Linux SRv6 实战

（第二篇）

服务链功能详解

李嘉明 思科系统工程师

苏远超 思科首席工程师

钟　庆 思科系统工程师

摘要：本文基于 Linux SRv6 功能，结合 Vagrant，Snort 等工具，对 SRv6 服务链功能进行详细解析和验证。实现 SRv6 服务链上同时支持 SR-aware 服务和 Non SR-aware 服务。

本文所有代码见: https://github.com/ljm625/srv6_Sandbox

一、SRv6 服务链简介

如本文第一篇（https://www.sdnlab.com/22842.html）所述，SR 实现服务链本质上是基于自身的流量工程能力，事实上服务节点（或代理）本质上只是 Segment 列表中的一个 Segment，这不但适用于 SR-aware 服务，也适用于 Non SR-aware 服务。除非特别说明，SR 服务链解决方案同时适用于 SR MPLS 和 SRv6，本文只以 SRv6 为例进行说明。

在本文第一篇中，我们使用 Mininet 等工具，验证了 Linux SRv6 的 VPN、流量工程和服务链(SR-aware 服务)功能。在本篇中，我们将更进一步深入分析 SRv6 服务链功能，验证 SRv6 服务链对 Non SR-aware 服务的支持能力。

下面是 SRv6 服务链相关操作简介（更多 SRv6 操作介绍参见本文第一篇）：

End.AD4：该操作要求 SL 不为 0（不是最后一跳），是为了兼容 Non SR-aware 服务的服务链操作。核心思想是配置了此操作的节点作为 SR 代理（SR Proxy），即在把数据包转发给 Non SR-aware 服务处理之前，把外层的封装暂时去掉，再转发给服务进行处理，这样服务无需支持 SRv6 即可正常工作。End.AD4 操作要求内层

必须是一个 IPv4 数据包，此操作将更新外层报头的 SL，去掉外层 IPv6 报头后发送给 Non SR-aware 服务；在服务处理完成后把数据包发回时，SR Proxy 重新添加 IPv6 和 SRH 报头，继续转发。End.AD4 操作需要为每条服务链维护一个动态的缓存，以封装服务发回的数据包。具体如下图所示：

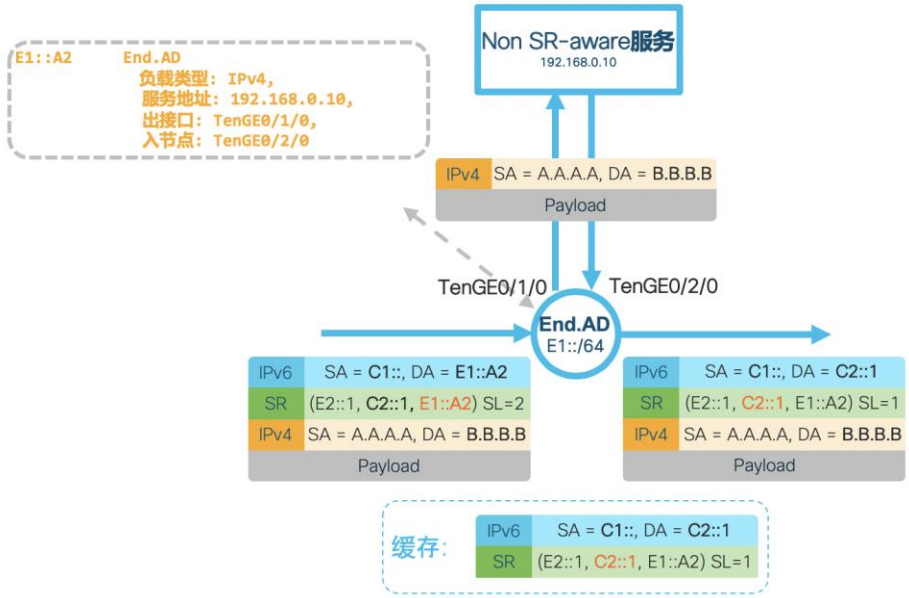

图 1：End.AD4 操作

End.AD6：该操作和 End.AD4 基本一致，唯一区别是其要求内层是一个 IPv6 数据包。

End.AM：该操作要求 SL 不为 0（不是最后一跳），也是为了兼容 Non SR-aware 服务的服务链操作。虽然同样是基于 SR Proxy 机制，但和 End.AD 操作不同，End.AM 操作会将 IPv6 目的地址更新为 SL=0 时的 Segment，即最终的 IPv6 目的地址，然后转发给服务；根据 RFC8200 规定，服务（中间节点）不会处理 SRH 而只是根据目的地址转发，这里假设服务可以处理跟在 SRH 之后的负载。服务把数据包发回后，End.AM 节点把 IPv6 目的地址更新为 SL 指定的 Segment，继续转发。End.AM 操作不需要为每条服务链维护缓存，其规则适用于所有经由 End.AM 节点的服务链。End.AM 这种实现机制只在 SRv6 上支持，SR MPLS 不支持类似的机制。

