ip_gen,
                           cap_list=[ASTFCapInfo(file="../cap2/udp_594B.pcap",
                           cps=10000)])


def register():
    return Prof1()
```

图 24 Trex 发包对应代码

上图是测试仪发包的对应代码，主要核心部分在发送的包为 594B 的 UDP 包，并在每次执行时发送 10000 个所定义的 UDP 包。

首先使用测试仪 ping 对端：

```
trex>ping -p 0 -d 10.1.0.2

Pinging 10.1.0.2 from port 0 with 64 bytes of data:
Reply from 10.1.0.2: bytes=64, time=8.30ms, TTL=126
Reply from 10.1.0.2: bytes=64, time=3.61ms, TTL=126
Reply from 10.1.0.2: bytes=64, time=5.94ms, TTL=126
Reply from 10.1.0.2: bytes=64, time=3.19ms, TTL=126
Reply from 10.1.0.2: bytes=64, time=5.64ms, TTL=126
trex>
```

图 25 Tester2 ping Tester1

可以 Ping 通后，开始正式打流：

```
tui>start -f astf/new.py -m 15 -d 100
```

图 26 Tester2 运行测试脚本，开始打流

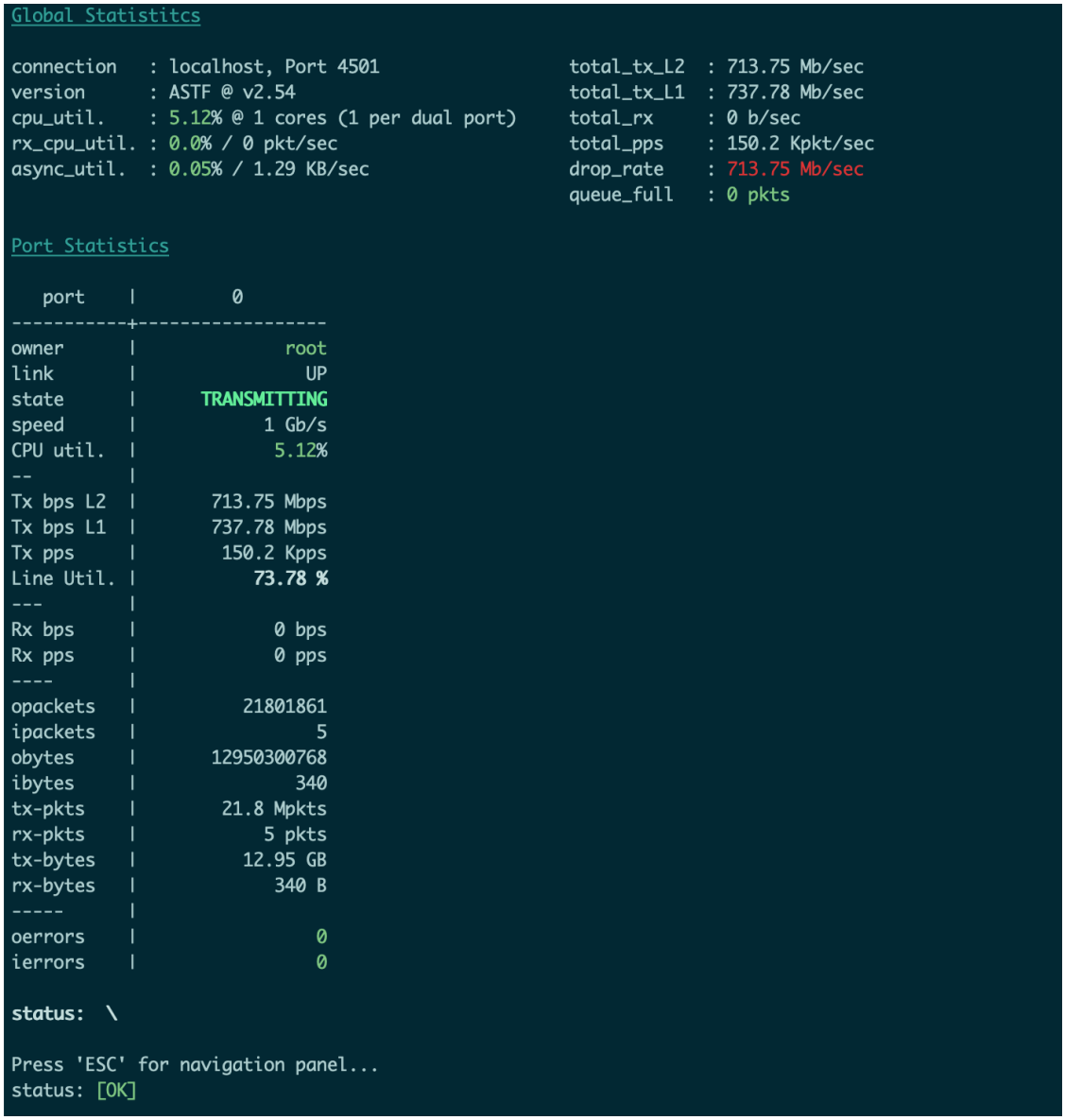

图 27 Tester2 的运行状态

上图为发端的状态，可以看到不断在根据我们定义的 UDP 数据包打流，流量转发速率为 710Mbps 左右，PPS 为 150.2K。

```
Global Statistitcs

connection   : localhost, Port 4501                total_tx_L2  : 0 b/sec
version      : STL @ v2.54                         total_tx_L1  : 0 b/sec
cpu_util.    : 0.0% @ 1 cores (1 per dual port)    total_rx     : 715.38 Mb/sec
rx_cpu_util. : 0.45% / 150.54 Kpkt/sec             total_pps    : 0 pkt/sec
async_util.  : 0.07% / 1.24 KB/sec                 drop_rate    : 0 b/sec
                                                   queue_full   : 0 pkts

Port Statistics

   port    |         1
-----------+------------------
owner      |              root
link       |                UP
state      |              IDLE
speed      |           10 Gb/s
CPU util.  |              0.0%
--         |
Tx bps L2  |             0 bps
Tx bps L1  |             0 bps
Tx pps     |             0 pps
Line Util. |               0 %
---        |
Rx bps     |       715.38 Mbps
Rx pps     |       150.54 Kpps
----       |
opackets   |                 0
ipackets   |          22424390
obytes     |                 0
ibytes     |       13320084300
tx-pkts    |            0 pkts
rx-pkts    |       22.42 Mpkts
tx-bytes   |               0 B
rx-bytes   |          13.32 GB
-----      |
oerrors    |                 0
ierrors    |                 0

