Linux SRv6 实战

（第四篇）

“以应用为中心”的 Overlay & Underlay 整合方案

苏远超 思科首席工程师

李嘉明 思科系统工程师

摘要：本文阐述并实现了以应用为中心的集中式/分布式 Overlay & Underlay 整合方案，将重点介绍分布式方案，最后对集中式方案和分布式方案的性能做简单的对比。

一、“以应用为中心”的 Overlay & Underlay 整合理念

随着多云应用成为新常态，多云环境下 Underlay 和 Overlay 整合需求越来越多，而业界目前两种常见的解决方案都是“以网络为中心”的（VXLAN 或者基于流交接给 SR-TE），均存在一定的不足，具体见本系列文章的第三篇，这里不再赘述。

SRv6 天生具备整合 Underlay、Overlay 能力。而且通过使用 uSID（详见作者关于 uSID 的文章），可以轻而易举地基于现有设备编码超长的 SR-TE 路径（21 个航路点只需要 80B 的协议开销），这对于实现从应用到应用的端到端 SLA 而言至关重要。

我们的理念是，与其努力弥合两个节奏不同的世界，不如让节奏快的发展的更快。为此，我们提出了“以应用为中心”的 Overlay & Underlay 整合方案，并在本系列文章的第三篇中做了相应的展示：Underlay 设备作为航路点，把 Underlay 能力提供给应用；应用根据 Underlay 设备提供的能力，生成 SRH 并对 Overlay 网关/VPP 进行编程，从而实现 Overlay 与 Underlay 的整合。

“以应用为中心”的优势是：

- 对于 Overlay：更灵活、更多的控制权、更大的扩展性
- 对于 Underlay：更简单、更高效、更容易利用现有网络

然而，不同客户对应用和网络的掌控力度有着很大的不同，这不完全是技术问题，还涉及到管理边界、运维习惯等因素。因此，在“以应用为中心”这个大框架下，我们设计了两种方案：集中式方案和分布式方案。集中式方案可能更容易被网络中已经规模部署了 Underlay SDN 控制器的客户接受，而分布式方案可能更容易被以云为中心的客户接受。需要说明的是，集中式方案和分布式方案并非是对立的，事实上集中式方案可以迁移至分布式方案，也可以在网络中结合使用集中式方案和分布式方案。对于 Overlay 网关而言，只是生成 SRv6 Policy 的来源不同罢了。

下面将详细介绍“以应用为中心”的 Overlay & Underlay 整合方案的架构与实现。

二、集中式/分布式方案架构

2.1 集中式方案架构

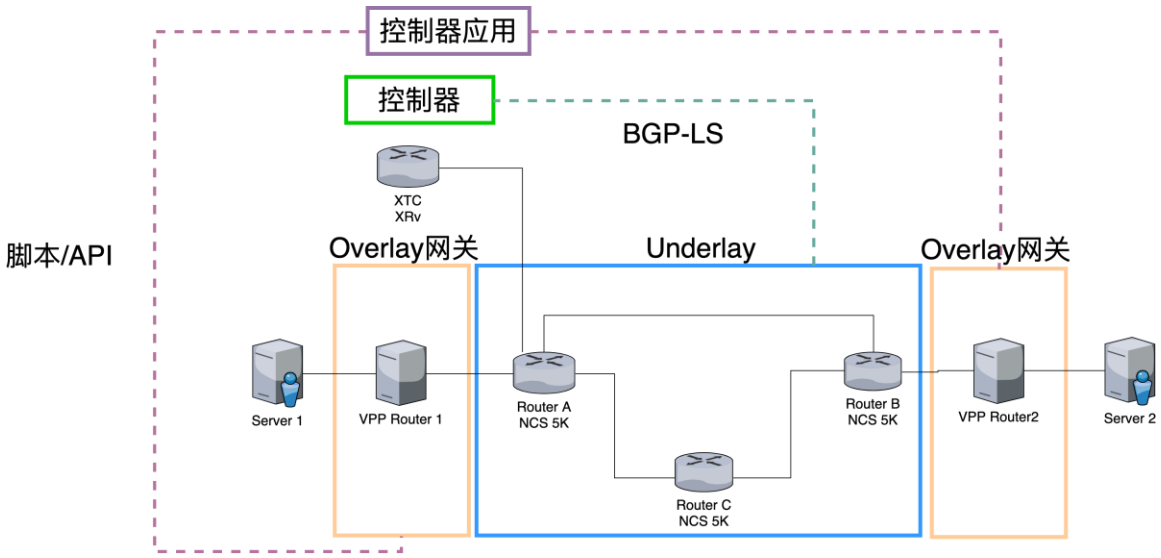

图 1 集中式方案架构

在本系列文章的第三篇中，我们已经详细介绍了集中式方案的架构与实现，这里我们只是简单回顾一下。

上图显示了集中式方案的典型架构。Underlay 部分由支持 SRv6 End 功能的设备组成，只充当航路点，不作为 SRv6 Policy 的头端(在测试时，采用了三台 Cisco NCS5500，组成一个环状拓扑结构)。Overlay 网关基于开源 VPP 开发，Overlay 网关同时作为 SRv6 Policy 的头端，负责将应用发出的 IPv4 流量封装入 SRv6 Policy 发往对端，同时对收到的流量执行 End.DX4 操作以去掉 IPv6 报头并执行 IPv4 转发。

控制器采用 Cisco XTC，XTC 通过 BGP-LS 得到全网拓扑结构，根据 SLA 要求计算出最优路径。上层控制器应用通过 Rest API 从控制器取到拓扑和最优路径信息，构造出相应的 SRv6 Policy，并通过 VPP API 下发给 Overlay 网关，从而实现动态的路径创建和控制。

可以看到在集中式方案中，所有的操作均依赖于集中的上层控制器应用对 Underlay 控制器(XTC)的调用，包括 Overlay SLA 需求的传递、Underlay 设备 SRv6 信息的采集、Underlay 网络发生状态触发的重算。

当海量的应用均需获得端到端 SLA 路径时，对 Underlay 控制器的高频调用可能会遇到性能瓶颈（第 5.2 节的性能测试证明了这一点）。因此，我们设计了分布式方案。

2.2 分布式方案架构

分布式方案引入了 etcd，作为存储和维护 Underlay 网络 SRv6 Policy 信息的分布式数据库，并作为 Overlay 和 Underlay 整合的桥梁。

分布式方案的核心思想如下图：

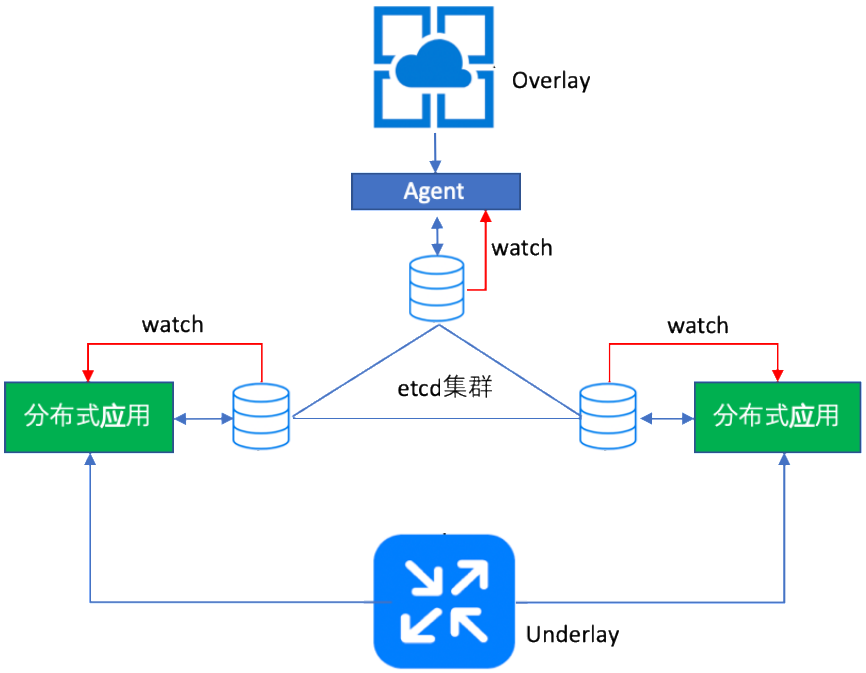

图 2 分布式方案的核心思想

在分布式方案中，有三个关键的变化：

1. 控制器应用变为分布式了，并且控制器应用不直接编程 Overlay 网关上的 SRv6 Policy，而是把计算所得的 SRv6 Policy 信息写入 etcd
2. 运行在 Overlay 网关上的控制代理通过 etcd 获取和监控 Underlay 的 SRv6 Policy 信息，并根据此信息编程 Overlay 网关上的 SRv6 Policy
3. Overlay 网关上的控制代理基于 etcd 的 watch 机制获知 Underlay 的变化，Overlay 网关可自行决定如何处理，例如重算还是回退到常规转发等