具体如下图所示：

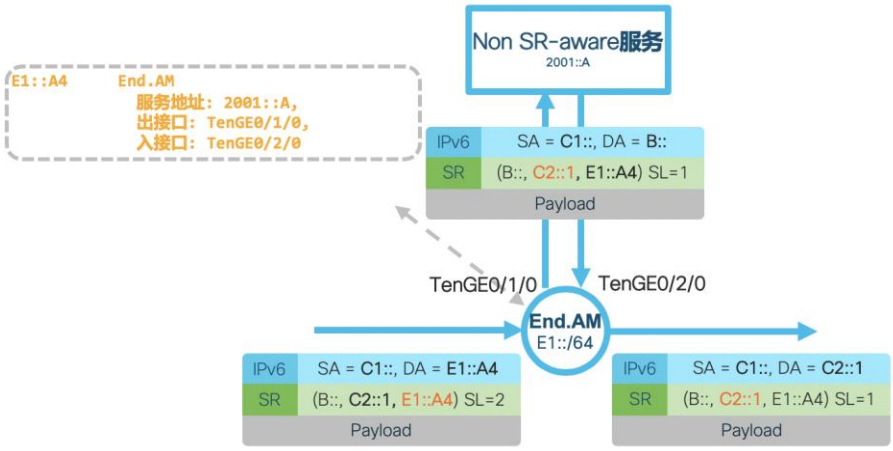

图 2：End.AM 操作

二、为什么使用 Linux SRv6 实现服务链？

在数据中心/云网络部署中，不少用户采用主机叠加网络（Host Overlay）的方式为租户提供服务。这种方式采用主机的虚拟交换机/FD.IO 进行 Host Overlay 部署，实现不同主机上虚拟机或者容器之间的二三层互通，也常用于实现服务链。

RFC 7665 提出了服务功能链（SFC）架构，RFC 8300 提出网络服务头（NSH）作为实现 SFC 架构的封装。NSH 携带了服务链路径和元信息（Metadata），这些信息可在不同的服务之间共享，利用 NSH 可实现动态的服务链配置，可在服务链形成后进行动态的路径和拓扑修改、插入新的服务，提供端到端可视化、OAM、性能管理能力。

但 NSH 目前实际应用很少，原因有三点：

1. NSH 目前面临的最大问题是：网络设备、VNF、主机操作系统对 NSH 的支持非常有限。网络设备只有个别路由器例如思科 ASR9000 支持，大部分硬件交换机不支持；实现 NSH 所设计的功能依赖于服务链中的 VNF 对路径和元信息的操作能力，但很多 VNF 都不支持

2. NSH 需要在每条服务链的所有服务设备上维持状态，这很大程度上限制了扩展性

3. NSH 携带了服务链路径信息，但是流量在不同 VNF 之间的引导还需要通过其他隧道技术机制来实现（例如 VxLAN、GRE 等），没有实现 Overlay 和 Underlay 的融合。这造成解决方案的差异性不强，OVS+Openflow 等替代方案可以实现类似的功能且较简单

如本文第一篇所述， Linux 内核从 4.10 版本（2017 年 2 月）就开始支持 SRv6，支持丰富的 SRv6 操作，通过开源的 Linux 内核模块 SREXT（https://github.com/netgroup/SRv6-net-prog/）支持更多的操作，其中包括对服务链功能的增强。通过把不同的 SRv6 Segment 组合起来，Linux SRv6 能够完美地

整合 Overlay、Underlay 和服务链，其性能通过应用 DPDK 或者 FD.IO/VPP 也可以得到极大的提升。

SRv6 可以与现有不支持 SRv6 的 IPv6 网络无缝互操作，基于 SRv6 的服务链也可以同时支持 SR-aware 和 Non SR-aware 的服务，因此基于 Linux SRv6 的服务链具有很高的实用性，基本上可以部署在任何支持 IPv6 的网络上。