status:  -

Press 'ESC' for navigation panel...
status:
```

图 28 Tester1 的运行状态

上图为收端的数据，可以看到接收端(Rx)的速率为 715Mbps 左右，PPS 为 150.5K 左右，和发送端速率一致，证明该速率可以正常转发，无丢包。

4.2 控制器应用程序实现动态路径调整和策略下发

在上一个测试中，我们测试了 VPP+NCS5500 环境下具有三个 Segment 的 SRv6 Policy 转发性能，表现符合预期，在这一节中，我们通过 Python 实现了一个示例控制器应用程序，通过调用 XTC 控制器的 Rest API 来获取算路信息，并动态的调整 SRv6 Policy，实现路径的调整。

控制器应用程序使用 Python 编写，具体的代码在：
https://github.com/ljm625/srv6_vpp_demo_controller

免责声明：本控制器应用程序代码只是用于示例说明，并不代表 Cisco 公司正式的实现方式，也不代表实践中仅限于这种实现方式。

前面在安装部分，我们已经解析了配置文件，下面简单解析一下控制器应用程序：

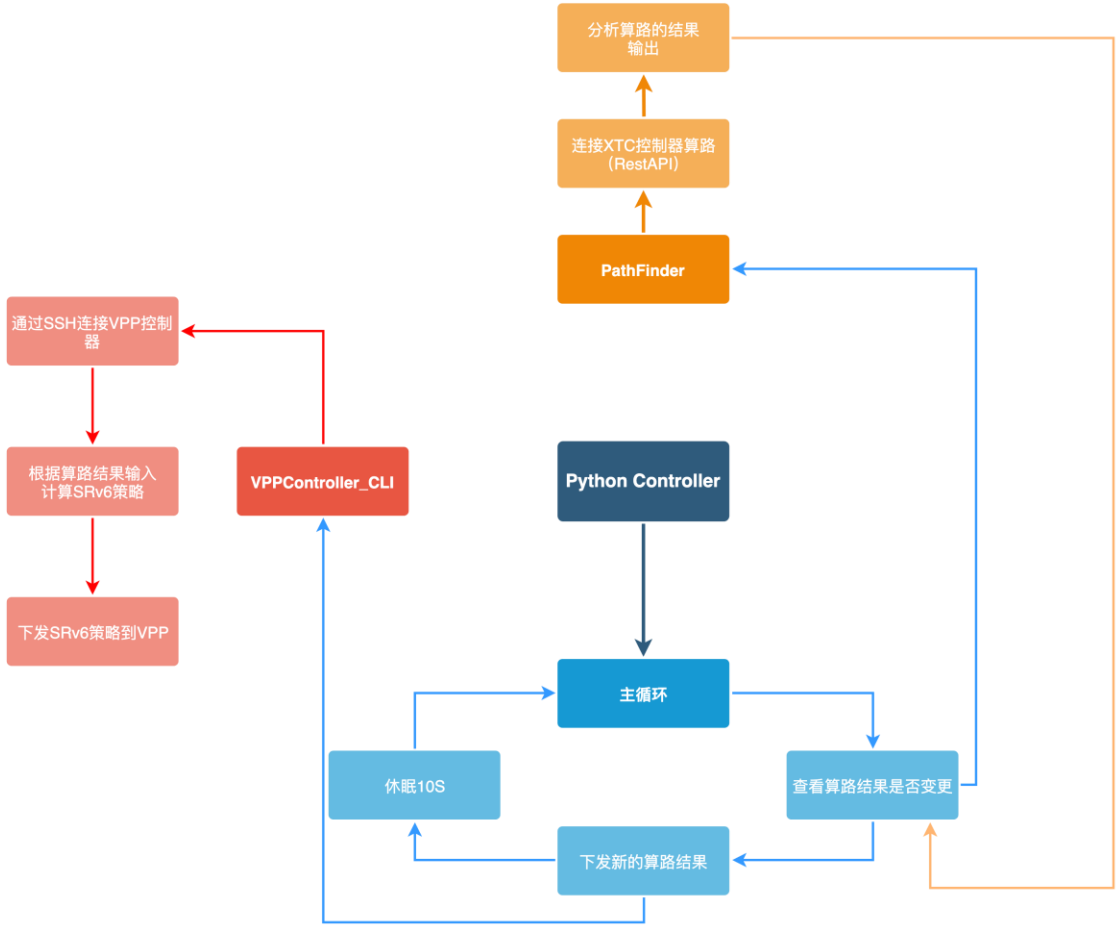

图 29 控制器应用程序流程图

上图为控制器应用程序流程图。

控制器应用程序主要分为三个部分：

1. 主循环

   主要进行循环，通过调用 PathFinder，检测算路结果是否发生变化，如果发生变化，调用 VPPController 模块生成新的 SRv6 Policy 并下发给 VPP 设备。

2. PathFinder 类

   主要通过 Rest API，调用 XTC 控制器，获取算路结果，并返回算路结果供使用。

重要说明：在写本文时，Cisco XTC 尚不支持直接计算 SRv6 Policy，只支持计算 SR-MPLS Policy，因此我们实际上是使用了 XTC 的 SR 原生算法，计算出基于相同优化目标和约束条件下的 SR-MPLS Policy 的 Segment 列表，但不下发（dry-run），然后用下面谈到的 VPPController 类构造相应的 SRv6 Policy 并下发。

3. VPPController_CLI 类

   这个类其实还有其他实现方法，这里使用的是 CLI 方式。当路径被计算出来之后，它会根据给出的节点生成对应的 SRv6 Policy，并通过 CLI 下发给 VPP 路由器。

接下来我们启动控制器应用程序，执行：

```
1.  python3 main.py
```

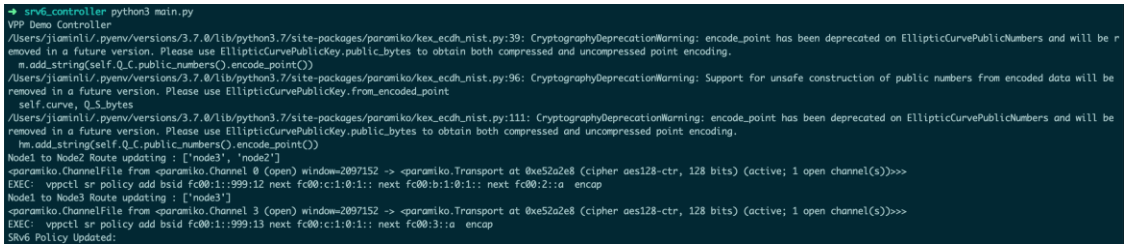

图 30 控制器应用程序运行状态

控制器应用启动之后，由于没有之前算路结果的记录，默认会自动进行算路和下发 SRv6 Policy 给 VPP1。

Underlay 控制器 XTC 得出的从 Router A 去往 Router B 的最低延迟算路结果为 Router A->Router C->Router B。

控制器应用程序下发给 VPP1 的 SRv6 Policy 为：