一个关键的事实是，在正常的 Underlay 网络中，发生路径重算的情况是很少的，可以重用的路径又是很多的。所以，分布式方案的本质是基于 etcd 建立了一个全网范围内的、超大容量、超高扩展性的“少写多读”的分布式数据库，这样 Overlay 网关上的控制代理或者其他应用就可以用应用开发人员非常熟悉的 etcd API 获得 Underlay SRv6 Policy 信息，以及通过 etcd watch 机制监控 Underlay SRv6 Policy，并独立地做出决策。这赋予了应用开发者高度的灵活性和快速的迭代能力，因为 Underlay 能力已经被抽象成一个个 etcd 数据库对象，而不再是难以把控的“黑盒子”。

在本文的后续章节中，将详细介绍我们的分布式方案实现示例。下图显示了此分布式方案实现示例的拓扑。在这里特地采用了与集中式方案一致的物理拓扑来解析分布式方案的实现，以说明分布式方案其实可以由集中式方案演变而来。

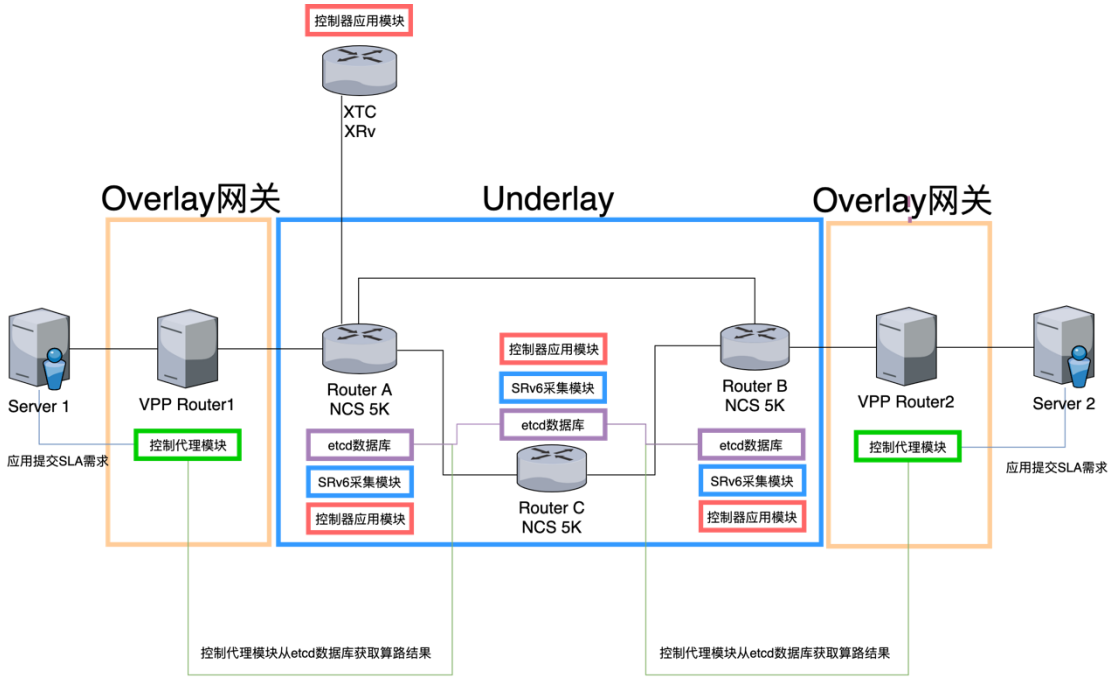

图 3 分布式方案实现示例拓扑

在我们的实现中，利用了 Cisco IOS XR 对第三方容器的支持能力，即分布式方案的各个组件容器都运行在 Cisco NCS5500 之上。

- 控制器应用模块容器负责接收算路请求，通过 Cisco XTC 的 API 获得 IPv4/Prefix-SID 形式的算路结果；模块读取 etcd 数据库中的全网 SRv6 Segment，将算路结果转换为 SRv6 Segment 列表并存储到数据库中，以供控制代理模块读取。
- SRv6 信息采集模块容器通过 gRPC 采集 Underlay 设备的 SRv6 Segment 等信息，并存储在 etcd 数据库中。
- etcd 数据库容器运行 etcd。

- 控制代理模块需要部署在 Overlay 网关(VPP)上。此模块通过读取配置文件得到应用的 SLA 需求，查询 etcd 数据库获取此 SLA 需求相应的最优路径，并部署 SRv6 Policy 到 Overlay 网关。

需要指出的是，图中的控制器应用模块可以部署在网络中的任意位置，既可以像本文介绍的实现示例一样作为容器部署在多台 Underlay 设备上，也可以部署在外置的主机上，控制器应用模块本身可以做到是无状态的，因此支持在多个控制器应用模块实例间实现负载均衡。etcd 数据库集群同样可以运行在 Cisco IOS XR 设备或者外置的主机上。

可以看到，在分布式架构设计中，整个系统可以不依赖于其中一台设备或者模块，每个模块均可水平扩展，因此扩展性非常好；使用了业界成熟的 etcd 数据库作为信息存储/推送模块，大大简化了程序设计。但分布式架构涉及到多个组件间的协同交互，因此实现比集中式方案复杂。

三、实现流程

3.1 集中式方案实现流程解析

详见本系列文章第三篇。

3.2 分布式方案实现流程解析

下面首先对运行在每台 Cisco NCS5500 上的容器进行简单的解析：

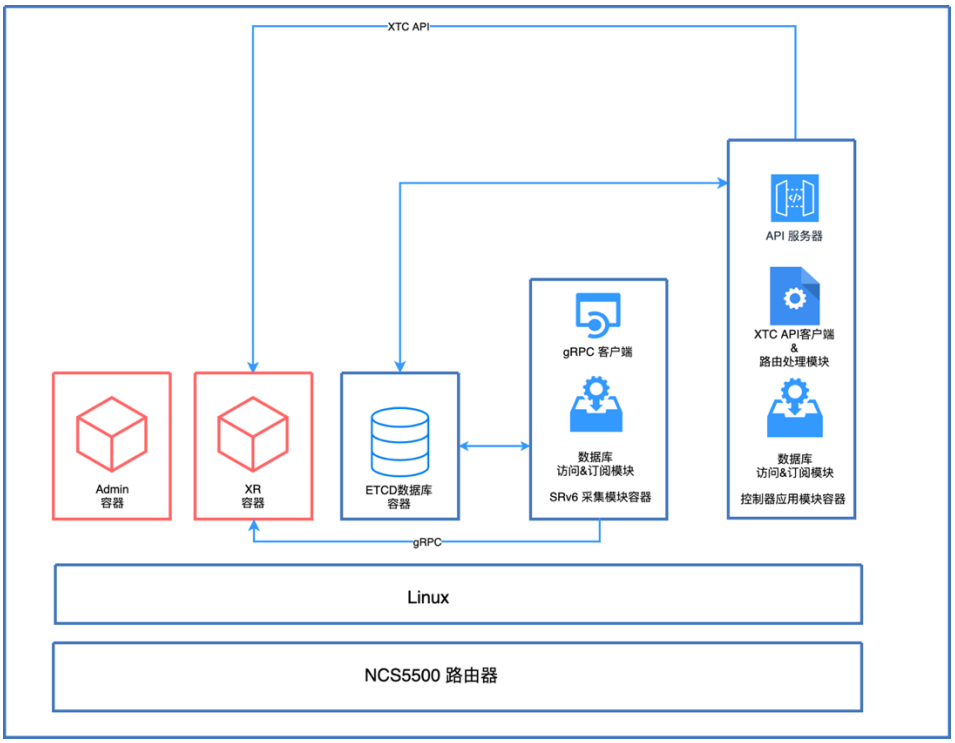

图 4 Cisco NCS5500 容器架构图

如上图所示，Cisco NCS5500 路由器操作系统基于 Linux，自带两个容器（LXC），即 Admin 容器以及 XR 容器。在自带的容器之外，我们部署了三个应用容器：

- etcd 数据库容器：作为整个分布式方案的中间件数据库。
- SRv6 数据采集模块容器：该容器内主要包括两个模块，一个是 gRPC 模块，主要负责通过 gRPC 访问 XR 容器，获取 SRv6 相关的信息；另一个是数据库访问/订阅模块，负责把定时抓取到的数据和 etcd 里存储的信息进行比较，如果不同则进行更新。
- SRv6 控制器模块容器：该容器包括一个 API 服务器用于提供北向 API 接口；以及一个 XTC API 客户端用于调用 XTC 进行算路；还包括一个数据库访问/订阅模块用于从 etcd 读取全网 SRv6 信息，并存储算路结果。