三、准备工作

验证环境基于 Vagrant，可在 Windows/Linux/Mac 下进行，需要提前自行安装好 Virtualbox 和 Vagrant，这里不再赘述这两个软件的安装过程。容器内的软件如下

- Linux，内核版本高于 4.14
- Snort，开源 IDS 软件，我们会使用两个版本的 Snort：SR-aware 版本以及 Non SR-aware 版本

3.1 拓扑说明

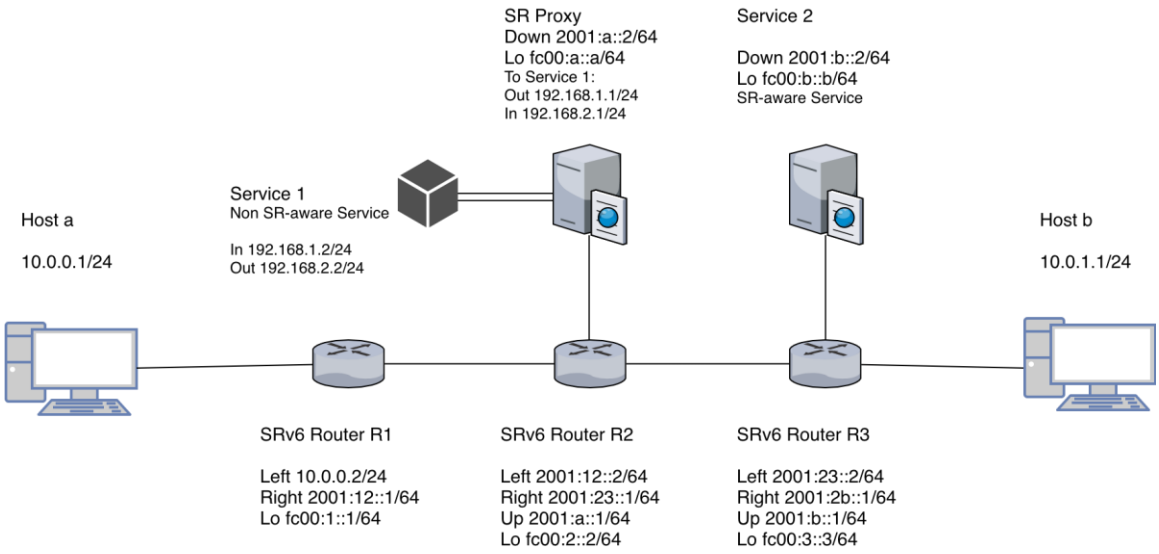

图 3 SRv6 服务链拓扑

拓扑如图 3 所示，有三台支持 SRv6 的路由器。其中 R2 连接配置了 End.AD4 操作的 SR Proxy，SR Proxy 下挂 Non SR-aware 版本的 Snort(Service1)；R3 连接了 SR-aware 版本的 Snort(Service2)。

主机 a 和主机 b 只通过 IPv4 连接到路由器 R1 以及 R3，默认情况下它们无法通讯。路由器之间只有 IPv6 地址和路由。

主机 a 发送给主机 b 的数据包，经由 R1、R2，然后发往 Service1；接着经由 R3，发往 Service2；最后发给主机 b。

3.2 部署脚本说明

在 Vagrant File 里面我们定义了拓扑结构，每个节点的镜像文件和配置信息，当我们执行“Vagrant up”命令的时候，会自动根据 Vagrant 的描述文件进行虚拟机的创建、启动及配置。

```
# Node R1 configuration
config.vm.define "r1" do |r1|
        r1.vm.box = "srouting/srv6-net-prog"
        r1.vm.box_version = "0.4.14"
        r1.vm.synced_folder(".", nil, :disabled => true, :id => "vagrant-root")
        r1.vm.network "private_network", ip: "10.0.0.2", virtualbox__intnet: "netv4a"
        r1.vm.network "private_network", ip: "2001:12::1",netmask: "64", virtualbox__intnet: "net12"

        r1.vm.provider "virtualbox" do |virtualbox|
                virtualbox.memory = "256"
                virtualbox.customize ['modifyvm', :id, '--nictrace2', 'on']
                virtualbox.customize ['modifyvm', :id, '--nictracefile2', 'trace1.pcap']
                virtualbox.customize ['modifyvm', :id, '--nictrace3', 'on']
                virtualbox.customize ['modifyvm', :id, '--nictracefile3', 'trace12.pcap']
                virtualbox.customize ['modifyvm', :id, '--cableconnected1', 'on']
                virtualbox.customize ['modifyvm', :id, '--cableconnected2', 'on']
                virtualbox.customize ['modifyvm', :id, '--cableconnected3', 'on']

        end
        r1.vm.provision "shell", path: "config/config_r1.sh"
end
```