```
2.  vppctl sr policy add bsid fc00:1::999:12 next fc00:c:1:0:1:: next fc00:b:1:0:1::
     next fc00:2::a encap
```

表示去往 VPP2 的 SRv6 Policy，先去往 Router C 的 Segment，再去往的 Router B 的 Segment，最后去往 VPP2，和 Underlay 算路结果一致。

控制器应用程序还计算了从 Router A 去往 Router C 的最低延迟路径。

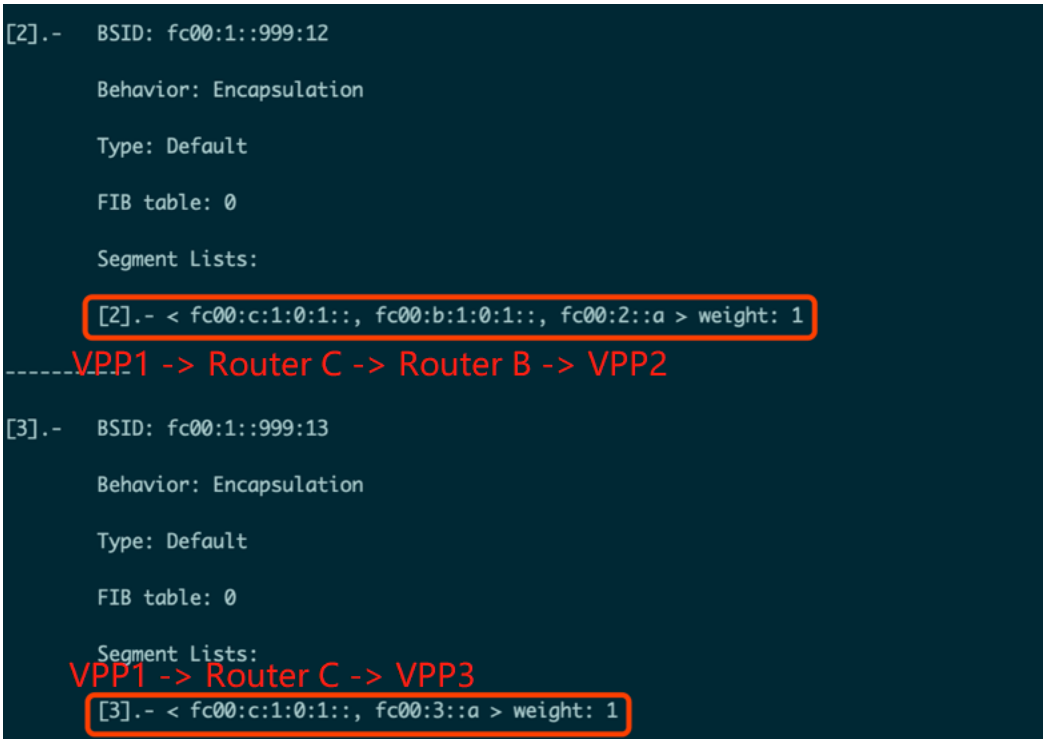

图 31 VPP1 上去往 VPP2/VPP3 的 SRv6 Policy

上图为 VPP1 上去往 VPP2/VPP3 的 SRv6 Policy，可以看到控制器应用程序已经动态下发了 SRv6 Policy 路径。

现在开始从 10.0.0.2 ping 10.0.1.2。在某个时刻，我们在 Router A 上手工配置延迟（在 performance-measurement 子模式下）并通过 ISIS 进行泛洪（现网情况下可以通过 SR 性能测量功能或者探针获得实延迟迟），使得最优路径发生变化。

```
performance-measurement
 interface HundredGigE0/0/1/1
  delay-measurement
   advertise-delay 500
  !
 !
 interface HundredGigE0/0/1/3
  delay-measurement
   advertise-delay 50
  !
 !
!
end

RP/0/RP0/CPU0:R31#config
Wed Apr 24 08:01:53.217 UTC
RP/0/RP0/CPU0:R31(config)#performance-measurement
RP/0/RP0/CPU0:R31(config-perf-meas)#interface HundredGigE 0/0/1/1
RP/0/RP0/CPU0:R31(config-pm-intf)#de
delay-measurement  describe
RP/0/RP0/CPU0:R31(config-pm-intf)#delay-measurement advertise-delay 50
RP/0/RP0/CPU0:R31(config-pm-intf)#exit
RP/0/RP0/CPU0:R31(config-perf-meas)#interface HundredGigE 0/0/1/3
RP/0/RP0/CPU0:R31(config-pm-intf)#de
delay-measurement  describe
RP/0/RP0/CPU0:R31(config-pm-intf)#delay-measurement advertise-delay 500
RP/0/RP0/CPU0:R31(config-pm-intf)#
```

图 32 在 Router A 上修改链路的延迟

如上图，在 Router A 上修改配置，将 2 条链路的延迟做了对调。

```
Node1 to Node2 Route updating : ['node2']
<paramiko.ChannelFile from <paramiko.Channel 7 (open) window=2097152 -> <paramiko.Transport at 0xe52a2e8 (cipher aes128-ctr, 128 bits) (active; 1 open channel(s))>>>
EXEC:  vppctl sr policy add bsid fc00:1::999:12 next fc00:b:1:0:1:: next fc00:2::a  encap
Node1 to Node3 Route updating : ['node2', 'node3']
<paramiko.ChannelFile from <paramiko.Channel 10 (open) window=2097152 -> <paramiko.Transport at 0xe52a2e8 (cipher aes128-ctr, 128 bits) (active; 1 open channel(s))>>>
EXEC:  vppctl sr policy add bsid fc00:1::999:13 next fc00:b:1:0:1:: next fc00:c:1:0:1:: next fc00:3::a  encap
SRv6 Policy Updated:
```

图 33 控制器应用程序新的算路结果

再次查看控制器应用程序，可以看到控制器应用程序检测到了路径的变化，重新进行了算路。

如上图，得出的从 Router A 去往 Router B 的最低延迟算路结果为 Router A->Router B 。

对应的 SRv6 Policy 为：