下图显示了分布式方案实现示例的总体运行逻辑：

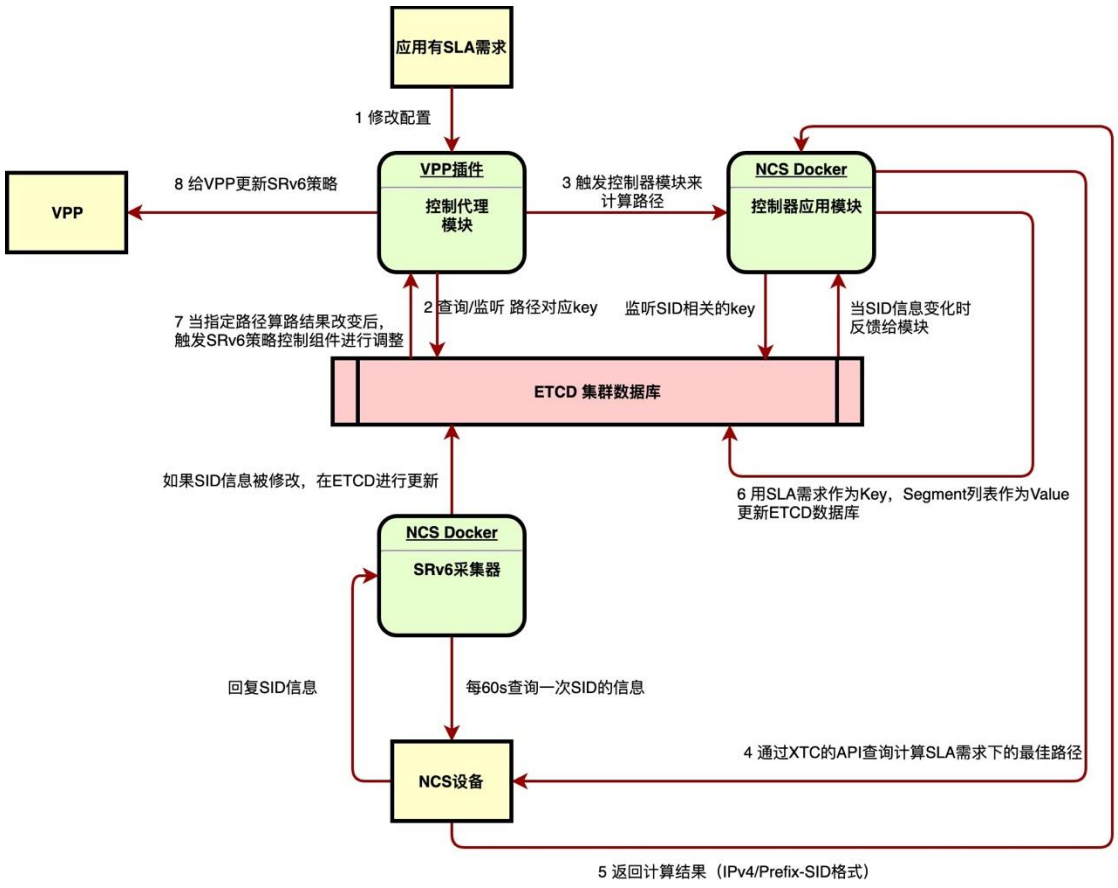

图 5 分布式实现示例总体运行逻辑

如上图，当应用有 SLA 需求时，会首先修改控制代理模块的配置（❶）。控制代理模块会尝试获取并监听 etcd 数据库中关于该 SLA 需求的算路结果（❷）。。

如果没有已有的算路结果的话，控制代理模块就会通过 API 触发控制器模块算路（❸）。控制器应用模块会通过 XTC API 访问 XTC 网络控制器（❹）。控制器应用模块获得 XTC 返回 Prefix-SID 格式的算路结果（❺）。控制器应用模块根据自己从 etcd 数据库读取到的全网 SRv6 信息，将 Prefix-SID 信息转换为 Segment 列表，存储回数据库（❻）。

当算路结果更新后，控制代理模块即会收到推送（❼）。最后通过代码下发/更新相应的 SRv6 Policy 给 VPP（❽）。

在这个流程中，各个模块之间使用了 etcd 数据库的 watch 机制，因此当网络状态发生变化时，可以立刻更新相关 SRv6 Policy。

下面我们详细解析各模块的运行流程。

3.2.1 SRv6 采集模块

SRv6 采集模块的具体代码在：https://github.com/ljm625/xr-srv6-etcd

SRv6 采集模块主要通过 gRPC，访问 Cisco NCS5500 设备，获取 SRv6 对应的 YANG 信息，需要注意的是当前的 YANG 实现中只包含设备本地的信息，因此需要在每台 Cisco NCS5500 设备上部署此容器。

具体的模块运行流程如下图：

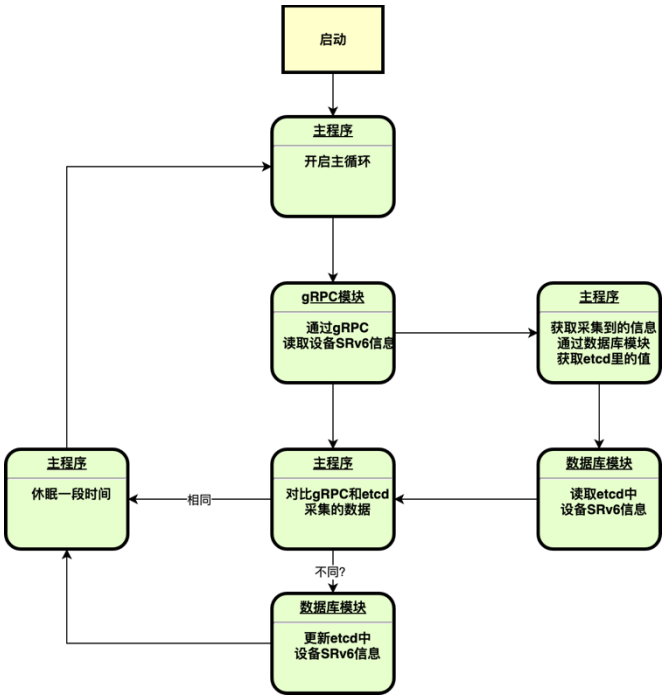

图 6 SRv6 采集模块运行流程

SRv6 模块每隔一段时间通过 gRPC 采集设备本地的 SRv6 相关信息，并存储到 etcd 集群数据库中。

3.2.2 控制器应用模块

控制器应用模块的具体代码见：https://github.com/ljm625/xr-srv6-controller

控制器应用模块是整个程序中最复杂的模块，包含四个部分：

- API 服务器
  - 提供算路 API 及查询全网 Underlay 端点设备的 API。
- etcd 数据库模块
  - 主要负责读取/写入 etcd 数据库的信息。
  - 还负责订阅全网设备的 SRv6 信息，如果发生变化，etcd 会推送给本模块。
- XTC 客户端
  - 主要负责通过 XTC API 获取算路结果。
- 算路模块
  - 执行主要的算路逻辑，根据 XTC 算路结果，把表示为 IPv4 地址的 Segment 转换为 SRv6 Segment（XTC 尚不支持 SRv6 Policy 计算）。