图 4 Vagrant 配置示例

上图是 Vagrant 的配置文件中定义的一个节点。包括以下部分的内容：

1. 镜像信息

其中“vm.box”定义了我们使用的镜像，“vm.box_version”定义了所使用镜像的版本，在 vagrant 启动虚拟机的时候，如果本地找不到这个镜像，会去 vagrant 的镜像仓库下载，这一点和 docker 的镜像下载机制基本相同。

2. 虚拟网络

接着定义了该虚拟机的网络，这部分就是用来定义 Vagrant 的网络拓扑的相关配置，其中“virtualbox_intnet”定义了不同的私网，对于 IP 地址，这里可以设置 v4 或者 v6 地址，但如果是 IPv6 地址，建议手动设置子网掩码，会避免很多问题。

3. 内存、网卡状态、抓包等信息

“memory”定义虚拟机的内存大小，由于是测试用途，256MB 即可满足需求。

“nictrace”则是配置通过网卡实时抓包，并保存为 pcap 文件。我们设置了在所有的网卡上进行实时的抓包存储，这样方便测试和理解。

“cableconnected”定义了网卡的连接状态。读者可能有一个疑问是我们只定义了 2 个网络，为什么会有 3 张网卡？其实第一个网卡是系统默认定义的 NAT 网卡，用于连接互联网。此外，Virtualbox 具有每台虚拟机最多支持 4 张网卡的限制，因

此所定义的虚拟机，除了默认的 NAT 网络，如果有多于 3 个网络，那么多出的网卡将不会被正确添加。

4. Day0 配置

“vm.provision” 定义了当虚拟机启动之后运行的 Day0 自动配置脚本，在脚本里我们配置了默认路由，启用 SRv6 以及安装一些测试用的软件，相关的配置会在后面章节中说明。

四、安装教程

由于本次使用的是 Vagrant 安装方案，所以宿主机可以是 Windows/Mac/Linux，只需要按照官网的教程安装 git、Vagrant 以及 Virtualbox 即可。

在完成安装 git、Vagrant 以及 Virtualbox 之后，就可以启动测试环境了。

```
git clone https://github.com/ljm625/srv6_Sandbox
cd srv6_Sandbox
vagrant up
```

接着 Vagrant 会自动下载镜像，启动拓扑里面的主机，以及完成 Day0 的配置。需要注意的是单台虚拟机的内存为 256MB，因此电脑需要 256*6=1.5GB 的空闲内存空间。

这里用的镜像是定制过的 Debian image，已经包含了 4.14 的 Kernel 以及最新版的 iproute2，因此无需重新安装 iproute2 或者更新内核。

下面对每台设备的 Day0 启动脚本关键部分进行解析。

1. R1

对路由器 R1 的配置主要是把 IPv4 数据包封装入 SRv6 并转发；同时对回程数据包指定与 End.DX4 操作对应的 Segment，解封装后发给主机 a(10.0.0.1)，具体可以参考本文第一篇。

```
# 把去往 10.0.1.0/24 的数据包封装入 SRv6，设置 Segment 列表，并转发

ip route add 10.0.1.0/24 encap seg6 mode encap segs fc00:2::a,fc00:
3::a dev eth2

# 对 R1 收到的回程数据包执行 End.DX4 操作，把内层的 IPv4 数据包发给主机 a

ip -6 route add fc00:1::a/128 encap seg6local action End.DX4 nh4 10.
0.0.1 dev eth1
```

2. R2

对路由器 R2 的配置对指定与 End 操作对应的 Segment，此操作更新 SL 并更新数据包的 IPv6 目的地址。

```
# 中间节点，只需配置 End 操作

ip -6 route add fc00:2::a/128 encap seg6local action End dev eth1
```