```
1. vppctl sr policy add bsid fc00:1::999:12 next fc00:b:1:0:1:: next fc00:2::a enca
   p
```

此时再去 VPP1 上查看 SRv6 Policy：

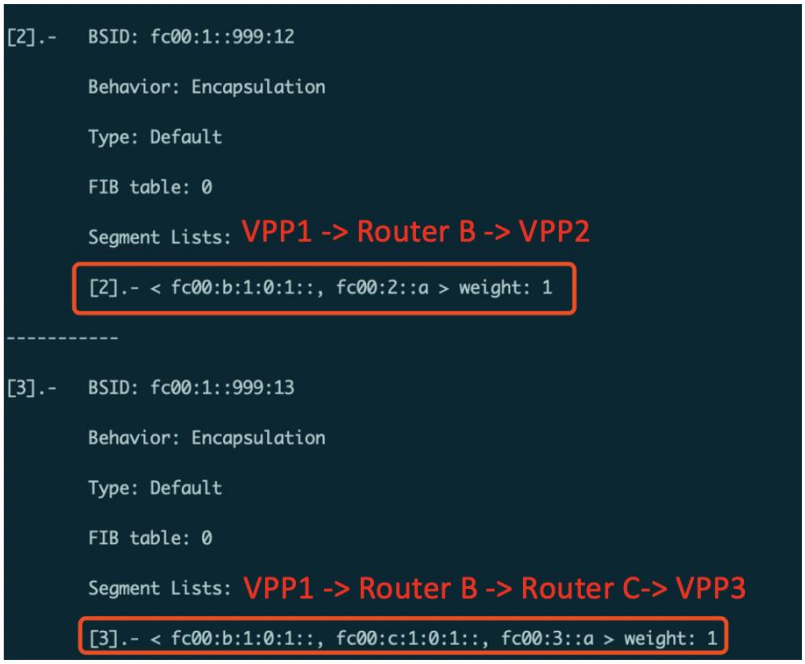

图 34 VPP1 上更新后的 SRv6 Policy

如上图红框部分，VPP1 上去往 VPP2/VPP3 的 SRv6 Policy 已经由控制器应用程序自动进行了更新。

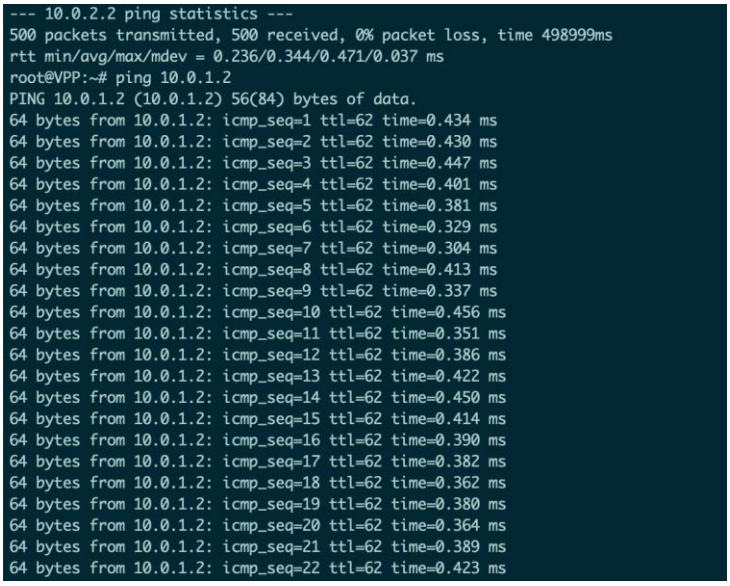

图 35 Server1 ping Server2

查看 ping 界面，在 SRv6 Policy 切换过程中没有出现丢包。

4.3 TI-LFA 保护测试

在上一个测试中，我们使用控制器应用程序来实现了路径的动态计算，在这个实验中，我们将利用 Cisco NCS5500 的 TI-LFA 功能，测试 TI-LFA 链路保护效果。需要注意的是，之前的实验并没有开启 TI-LFA。

在 Router A/Router B/Router C 上，在 ISIS 下对所有链路启用 TI-LFA 链路保护。相关配置如下所示（仅显示 TI-LFA 相关部分）：

```
router isis 1
 is-type level-2-only
 net 49.0001.1921.6800.0002.00
 distribute link-state
 address-family ipv4 unicast
  metric-style wide
  router-id Loopback0
  segment-routing mpls
 !
 address-family ipv6 unicast
  metric-style wide
  router-id Loopback0
  segment-routing srv6
   locator NCS2
   !
  !
 !
 interface Loopback0
  passive
  circuit-type level-2-only
  address-family ipv4 unicast
   prefix-sid absolute 16032
  !
  address-family ipv6 unicast
  !
 !
 interface HundredGigE0/0/1/1
  circuit-type level-2-only
  point-to-point
  address-family ipv4 unicast
   metric 1
  !
  address-family ipv6 unicast
   fast-reroute per-prefix
   fast-reroute per-prefix ti-lfa
   metric 1
  !
 !
 interface HundredGigE0/0/1/3
  circuit-type level-2-only
  point-to-point
  address-family ipv4 unicast
   metric 1
  !
  address-family ipv6 unicast
   fast-reroute per-prefix
   fast-reroute per-prefix ti-lfa
   metric 1
  !
 !
!
```

图 36 TI-LFA 相关配置

```
RP/0/RP0/CPU0:R32#show isis neighbors
Thu Apr 25 04:40:24.979 UTC

IS-IS 1 neighbors:
System Id      Interface        SNPA           State Holdtime Type IETF-NSF
R31            Hu0/0/1/1        *PtoP*         Up    22       L2   Capable
R33            Hu0/0/1/3        *PtoP*         Up    29       L2   Capable

Total neighbor count: 2
RP/0/RP0/CPU0:R32#show isis ipv6 fast-reroute summary
Thu Apr 25 04:40:36.873 UTC

IS-IS 1 IPv6 Unicast FRR summary

                        Critical   High       Medium     Low        Total
                        Priority   Priority   Priority   Priority
Prefixes reachable in L2
  All paths protected   0          0          2          3          5
  Some paths protected  0          0          0          0          0
  Unprotected           0          0          0          0          0
  Protection coverage   0.00%      0.00%      100.00%    100.00%    100.00%
```

图 37 ISIS 的邻居配置以及 IPv6 Fast-Reroute 信息

上图显示了 Router B 上的所有链路均已启用保护（具有备份链路）

```
RP/0/RP0/CPU0:R32#show isis ipv6 fast-reroute
Thu Apr 25 04:40:43.021 UTC

IS-IS 1 IPv6 Unicast FRR backups

Codes: L1 - level 1, L2 - level 2, ia - interarea (leaked into level 1)
       df - level 1 default (closest attached router), su - summary null
       C - connected, S - static, R - RIP, B - BGP, O - OSPF
       E - EIGRP, A - access/subscriber, M - mobile, a - application
       i - IS-IS (redistributed from another instance)
       D - Downstream, LC - Line card disjoint, NP - Node protecting
       P - Primary path, SRLG - SRLG disjoint, TM - Total metric via backup