具体的算路流程如下图所示：

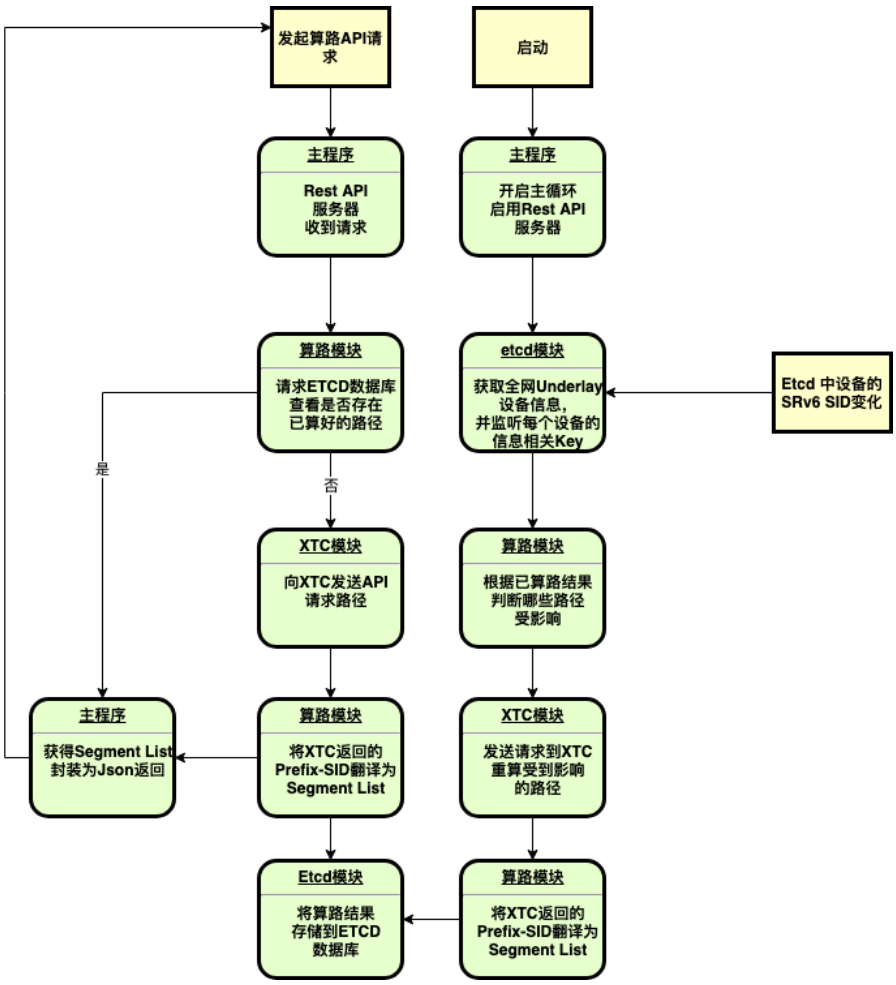

图 7 控制器应用模块运行流程

如上图，根据是否有算路的历史结果，算路模块会执行不同的策略，并且在网络状态发生变化时，会自动触发控制器模块重新计算影响到的路径。

3.2.3 控制代理模块

控制代理模块具体代码在：https://github.com/ljm625/srv6-vpp-controller

该模块主要负责读取 Overlay 应用的 SLA 请求信息，以及 VPP 的本地 SID 表配置，需要对本地 VPP 的 SID 进行配置，并将相关信息上传到 etcd。他还从 etcd 处获得算路结果以及对端 VPP 指定 VRF 的 End.DX4/DX6 SID 信息，并通过 VPP API 下发相应的 SRv6 Policy。

具体的逻辑流程如下图：

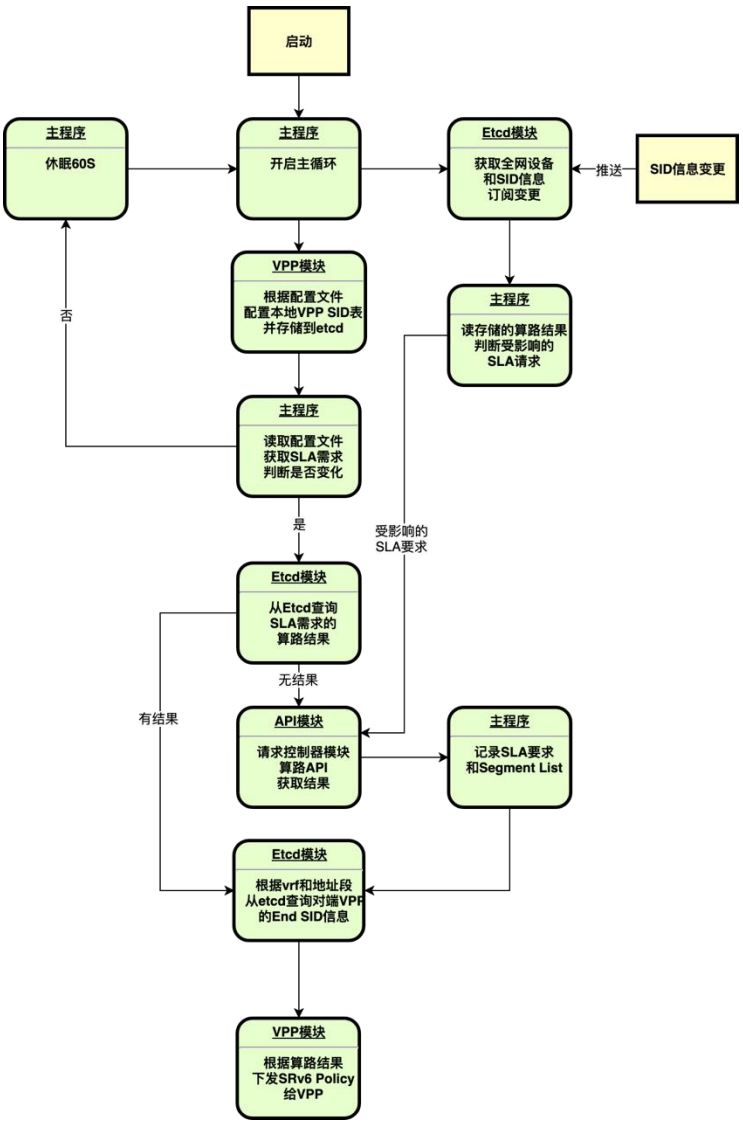

图 8 控制代理模块运行流程

程序主要监听配置文件的变化（SLA 需求的变更）以及算路结果的变化。

当配置文件发生变化时，模块会自动判断出需要增/删的路径，算路结果发生变化之后，则说明网络出现变动，也会触发该模块自动修改 Overlay 网关的 SRv6 Policy，动态作出相应调整。

四、实现示例的部署

4.1 集中式方案实现示例的部署

详见本系列文章第三篇。

4.2 分布式方案实现示例的部署

主要采用 Docker 进行部署，相关容器均已上传到 Docker Hub。

4.2.1 etcd 数据库部署

etcd 数据库部署在多台 Cisco NCS5500 中，容器的相关代码见：
https://github.com/ljm625/ios-xr-etcd

安装很简单，分为单点部署和多点部署两种情况。

单点部署：

```
bash

mkdir /misc/app_host/etcd
export DATA_DIR=/misc/app_host/etcd
export IP=Cisco NCS5500 的 TPA IP 地址，如 172.20.100.150

docker run
  -itd \
  --cap-add=SYS_ADMIN \
  --cap-add=NET_ADMIN \
  -v /var/run/netns:/var/run/netns \
  --volume=${DATA_DIR}:/etcd-data \
  --name etcd ljm625/etcd-ios-xr:latest \
  /usr/local/bin/etcd \
  --data-dir=/etcd-data --name node1 \
  --initial-advertise-peer-urls http://${IP}:2380 \
  --listen-peer-urls http://${IP}:2380 \
  --advertise-client-urls http://${IP}:2379 \
   --listen-client-urls http://${IP}:2379 \
  --initial-cluster node1=http://${IP}:2380
```