3. R3

对路由器 R3 的配置和 R1 的配置基本相同，对回程数据包指定与 End.DX4 操作对应的 Segment，解封装并转发给主机 b(10.0.1.1)，实现主机 a 和主机 b 的通信；并对主机 b 的回程数据包封装，添加 SRH 报头。

```
# 对 R3 收到的回程数据包执行 End.DX4 操作，把内层的 IPv4 数据包发给主机 b

ip -6 route add fc00:3::a/128 encap seg6local action End.DX4 nh4 10.
0.1.1 dev eth1

# 把去往 10.0.0.0/24 的数据包封装入 SRv6，设置 Segment 列表，并转发

ip route add 10.0.0.0/24 encap seg6 mode encap segs fc00:1::a dev e
th2
```

4. SR Proxy

SR Proxy 的配置包含 3 部分：

- 使用 Linux Network Namespace（Linux 自带的网络虚拟化）机制，创建一个名字为”snort”的 VNF，相当于 SR Proxy 下挂了一个新的主机，突破了 Vagrant 单机只能有最多 4 张网卡的限制。这部分使用了一键脚本进行配置，VNF 的 2 张网卡 IP 分别为 192.168.1.2 和 192.168.2.2，VNF 会对传入的流量进行处理，然后从 192.168.2.2 发出。
- 在 SR Proxy 的 VNF 内安装并配置 Snort，实现 IDS 功能，即上图中的 Service1。
- 在 SR Proxy 下安装 SREXT 内核模块，这个内核模块扩展了 Linux 内核所支持的 SRv6 操作，使其能够支持 End.AD4、End.AD6 以及 End.AM 等操作，从而在服务链上支持 Non SR-aware 服务。

安装完毕之后配置 SR Proxy。

```
# 配置 SR Proxy 的 VNF，使用 Linux Namespace 进行隔离。

#这里使用了一键脚本
cd ~/
rm -rf srv6_Sandbox/
git clone https://github.com/ljm625/srv6_Sandbox
cd srv6_Sandbox/config/
sh deploy-vnf-v4.sh add snort veth0 veth1 192.168.1.1/24 192.168.2.
1/24 192.168.1.2/24 192.168.2.2/24
```

```
# 安装Snort，这里安装的是官方版本的Snort，不支持SRv6
cd ~/
wget https://snort.org/downloads/snort/daq-2.0.6.tar.gz
wget https://snort.org/downloads/snort/snort-2.9.12.tar.gz

tar xvzf daq-2.0.6.tar.gz
cd daq-2.0.6
./configure && make && sudo make install

cd ~/
tar xvzf snort-2.9.12.tar.gz
cd snort-2.9.12
./configure --enable-sourcefire --disable-open-serverid && make &&
sudo make install
sudo ldconfig


# 配置Snort（IDS）的默认监测规则
sudo mkdir -p /etc/snort/ /etc/snort/rules/ /var/log/snort

touch /etc/snort/snort.conf /etc/snort/rules/local.rule
echo 'var RULE_PATH rules' >> /etc/snort/snort.conf
echo 'include $RULE_PATH/local.rule' >> /etc/snort/snort.conf
echo 'alert icmp any any -> any any (msg:"ICMP detected"; sid:1000)
' >> /etc/snort/rules/local.rule


# 安装SREXT 模块，其扩展了Linux 所支持的SRv6 操作，例如End.AD4
cd ~/
git clone https://github.com/SRouting/SRv6-net-prog
cd SRv6-net-prog/srext/
make && make install && depmod -a && modprobe srext

# 配置SR Proxy 启用End.AD4 操作，服务的地址设置为192.168.1.2
srconf localsid add fc00:a::a1 end.ad4 ip 192.168.1.2 veth0 veth1
```

5. Service2

Service2 的配置相比 SR Proxy 要简单很多，因为上面运行的是支持 SRv6 的服务。

所以 Service2 只需要配置一个 End 操作即可，以更新 SL 和 IPv6 目的地址。

```
# 配置Service2 支持End 操作
ip -6 route add fc00:b::a1/128 encap seg6local action End dev eth1


# 配置SR-aware 版本Snort 的监测规则
sudo mkdir -p /etc/snort/ /etc/snort/rules/ /var/log/snort
```

```
touch /etc/snort/snort.conf /etc/snort/rules/local.rule
echo 'var RULE_PATH rules' >> /etc/snort/snort.conf
echo 'include $RULE_PATH/local.rule' >> /etc/snort/snort.conf
echo 'alert icmp any any -> any any (msg:"ICMP detected"; sid:1000)
' >> /etc/snort/rules/local.rule
```