Maximum parallel path count: 8

C  2001:ab::/64
     is directly connected, HundredGigE0/0/1/1
   L2 RIB backup [2/115]
     via fe80::2bc:60ff:fe72:4a4, HundredGigE0/0/1/1, R31, SRGB Base: 16000, Weight: 0
       No FRR backup
L2 2001:ac::/64 [2/115]
     via fe80::2bc:60ff:fe72:4a4, HundredGigE0/0/1/1, R31, SRGB Base: 16000, Weight: 0
       Backup path: LFA, via fe80::2bc:60ff:fe71:d4a4, HundredGigE0/0/1/3, R33, SRGB Base: 16000, Weight: 0, Metric: 2
     via fe80::2bc:60ff:fe71:d4a4, HundredGigE0/0/1/3, R33, SRGB Base: 16000, Weight: 0
       Backup path: LFA, via fe80::2bc:60ff:fe72:4a4, HundredGigE0/0/1/1, R31, SRGB Base: 16000, Weight: 0, Metric: 2
C  2001:bc::/64
     is directly connected, HundredGigE0/0/1/3
   L2 RIB backup [2/115]
     via fe80::2bc:60ff:fe71:d4a4, HundredGigE0/0/1/3, R33, SRGB Base: 16000, Weight: 0
       No FRR backup
L2 fc00:a::1/128 [1/115]
     via fe80::2bc:60ff:fe72:4a4, HundredGigE0/0/1/1, R31, SRGB Base: 16000, Weight: 0
       Backup path: LFA, via fe80::2bc:60ff:fe71:d4a4, HundredGigE0/0/1/3, R33, SRGB Base: 16000, Weight: 0, Metric: 2
L2 fc00:a:1::/64 [2/115]
     via fe80::2bc:60ff:fe72:4a4, HundredGigE0/0/1/1, R31, SRGB Base: 16000, Weight: 0
       Backup path: LFA, via fe80::2bc:60ff:fe71:d4a4, HundredGigE0/0/1/3, R33, SRGB Base: 16000, Weight: 0, Metric: 3
L2 fc00:c::1/128 [1/115]
     via fe80::2bc:60ff:fe71:d4a4, HundredGigE0/0/1/3, R33, SRGB Base: 16000, Weight: 0
       Backup path: LFA, via fe80::2bc:60ff:fe72:4a4, HundredGigE0/0/1/1, R31, SRGB Base: 16000, Weight: 0, Metric: 2
L2 fc00:c:1::/64 [2/115]
     via fe80::2bc:60ff:fe71:d4a4, HundredGigE0/0/1/3, R33, SRGB Base: 16000, Weight: 0
       Backup path: LFA, via fe80::2bc:60ff:fe72:4a4, HundredGigE0/0/1/1, R31, SRGB Base: 16000, Weight: 0, Metric: 3
```

图 38 TI-LFA 备份路径信息

在图 38 所示的信息中，可以看到 fc00: a:1::/64 (下一个 Segment)的备份路径为 HundredGigE0/0/1/3。

在开始测试之前，我们再次确认 VPP2 上去往 VPP1 的 SRv6 Policy：

```
[6].-   BSID: fc00:2::999:12
        Behavior: Encapsulation
        Type: Default
        FIB table: 0
        Segment Lists:
        [6].- < fc00:b:1:0:1::, fc00:a:1:0:1::, fc00:1::1a > weight:
```

VPP2 -> Router B -> Router A -> VPP1

图 39 VPP2 上的 SRv6 Policy

SRv6 Policy 为先去往 Router B，再去往 Router A，最后到达 VPP1 执行 End.DX4 操作。

下面开始打流测试：

```
Global Statistitcs

connection   : localhost, Port 4501              total_tx_L2  : 0 b/sec
version      : ASTF @ v2.54                      total_tx_L1  : 0 b/sec
cpu_util.    : 0.0% @ 1 cores (1 per dual port)  total_rx     : 0 b/sec
rx_cpu_util. : 0.0% / 0 pkt/sec                  total_pps    : 0 pkt/sec
async_util.  : 0.05% / 1.21 KB/sec               drop_rate    : 0 b/sec
                                                 queue_full   : 0 pkts

Port Statistics

   port    |        0
-----------+------------------
owner      |             root
link       |               UP
state      |           LOADED
speed      |           1 Gb/s
CPU util.  |             0.0%
--         |
Tx bps L2  |            0 bps
Tx bps L1  |            0 bps
Tx pps     |            0 pps
Line Util. |              0 %
---        |
Rx bps     |            0 bps
Rx pps     |            0 pps
----       |
opackets   |                0
ipackets   |                0
obytes     |                0
ibytes     |                0
tx-pkts    |           0 pkts
rx-pkts    |           0 pkts
tx-bytes   |              0 B
rx-bytes   |              0 B
-----      |
oerrors    |                0
ierrors    |                0

status:  -

Press 'ESC' for navigation panel...
status: [OK]