执行后即可完成部署。

图 9 etcd 数据库容器正常运行

查看容器日志，可以看到 etcd 数据库容器正常运行。

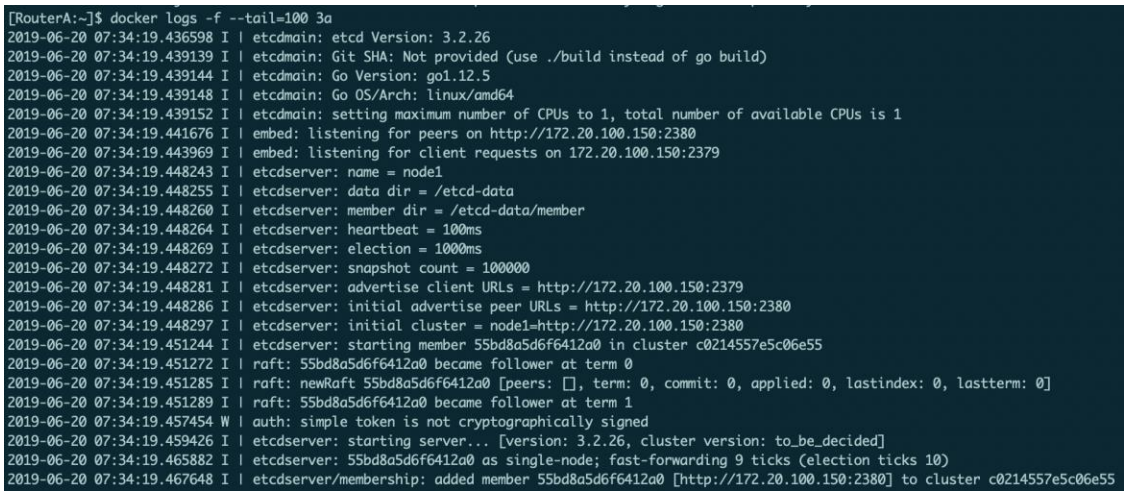

图 10 etcd 数据库容器运行日志

集群部署：

在集群部署中，我们把 etcd 容器部署到三台 Cisco NCS5500 上，使用的镜像相同。

第一台 Cisco NCS5500：

```
mkdir /misc/app_host/etcd
export DATA_DIR=/misc/app_host/etcd
export IP=你的 NCS 的 TPA IP 地址，如 172.20.100.150
export NAME=本机的节点名称，要和下面的相同，如 etcd-node-0 （即节点 1）
NAME_1=etcd-node-0    //集群节点 1 名称
NAME_2=etcd-node-1    //集群节点 2 名称
NAME_3=etcd-node-2    //集群节点 3 名称
HOST_1=172.20.100.150  //集群节点 1 IP
HOST_2=172.20.100.151  //集群节点 2 IP)
HOST_3=172.20.100.152  //集群节点 3 IP

CLUSTER_STATE=new //集群状态，默认即可
TOKEN=my-etcd-token //集群的通信密钥，任意内容，需要保持一致

CLUSTER=${NAME_1}=http://${HOST_1}:2380,${NAME_2}=http://${HOST_2}:2380,
${NAME_3}=http://${HOST_3}:2380

docker run
  -itd \
  --cap-add=SYS_ADMIN \
  --cap-add=NET_ADMIN \
  -v /var/run/netns:/var/run/netns \
```

```
  --volume=${DATA_DIR}:/etcd-data \
  --name etcd ljm625/etcd-ios-xr:latest \
  /usr/local/bin/etcd \
  --data-dir=/etcd-data --name ${NAME} \
  --initial-advertise-peer-urls http://${IP}:2380 \
  --listen-peer-urls http://${IP}:2380 \
  --advertise-client-urls http://${IP}:2379 \
   --listen-client-urls http://${IP}:2379 \
  --initial-cluster ${CLUSTER} \
  --initial-cluster-state ${CLUSTER_STATE} --initial-cluster-token
${TOKEN}
```

第二台 Cisco NCS5500：

```
mkdir /misc/app_host/etcd
export DATA_DIR=/misc/app_host/etcd
export IP=你的 NCS 的 TPA IP 地址，这里为 172.20.100.151
export NAME=本机的节点名称，要和下面的相同，如 etcd-node-1 （即节点 2）
NAME_1=etcd-node-0   //集群节点 1 名称
NAME_2=etcd-node-1   //集群节点 2 名称
NAME_3=etcd-node-2   //集群节点 3 名称
HOST_1=172.20.100.150  //集群节点 1 IP
HOST_2=172.20.100.151  //集群节点 2 IP
HOST_3=172.20.100.152  //集群节点 3 IP

CLUSTER_STATE=new //集群状态，默认即可
TOKEN=my-etcd-token //集群的通信密钥，任意内容，需要保持一致

CLUSTER=${NAME_1}=http://${HOST_1}:2380,${NAME_2}=http://${HOST_2}:2380,
${NAME_3}=http://${HOST_3}:2380

docker run
  -itd \
  --cap-add=SYS_ADMIN \
  --cap-add=NET_ADMIN \
  -v /var/run/netns:/var/run/netns \
  --volume=${DATA_DIR}:/etcd-data \
  --name etcd ljm625/etcd-ios-xr:latest \
  /usr/local/bin/etcd \
  --data-dir=/etcd-data --name ${NAME} \
  --initial-advertise-peer-urls http://${IP}:2380 \
  --listen-peer-urls http://${IP}:2380 \
```

```
  --advertise-client-urls http://${IP}:2379 \
   --listen-client-urls http://${IP}:2379 \
  --initial-cluster ${CLUSTER} \
  --initial-cluster-state ${CLUSTER_STATE} --initial-cluster-token
${TOKEN}
```

第三台 Cisco NCS5500：

```
mkdir /misc/app_host/etcd
export DATA_DIR=/misc/app_host/etcd
export IP=你的 NCS 的 TPA IP 地址，这里为 172.20.100.152
export NAME=本机的节点名称，要和下面的相同，如 etcd-node-0 （即节点 3）
NAME_1=etcd-node-0    //集群节点 1 名称
NAME_2=etcd-node-1    //集群节点 2 名称
NAME_3=etcd-node-2    //集群节点 3 名称
HOST_1=172.20.100.150  //集群节点 1 IP
HOST_2=172.20.100.151  //集群节点 2 IP
HOST_3=172.20.100.152  //集群节点 3 IP

CLUSTER_STATE=new //集群状态，默认即可
TOKEN=my-etcd-token //集群的通信密钥，任意内容，需要保持一致

CLUSTER=${NAME_1}=http://${HOST_1}:2380,${NAME_2}=http://${HOST_2}:2380,
${NAME_3}=http://${HOST_3}:2380

docker run
  -itd \
  --cap-add=SYS_ADMIN \
  --cap-add=NET_ADMIN \
  -v /var/run/netns:/var/run/netns \
  --volume=${DATA_DIR}:/etcd-data \
  --name etcd ljm625/etcd-ios-xr:latest \
  /usr/local/bin/etcd \
  --data-dir=/etcd-data --name ${NAME} \
  --initial-advertise-peer-urls http://${IP}:2380 \
  --listen-peer-urls http://${IP}:2380 \
  --advertise-client-urls http://${IP}:2379 \
   --listen-client-urls http://${IP}:2379 \
  --initial-cluster ${CLUSTER} \
  --initial-cluster-state ${CLUSTER_STATE} --initial-cluster-token
${TOKEN}
```

执行后即完成了部署。

访问 etcd 的数据库，使用默认的端口 2379 即可。

4.2.2 SRv6 采集模块部署

SRv6 的 YANG 采集模块通过 gRPC 抓取 YANG 模型对应的 SRv6 部分的内容，获取设备本地的 SRv6 Segment 等信息。由于当前的 YANG 实现里只有设备本地的 SRv6 信息，因此需要在全网每台 Cisco NCS5500 上通过 docker 部署该模块。

首先需要打开 Cisco NCS5500 的 gRPC 功能：

```
grpc
 port 57777
 no-tls
 address-family ipv4
!
```

注意的是，这里的端口 57777 需要和之后的配置对应上，并关闭 TLS 功能。

然后进入 Cisco NCS5500 的 Linux Bash，启动 docker 容器，容器相关的代码见：https://github.com/ljm625/xr-srv6-etcd

需要注意的是，在启动该容器时需要正确配置以下参数：

- gRPC port: -g，需要和之前配置的一致，这里是 57777
- gRPC IP：-z，一般为 127.0.0.1，即本机 127.0.0.1
- gRPC username: -u，和 ssh 登陆的账户名一致，这里为 admin
- gRPC password: -p，和 ssh 登陆的密码一致，这里为 admin
- etcd IP: -i，指定 etcd 数据库的 IP 地址，这里为 172.20.100.150
- etcd Port: -e, 指定 etcd 数据库的端口，这里为默认端口 2379

执行的命令如下：

```
docker pull ljm625/xr-srv6-etcd:yang

docker run -itd   --cap-add=SYS_ADMIN   --cap-add=NET_ADMIN \
  -v /var/run/netns:/var/run/netns ljm625/xr-srv6-etcd:yang \
   -g 57777 -u admin -p admin -i 172.20.100.150 -e 2379 -z 127.0.0.1
```

第一行为下载 docker 镜像，第二行根据给定的参数启动容器。

启动成功后可以查看日志：

```
[{'allocation-type': 'dynamic',
  'create-timestamp': {'age-in-nano-seconds': '2744152',
                       'time-in-nano-seconds': '1559307681197197295'},
  'function-type': 'end-with-psp',
  'has-forwarding': True,
  'locator': 'PE_2',
  'locator-name': 'PE_2',
  'owner': [{'owner': 'sidmgr'}],
  'sid': 'fc00:a:1:0:1::',
  'sid-context': {'application-data': '00:00:00:00:00:00:00:00',
                  'key': {'e': {'opaque-id': 1, 'table-id': 3766484992},
                          'sid-context-type': 'end-with-psp'}},
  'sid-opcode': 1,
  'state': 'in-use'},
 {'allocation-type': 'dynamic',
  'create-timestamp': {'age-in-nano-seconds': '2744052',
                       'time-in-nano-seconds': '1559307781269197295'},
  'function-type': 'end-x-with-psp',
  'has-forwarding': True,
  'locator': 'PE_2',
  'locator-name': 'PE_2',
  'owner': [{'owner': 'isis-1'}],
  'sid': 'fc00:a:1:0:40::',
  'sid-context': {'application-data': '00:00:00:00:00:00:00:00',
                  'key': {'sid-context-type': 'end-x-with-psp',
                          'x': {'interface': 'GigabitEthernet0/0/0/0',
                                'is-protected': False,
                                'nexthop-address': 'fe80::520a:ff:fe24:3',
                                'opaque-id': 0}}},
  'sid-opcode': 64,
  'state': 'in-use'}]
end-with-psp - fc00:a:1:0:1::
end-x-with-psp - fc00:a:1:0:40::
[*] Etcd Version : 3.2.26
[*] Updating SID Info
```

图 11 SRv6 采集模块容器运行日志

如上图输出所示，即表示启动成功，请注意需要在每台 Cisco NCS5500 上启动这个容器。

除此之外，还有另外一种选择是使用 SRv6 CLI 抓取模块进行部署。

SRv6 的 CLI 抓取模块通过 gRPC 执行 CLI 命令，通过查询 ISIS 数据库，获取全网设备的 SRv6 Segment 等信息。由于该部分数据暂时不存在于 YANG 中，因此目前只能通过 CLI 访问到。全网的设备均需要开启 ISIS，并分发自身的 SRv6 Segment 信息。该模块只需要在全网任意一台或多台 Cisco NCS5500 上通过 docker 部署即可。

部署步骤和运行方式，与前文完全一致，这里不再赘述，只需要将 ljm625/xr-srv6-etcd:yang 镜像替换为 ljm625/xr-srv6-etcd:cli 即可。

4.2.3 控制器应用模块部署

第三个需要部署在 Cisco NCS5500 的模块为控制器应用模块，该模块可以部署在一台或者多台 Cisco NCS5500 上。

容器的相关代码见：https://github.com/ljm625/xr-srv6-controller

首先需要配置 Cisco NCS5500 作为 PCE（XTC）。

```
pce
 address ipv4 172.20.100.150
 rest
  user cisco
  !
  authentication basic
 !
 logging
  no-path
  fallback
 !
!
```

接着进入 Cisco NCS5500 的 bash，启动对应的容器，其中有一些参数需要进行配置：

- XTC IP : XTC 的 IP，如果是本机的话为 TPA 地址，这里为 172.20.100.150
- XTC Username : XTC 的登陆用户名，这里为 cisco
- XTC Password : XTC 的登陆密码，这里为 cisco
- etcd IP : etcd 的 IP 地址，这里为 172.20.100.150
- etcd Port :etcd 的端口信息，这里为 2379

```
docker pull ljm625/xr-srv6-controller

docker run -itd --cap-add=SYS_ADMIN --cap-add=NET_ADMIN -v
/var/run/netns:/var/run/netns ljm625/xr-srv6-controller -u cisco -p
cisco -i 172.20.100.150 -e 2379 -x 172.20.100.150
```

启动控制器应用模块容器之后可以查看日志：

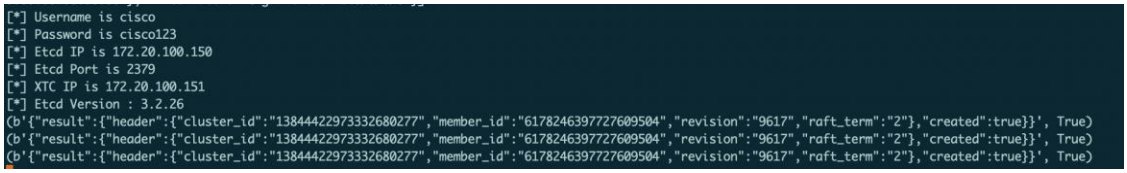

图 12 控制器应用模块容器运行日志

看到如上图的日志说明启动成功，并正常运行。

4.2.4 控制代理模块部署

最后一个模块安装在 Overlay 网关/VPP 上，读取应用要求的 SLA 信息，算路并动态下发 SRv6 Policy 给 VPP。

该模块无法通过 docker 进行部署，并且必须和 Overlay 网关/VPP 部署在同一 Linux 上。

4.2.4.1 安装 Python3 运行环境

该模块要求 Python3.6 以上的环境运行，这里以 Ubuntu 16.04 LTS 版本作为示例：

添加 python3.6 的源：

```
sudo add-apt-repository ppa:jonathonf/python-3.6
```

安装 python3.6:

```
sudo apt-get update
sudo apt-get install python3.6
```

调整 python3 优先级，使得 python3.6 优先级较高：

```
sudo update-alternatives --install /usr/bin/python3 python3
/usr/bin/python3.5 1
sudo update-alternatives --install /usr/bin/python3 python3
/usr/bin/python3.6 2
```

4.2.4.2 安装配置 VPP Python Script

首先执行：

```
sudo apt-get install vpp-plugin-core vpp-plugin-dpdk vpp-dev vpp-api-
python
```

安装一些必备的模组，其中包括 vpp 的 api json 文件，这个是 api 所必须的。

接着安装 vpp 在 python 下的 api 包，vpp_papi，其实在上一步，我们已经自动通过 deb 安装了 2.7 版本的 vpp-papi，但适用于 python 3.x 版本的依赖需要自己手动安装。

执行：

```
git clone https://github.com/FDio/vpp
cd vpp
git checkout stable/1904  //这里需要和自己本机的 vpp 版本一致，这里为 19.04 版，所以输入 1904
cd src/vpp-api/python
sudo python3 setup.py install
```

完成 vpp_papi 依赖的安装。

4.2.4.3 安装配置控制代理模块

```
git clone https://github.com/ljm625/xr-srv6-controller
cd xr-srv6-controller
python3 -m pip install -r requirements.txt
```

接下来配置 config.json 文件：

```
{
  "config": {
    "etcd_host": "172.20.100.150",
    "etcd_port": 2379,
    "controller_host": "172.20.100.150",
    "controller_port": "9888",
    "bsid_prefix": "fc00:1:999::"
  },
  "sla": [
    {
      "dest_ip":"10.1.1.0/24",
      "source":"RouterA",
      "dest": "RouterB",
      "method": "latency",
      "extra": {},
      "vrf_name": "c1"

    }
  ],
  "sid": [
    {
      "interface": "GigabitEthernet0/4/0",
```

```
      "action": "end.dx4",
      "gateway": "10.0.1.1",
      "ip_range": "10.0.1.0/24",
      "vrf_name": "c1"

    }
  ]

}
```

里面有几个参数：

- etcd_host/etcd_port: etcd 数据库的 IP 和端口信息
- controller_host/controller_port: 控制器应用模块的 IP 地址和端口，需要注意的是这个模块默认端口即为 9888
- bsid_prefix: 本机 VPP SRv6 Policy 的 BSID 前缀范围，新创建的 SRv6 Policy 都会从这个前缀范围内分配 BSID。
- sla：sla 部分为应用的 SLA 需求，为一个数组，可以包含多个 SLA 需求
  - source：起始设备
  - dest：目的设备
  - method：算路依据，有三种，分别是 latency/te/igp
  - extra：额外信息，例如避免使用特定资源（节点、链路、SRLG）等，目前暂未实现，为预留的扩展字段
  - dest_ip：需要访问的对端 Overlay 的网段信息，可以为 IPv4 或者 IPv6 地址段
  - vrf_name：该规则对应的 VRF 名称，用于隔离多个 VPN。需要和对端一致。
- sid：sid 部分为 VPP 节点的本地 SID 信息，主要是 VPP/Overlay 网关定义的 End.DX4 或者 End.DX6 信息，可以包含多个需要定义的 SID 信息
  - action：SID 对应的 action，可以为 End.DX4 或者 End.DX6
  - interface：End.DX4 和 End.DX6 解封装之后包将通过该网卡转发
  - gateway：End.DX4 和 End.DX6 解封装之后将包转发给该网关设备
  - ip_range：该网关下设备的 IP 段。