tui>start -f astf/new.py -m 16 -d 100
```

图 40 Tester2（发送端）打流

在 Tester2 运行 UDP 打流代码，开始打流：

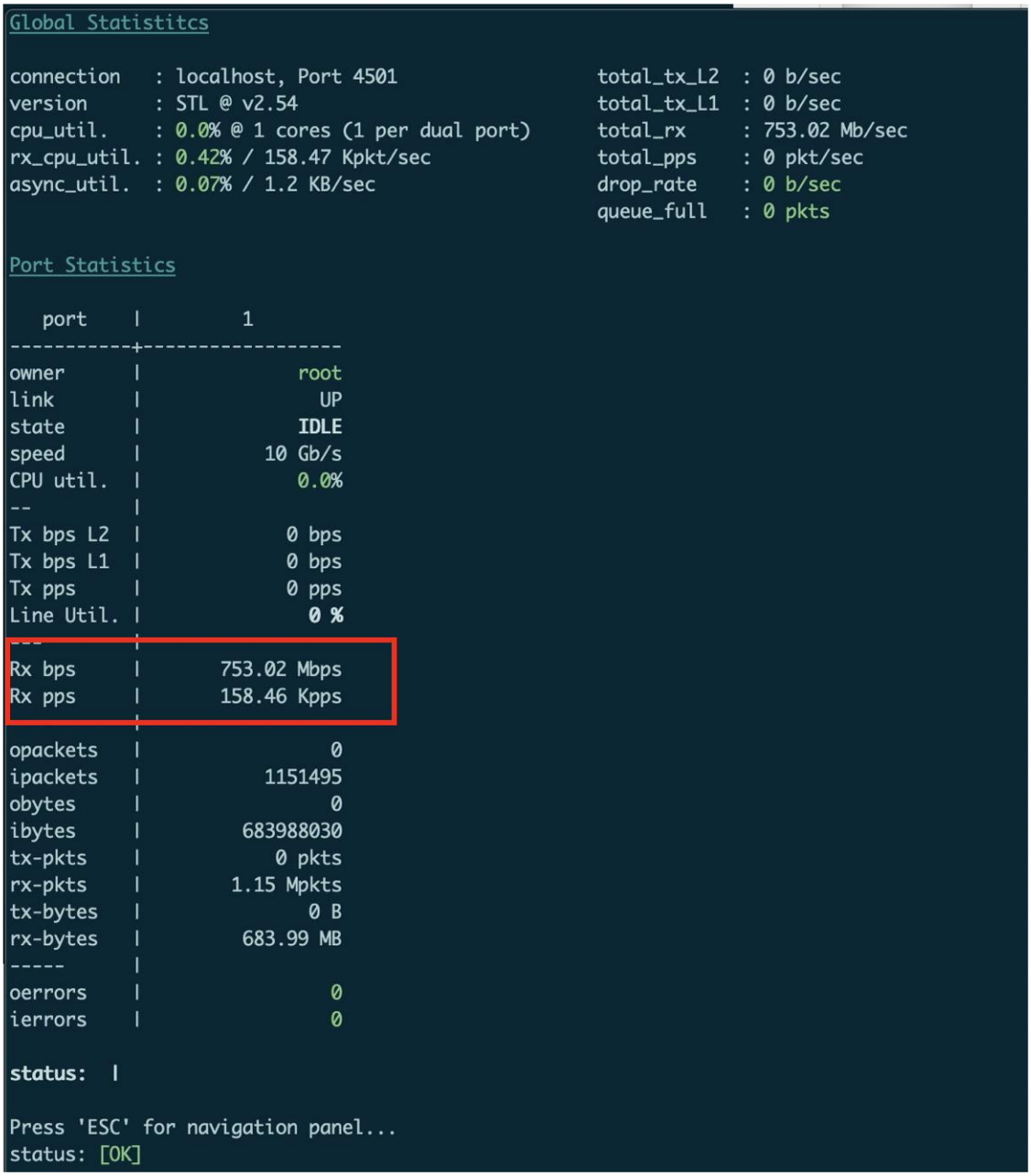

图 41 Tester1（接收端）状态

Tester1(接收端)可以收到发送端的流量。

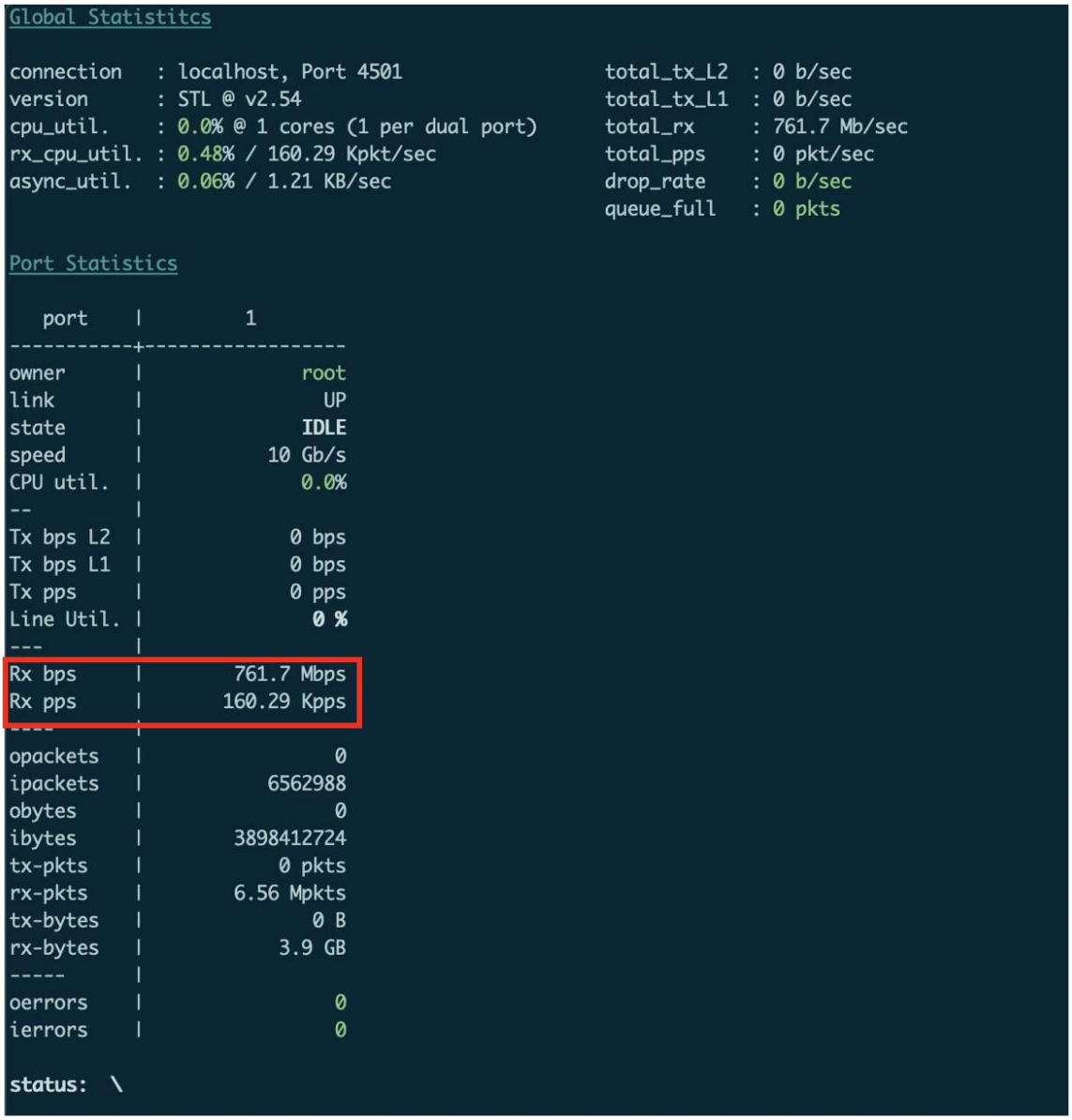

图 42 Router B 上的端口统计信息

从上图可见目前流量经由 HGE 0/0/1/1 转发。

接下来 shutdown 目前用于转发的 HGE 0/0/1/1 接口。

```
RP/0/RP0/CPU0:R32#config
Thu Apr 25 04:42:32.113 UTC
RP/0/RP0/CPU0:R32(config)#interface HundredGigE 0/0/1/
0/0/1/0  0/0/1/1  0/0/1/2  0/0/1/3
RP/0/RP0/CPU0:R32(config)#interface HundredGigE 0/0/1/1
RP/0/RP0/CPU0:R32(config-if)#shutdown
RP/0/RP0/CPU0:R32(config-if)#commit
Thu Apr 25 04:42:40.347 UTC
```

图 43 Shutdown 目前用于转发的端口

```
Global Statistitcs

connection   : localhost, Port 4501                total_tx_L2  : 0 b/sec
version      : STL @ v2.54                         total_tx_L1  : 0 b/sec
cpu_util.    : 0.0% @ 1 cores (1 per dual port)    total_rx     : 761.7 Mb/sec
rx_cpu_util. : 0.48% / 160.29 Kpkt/sec             total_pps    : 0 pkt/sec
async_util.  : 0.06% / 1.21 KB/sec                 drop_rate    : 0 b/sec
                                                   queue_full   : 0 pkts

Port Statistics

   port    |         1
-----------+------------------
owner      |              root
link       |                UP
state      |              IDLE
speed      |           10 Gb/s
CPU util.  |              0.0%
--         |
Tx bps L2  |             0 bps
Tx bps L1  |             0 bps
Tx pps     |             0 pps
Line Util. |               0 %
---        |
Rx bps     |         761.7 Mbps
Rx pps     |        160.29 Kpps
-----      |
opackets   |                 0
ipackets   |           6562988
obytes     |                 0
ibytes     |        3898412724
tx-pkts    |            0 pkts
rx-pkts    |         6.56 Mpkts
tx-bytes   |               0 B
rx-bytes   |            3.9 GB
-----      |
oerrors    |                 0
ierrors    |                 0

status:  \
```

图 44 Tester1（接收端）运行状态

从 Tester1 没有看到速率有所下降，说明自动切换路径成功。

```
RP/0/RP0/CPU0:R32#show interfaces HundredGigE 0/0/1/1
Thu Apr 25 04:42:50.658 UTC
HundredGigE0/0/1/1 is administratively down, line protocol is administratively down
  Interface state transitions: 28
```

图 45 Router B 上的端口信息

端口信息确认 HGE 0/0/1/1 已经被 shutdown，因此无法继续转发。