- vrf_name：该 Segment ID 对应的 VRF 名称，用于隔离多个 VPN。需要和对端 VPP 的 SLA 部分配置一致。

在该模块运行过程中，应用可以根据需求修改配置文件的 sla 部分，从而动态调整 SLA 规则。

修改好配置文件之后即可执行代码：

```
python3 main.py
```

图 13 控制代理模块容器运行日志

看到类似上图的输出即表示模块正常运行。

4.2.5 运行

在上述部署完成之后，实际上分布式方案实现示例的各个模块已经在运行。

首先可以通过检查各个模块的日志来查看运行状态。

- SRv6 采集模块：

图 14 SRv6 采集模块容器采集到设备 SRv6 信息

如上图，SRv6 采集模块容器的日志输出显示，此模块不断获取到设备本地的 SRv6 相关信息，并存储到 etcd 数据库，说明运行正常。

- 控制器应用模块

对于控制器应用模块，可以使用日志确认工作状态，并使用 API 来测试运行状态。

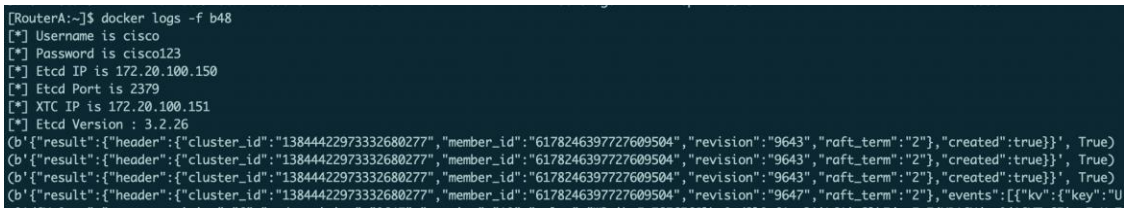

图 15 控制器应用模块容器正常运行，订阅 Underlay 设备信息

如上图，表示运行正常，并且已经订阅了网内设备 SRv6 信息对应的键值(Key)。

可以用 Postman 测试控制器的功能以及 API：

获取设备 API：

GET http://:9888/api/v1/devices

返回：

```
["RouterA", "SR-PCE", "RouterB"] //网内的 Underlay 设备列表
```

计算路径：

POST http://:9888/api/v1/calculate

JSON Payload：

```
{
        "source":"RouterA", //发端设备
        "dest":"SR-PCE",    //收端设备
        "method":"latency", //选路策略，可选 latency,te,igp 三种
        "extra":{}          //额外要求，如避免指定资源等，目前暂未实现
}
```

返回：

```
{
    "result": "success",  //获取成功
    "sid_list": ["fc00:c:1:0:1::"] //算路结果 Segment 列表
}
```

当这两个 API 可以正常访问，则表示控制器应用模块运行正常。

- 控制代理模块

对于控制代理模块，首先可以检查其运行日志：

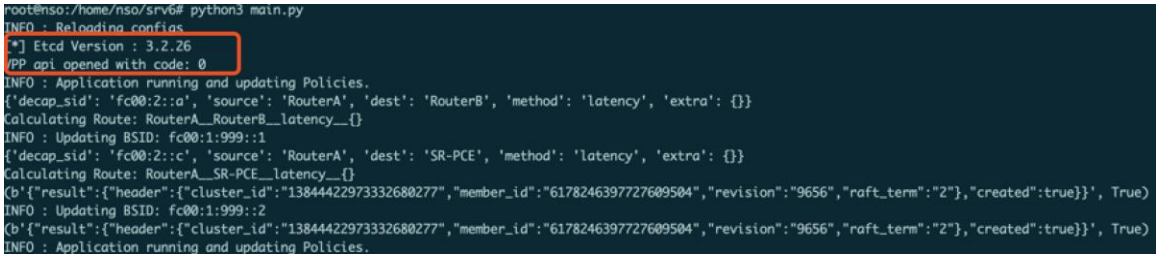

图 16 控制代理模块容器正常运行，算路并下发 SRv6 Policy

如上图，可以看到控制代理成功连接到 etcd 以及 Overlay 网关/VPP，并开始根据配置文件的 SLA 部分开始算路。

下面添加一个新的 SLA 需求到控制代理模块的配置文件中，此时保持控制代理模块在后台运行。

```
{
  "config": {  // 控制器相关信息
    "etcd_host": "172.20.100.150",
    "etcd_port": 2379,
    "controller_host": "172.20.100.150",
    "controller_port": "9888",
    "bsid_prefix": "fc00:1:999::"
  },
  "sla": [  // SLA 需求部分
    {
      "dest_ip":"10.1.1.0/24 ",
      "source":"RouterA",
      "dest": "RouterB",
      "method": "latency",
      "extra": {},
      "vrf_name": "c1"
    },
    {
      "dest_ip":"10.2.1.0/24 ",
      "source":"RouterA",
```

```
      "dest": "SR-PCE",
      "method": "latency",
      "extra": {},
      "vrf_name": "c1"
    },
    // 新添加的 SLA 需求
    {
      "dest_ip":"10.2.2.0/24 ",
      "source":"RouterB",
      "dest": "SR-PCE",
      "method": "latency",
      "extra": {},
      "vrf_name": "c1"
    },
    // 新添加的 SLA 需求
  "sid": [
    {
      "interface": "GigabitEthernet0/4/0",
      "action": "end.dx4",
      "gateway": "10.0.1.1",
      "ip_range": "10.0.1.0/24",
      "vrf_name": "c1"
    }
  ]

  ]
}
```

```
INFO : Application running and updating Policies.
INFO : Application running and updating Policies.
INFO : Application running and updating Policies.
{'decap_sid': 'fc00:3::a', 'source': 'RouterB', 'dest': 'SR-PCE', 'method': 'latency', 'extra': {}}
Calculating Route: RouterB__SR-PCE__latency__{}
INFO : Updating BSID: fc00:1:999::3
(b'{"result":{"header":{"cluster_id":"13844422973332680277","member_id":"6178246397727609504","revision":"9656","raft_term":"2"},"created":true}}', True)
INFO : Application running and updating Policies.
```

图 17 当添加了新的 SLA 需求，控制代理模块计算新路径

如上图，当修改了配置文件之后，模块自动读取到更改，计算新添加的从 RouterB 去往 SR-PCE 的低延迟路径。

在 VPP 上可以看到新下发的 SRv6 Policy：

```
vppctl show sr policies
```

```
root@nso:/home/nso/srv6# vppctl show sr policies
SR policies:
[0].-   BSID: fc00:1:999::1
        Behavior: Encapsulation
        Type: Default
        FIB table: 0
        Segment Lists:
        [0].- < fc00:b:1:0:1::, fc00:2::a > weight: 1
-----------
[1].-   BSID: fc00:1:999::2
        Behavior: Encapsulation
        Type: Default
        FIB table: 0
        Segment Lists:
        [1].- < fc00:c:1:0:1::, fc00:3::a > weight: 1
-----------
[2].-   BSID: fc00:1:999::3
        Behavior: Encapsulation
        Type: Default
        FIB table: 0
        Segment Lists:
        [2].- < fc00:b:1:0:1::, fc00:c:1:0:1::, fc00:2::c > weight: 1
```

图 18 Overlay 网关/VPP 上的 SRv6 Policy

如上图所示，控制代理模块成功下发了满足应用 SLA 需求的 SRv6 Policy。

接下来测试当网络发生变化时控制器应用模块能自动更新 SRv6 Policy。

我们继续之前的测试，首先查看 SRv6 采集模块采集的信息：

```
end-with-psp - fc00:a:1:0:1::
end-x-with-psp - fc00:a:1:0:40::
end-x-with-psp - fc00:a:1:0:41::
```

图 19 路由器 A 上的 SRv6 采集模块结果

接着关闭路由器 A 上与路由器 C 直连的端口：

```
RP/0/RP0/CPU0:RouterA#config
Fri Jul 12 03:52:10.906 UTC
RP/0/RP0/CPU0:RouterA(config)#interface GigabitEthernet 0/0/0/
0/0/0/0  0/0/0/1  0/0/0/2  0/0/0/3
RP/0/RP0/CPU0:RouterA(config)#interface GigabitEthernet 0/0/0/1
RP/0/RP0/CPU0:RouterA(config-if)#shutdown
RP/0/RP0/CPU0:RouterA(config-if)#commit
Fri Jul 12 03:52:18.352 UTC
```

图 20 关闭路由器 A 上与路由器 C 直连的端口

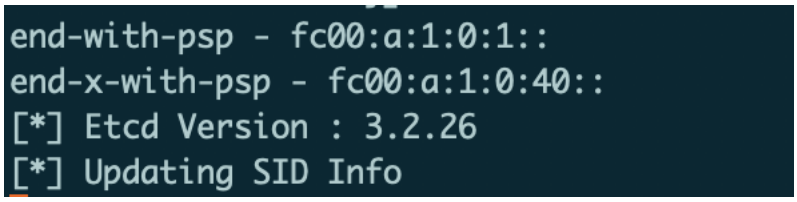

图 21 路由器 A 上的 SRv6 采集模块结果

如图 21 所示，可以看到 SRv6 采集模块采集到的信息发生了变化，接着 SRv6 采集模块将自动更新 etcd 里存储的 SRv6 信息。由于控制代理模块订阅了相应的键值，因此在算路结果发生变化后将收到 etcd 推送信息。

图 22 控制代理模块收到 etcd 推送信息

然后，控制代理模块更新相应的 SRv6 Policy。此时在 Overlay 网关/VPP 上使用 show sr policies，可以看到 Segment 列表已经自动完成了更新。

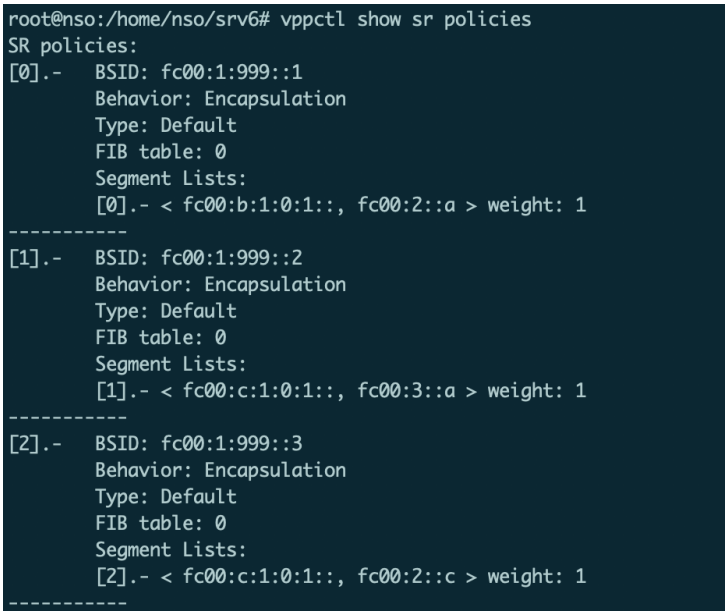

图 23 Overlay 网关/VPP 上的 SRv6 Policy 实现了自动更新

五、性能测试

下面将对分布式方案和集中式方案做一个简单的性能测试和对比。

测试软件使用的是开源软件 K6（https://k6.io）。

测试例为模拟 300 个虚拟用户，在 100 秒的固定时间内，不停访问 API 请求算路，相当于对 API 做压力测试。

对于集中式方案实现示例，由于每次请求都是直接从 XTC 取数据，因此这里简化为直接访问 XTC API。XTC 使用 XRv 9000 虚拟机，配置为 4vCPU，8G RAM。

对于分布式方案实现示例，我们对控制器应用模块 API 进行压力测试。

测试结果如下：

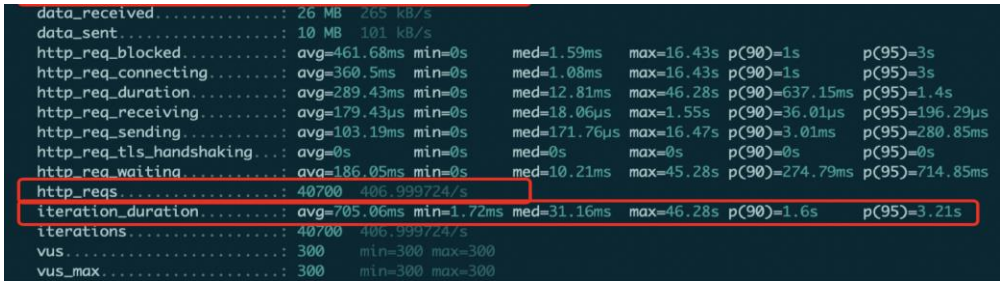

图 24 集中式方案压力测试结果

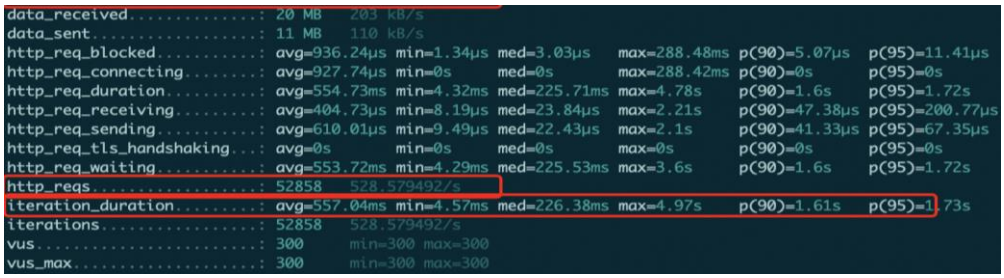

图 25 分布式方案压力测试结果-多实例

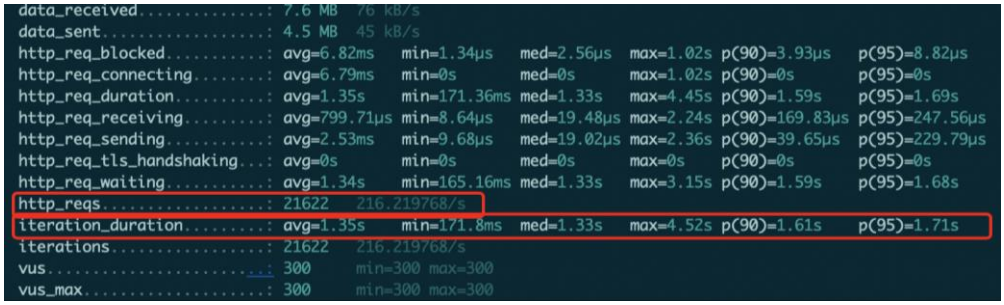

图 26 分布式方案压力测试结果-单实例

我们对比了三种场景，如图 24-26 所示，第一种为直接访问 XTC 的算路 API（集中式），第二种为访问三个负载均衡的控制器应用模块实例（分布式），第三种为访问单个控制器应用模块实例（分布式）的测试结果。

http_reqs 代表在 100 秒内总共处理的请求数，数字越大代表能力越强，可以看到 XTC 的处理能力相当不错，但与负载均衡的多个控制器应用模块相比，性能还是有着明显差距。

iteration_duration 代表每个 API 请求的平均返回时间，可以看到多个控制器应用模块实例<XTC<单个控制器应用模块实例。

对比结果	XTC	分布式方案（单实例）	分布式方案（多实例）
请求数量	40700	21622	52858
平均请求时延	705ms	1350ms	557ms

表 1 API 压力测试结果比较

具体的对比情况见表 1，可以看到即使是单实例的分布式方案，也能很好地在高负载环境下运行，虽然单个实例的请求处理能力受限，但当我们通过增加实例数量水平扩展后，分布式方案的性能可以获得成比率的提升。

本次测试拓扑只有三台设备，SRTE 数据库条目很少。而在真实生产环境中，SRTE 数据库条目数以万计，此时 XTC 在对大容量 SRTE 数据库进行大量计算时性能不可避免会下降。相反地，分布式方案由于可快速水平扩展，并使用 etcd 作为缓存，因此理论上可以提供更好的性能。

六、总结与展望

传统的 Underlay 与 Overlay 整合解决方案是“以网络为中心”。

本文提出的方案则是“以应用为中心”，即 Underlay 设备只需要将 SLA 能力提供给 etcd，由应用负责根据 Underlay SLA 能力采用 SRv6 编码路径。

相较于“以网络为中心”的解决方案，“以应用为中心”的解决方案可以让 Overlay 和 Underlay 独立进行演进，Underlay 不再成为 Overlay 的瓶颈，Overlay 也不会把复杂性引入 Underlay。“以应用为中心”的解决方案允许应用快速地迭代，自主控制 SLA 策略，非常有利于提供细颗粒度的差异化服务，这无疑对于提升云业务的竞争力大有裨益。

SRv6 支持“以网络为中心”和“以应用为中心”两种解决方案，用户可以灵活选择。我们认为，“以应用为中心”的解决方案未来将占据更为重要的位置，这是由于应用和网络具有不同的迭代速度所决定的。

Between overlay & underlay, there is a BRIDGE!

注：本文相关的代码均已上传至 Github，链接具体见第 4.2 节。

【参考文献】

1. uSID draft：https://tools.ietf.org/html/draft-filsfils-spring-net-pgm-extension-srv6-usid-00
2. SRH draft：https://tools.ietf.org/html/draft-ietf-6man-segment-routing-header-21

3. SRv6 draft：https://tools.ietf.org/html/draft-ietf-spring-srv6-network-programming-01
4. Linux SRv6 实战（第三篇）：多云环境下 Overlay(VPP) 和 Underlay 整合测试：https://www.sdnlab.com/23218.html
5. uSID:SRv6 新范式：https://www.sdnlab.com/23390.html
6. VPP 的相关资料：https://docs.fd.io/vpp/18.07/
7. VPP SRv6 相关资料/教程：https://wiki.fd.io/view/VPP/Segment_Routing_for_IPv6
8. etcd 资料/教程：https://etcd.io/
6. etcd 架构与实现解析：http://jolestar.com/etcd-architecture/