```
RP/0/RP0/CPU0:R32#show interfaces HundredGigE 0/0/1/3
Thu Apr 25 04:42:54.679 UTC
HundredGigE0/0/1/3 is up, line protocol is up
  Interface state transitions: 5
  Hardware is HundredGigE, address is 00bc.6071.ecac (bia 00bc.6071.ecac)
  Description: To-R33-HundredGigE0/0/1/1
  Internet address is 23.1.1.32/24
  MTU 9000 bytes, BW 100000000 Kbit (Max: 100000000 Kbit)
     reliability 255/255, txload 0/255, rxload 0/255
  Encapsulation ARPA,
  Full-duplex, 100000Mb/s, 100GBASE-AOC, link type is force-up
  output flow control is off, input flow control is off
  Carrier delay (up) is 10 msec
  loopback not set,
  Last link flapped 22:18:06
  ARP type ARPA, ARP timeout 04:00:00
  Last input 00:00:00, output 00:00:00
  Last clearing of "show interface" counters never
  30 second input rate 10000 bits/sec, 1 packets/sec
  30 second output rate 256603000 bits/sec, 46756 packets/sec
     15026493 packets input, 10370794171 bytes, 0 total input drops
     0 drops for unrecognized upper-level protocol
     Received 5 broadcast packets, 38158 multicast packets
              0 runs, 0 giants, 0 throttles, 0 parity
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     155405629 packets output, 106681578290 bytes, 0 total output drops
     Output 4 broadcast packets, 10421 multicast packets
     0 output errors, 0 underruns, 0 applique, 0 resets
     0 output buffer failures, 0 output buffers swapped out
     5 carrier transitions
```

图 46 Router B 上 HGE 0/0/1/3 的端口信息

查看 HundredGigE 0/0/1/3 的端口信息，发现 Router B 开始使用该端口转发流量。

```
RP/0/RP0/CPU0:R32#show isis ipv6 fast-reroute
Thu Apr 25 04:43:04.214 UTC

IS-IS 1 IPv6 Unicast FRR backups

Codes: L1 - level 1, L2 - level 2, ia - interarea (leaked into level 1)
       df - level 1 default (closest attached router), su - summary null
       C - connected, S - static, R - RIP, B - BGP, O - OSPF
       E - EIGRP, A - access/subscriber, M - mobile, a - application
       i - IS-IS (redistributed from another instance)
       D - Downstream, LC - Line card disjoint, NP - Node protecting
       P - Primary path, SRLG - SRLG disjoint, TM - Total metric via backup

Maximum parallel path count: 8

L2 2001:ac::/64 [2/115]
     via fe80::2bc:60ff:fe71:d4a4, HundredGigE0/0/1/3, R33, SRGB Base: 16000, Weight: 0
       No FRR backup
C  2001:bc::/64
     is directly connected, HundredGigE0/0/1/3
   L2 RIB backup [2/115]
     via fe80::2bc:60ff:fe71:d4a4, HundredGigE0/0/1/3, R33, SRGB Base: 16000, Weight: 0
       No FRR backup
L2 fc00:a::1/128 [2/115]
     via fe80::2bc:60ff:fe71:d4a4, HundredGigE0/0/1/3, R33, SRGB Base: 16000, Weight: 0
       No FRR backup
L2 fc00:a:1::/64 [3/115]
     via fe80::2bc:60ff:fe71:d4a4, HundredGigE0/0/1/3, R33, SRGB Base: 16000, Weight: 0
       No FRR backup
L2 fc00:c::1/128 [1/115]
     via fe80::2bc:60ff:fe71:d4a4, HundredGigE0/0/1/3, R33, SRGB Base: 16000, Weight: 0
       No FRR backup
L2 fc00:c:1::/64 [2/115]
     via fe80::2bc:60ff:fe71:d4a4, HundredGigE0/0/1/3, R33, SRGB Base: 16000, Weight: 0
       No FRR backup
```

图 47 Router B 上的 FRR 信息

从 fast-reroute 的输出来看，切换后已经没有了备份链路，默认为 HGE 0/0/1/3（FRR 路径即收敛后路径）， 证明切换成功。

接着等待实验发包结束。

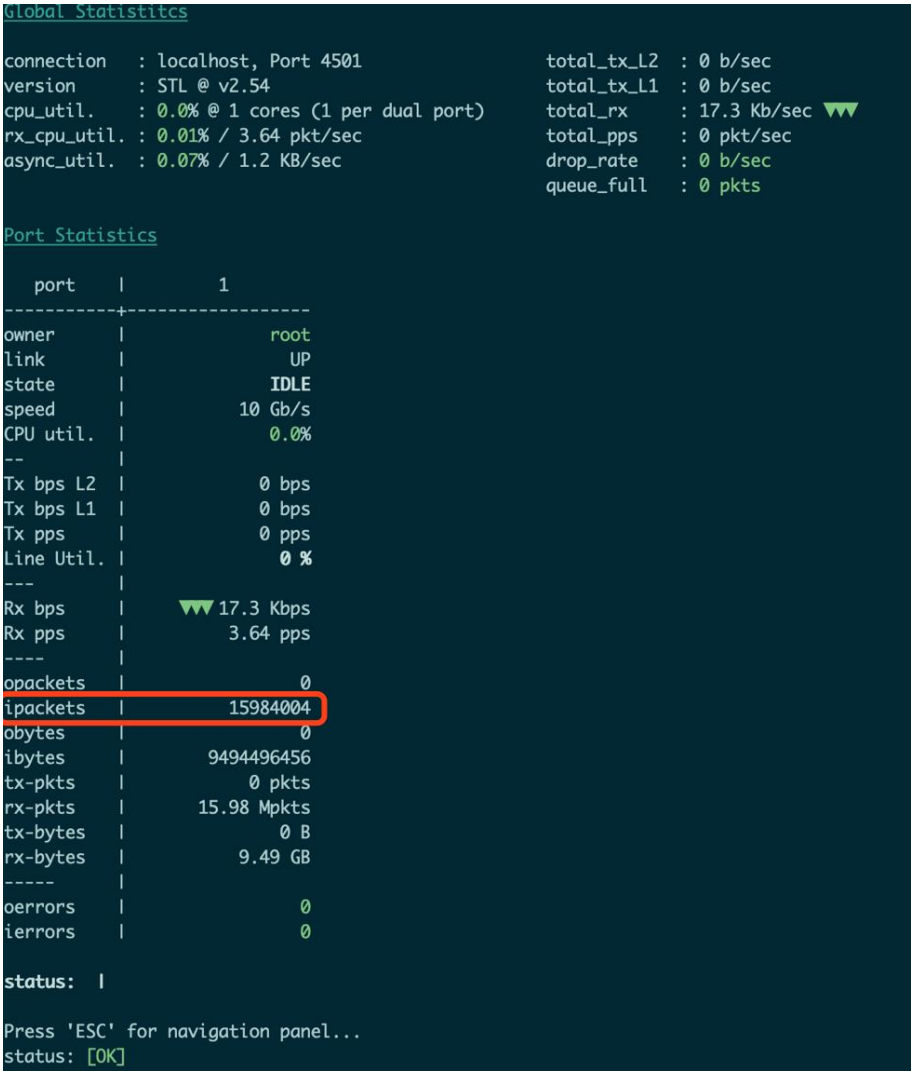

图 48 Tester1（接收端）统计信息

从 Tester1 上来看接收到了 15,984,004 个数据包。

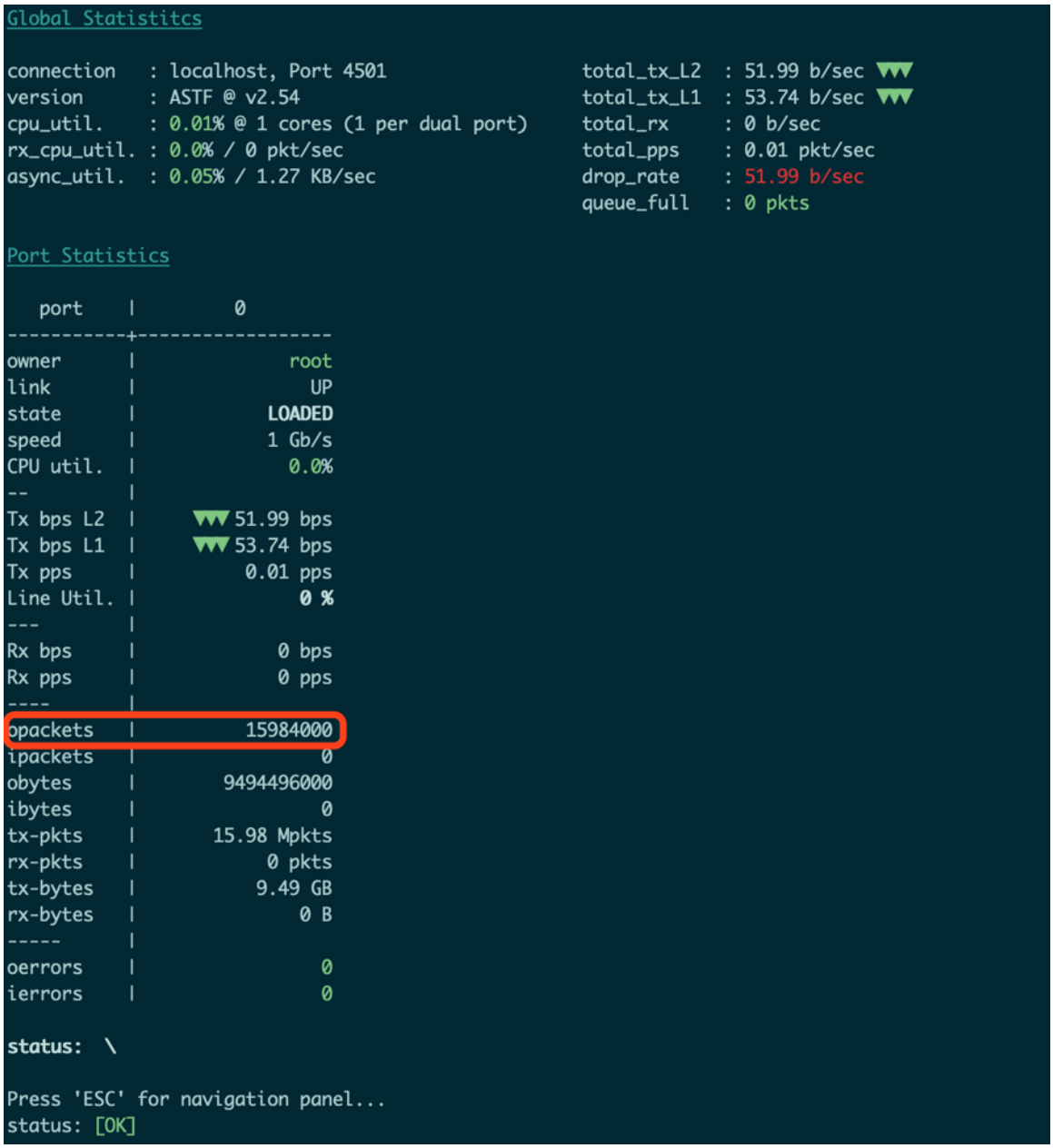

图 49 Tester2（发送端）统计信息

从 Tester2 上来看，发出了 15,984,000 个数据包，Tester1 多收的一些数据包为 ARP 等数据包，从数据包数量来看，没有出现丢包。

该测试总计测试了 5 次，整理后的数据如下：

测试次数	Tester2 发包数	Tester1 收包数
1	15,984,000	15,984,007
2	15,984,000	15,984,005
3	15,984,000	15,984,000

4	15,984,000	15,984,004
5	15,984,000	15,984,005

表 4 TI-LFA 保护测试结果

上表为多次测试下的测试结果。可以看到在目前的 PPS 速率下基本没有测出丢包，TI-LFA 保护效果非常好。

五、总结与展望

本文基于 Linux 上开源的 VPP 软件以及 Cisco NCS5500 路由器，验证了通过 SRv6 实现多云环境下 Overlay 和 Underlay 整合、一体化调度以及快速收敛功能。结果表明，这个方案在现有软硬件条件下就可以部署，不用进行大规模的升级，也能满足业务对性能、收敛方面的需求。

本系列的三篇文章，我们由浅入深地验证了 Linux 上 SRv6 的各项功能、不同的实现方式(内核/内核模块/VPP)、最后进行了性能测试和保护测试。希望能帮助各位读者深入了解 SRv6 的强大功能、快速上手 Linux SRv6 以及利用 SRv6 解决实际问题。

如笔者之前在公开演讲中所述，SRv6 两个最重要的特点是极简和可编程，是全新的思考、设计、运营网络的方式，是一次彻底的变革。

大潮已来，势不可挡。Let‘s Embrace SRv6!

【参考文献】

1. SRH draft: https://tools.ietf.org/html/draft-ietf-6man-segment-routing-header-18
2. SRv6 draft：https://tools.ietf.org/html/draft-ietf-spring-srv6-network-programming-00
3. Flex-Algo(灵活算法)：https://tools.ietf.org/html/draft-ietf-lsr-flex-algo-01
4. VPP 简介：https://wiki.fd.io/view/VPP/What_is_VPP%3F
5. VPP 18.07 版本相关资料：https://docs.fd.io/vpp/18.07/
6. VPP SRv6 相关资料/教程：https://wiki.fd.io/view/VPP/Segment_Routing_for_IPv6
7. Cisco Trex 的相关资料/教程：https://trex-tgn.cisco.com/
8. Cisco Trex ASTF(本实验用的模式)的相关资料/教程：https://trex-tgn.cisco.com/trex/doc/trex_astf.html