五、验证 Linux SRv6 高级服务链功能

5.1 概述

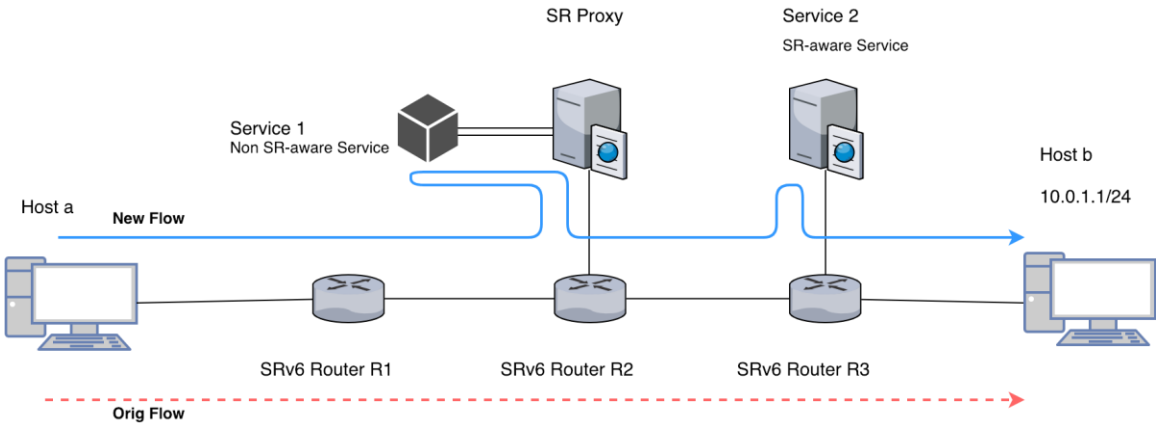

图 5 验证所用拓扑

本验证所用拓扑及验证场景如图 5 所示。在该拓扑中，默认情况 IPv4 主机 a 和主机 b 之间通过 SRv6 End.DX4 操作实现 VPN 互通，流量路径是 R1->R2->R3。通过修改 R1 的 SRv6 策略，我们使流量在去往主机 b 之前，先被引导至 SR Proxy 下 Non SR-aware 版本的 Snort(Service1)，再被引导至 Service2 的 SR-aware 版本的 Snort，最后到达主机 b 实现互通。

5.2 具体步骤

5.2.1 验证 SR Proxy 上的 Non SR-aware 服务(Service1)

在本次的验证中，我们已经默认实现了主机 a 与主机 b 之间的 IPv4 互访，这一步骤的细节参见本文第一篇。

当所有环境通过 vagrant up 启动之后，我们登陆到主机 a。

```
vagrant ssh hosta
ping 10.0.1.1
```

```
vagrant@srv6-net-prog:~$ ping 10.0.1.1
PING 10.0.1.1 (10.0.1.1) 56(84) bytes of data.
64 bytes from 10.0.1.1: icmp_seq=1 ttl=63 time=12.2 ms
64 bytes from 10.0.1.1: icmp_seq=2 ttl=63 time=2.85 ms
64 bytes from 10.0.1.1: icmp_seq=3 ttl=63 time=58.9 ms
64 bytes from 10.0.1.1: icmp_seq=4 ttl=63 time=2.78 ms
64 bytes from 10.0.1.1: icmp_seq=5 ttl=63 time=2.63 ms
64 bytes from 10.0.1.1: icmp_seq=6 ttl=63 time=2.55 ms
64 bytes from 10.0.1.1: icmp_seq=7 ttl=63 time=2.49 ms
64 bytes from 10.0.1.1: icmp_seq=8 ttl=63 time=2.70 ms
64 bytes from 10.0.1.1: icmp_seq=9 ttl=63 time=2.67 ms
64 bytes from 10.0.1.1: icmp_seq=10 ttl=63 time=2.58 ms
64 bytes from 10.0.1.1: icmp_seq=11 ttl=63 time=2.14 ms
64 bytes from 10.0.1.1: icmp_seq=12 ttl=63 time=2.47 ms
64 bytes from 10.0.1.1: icmp_seq=13 ttl=63 time=2.44 ms
```

图 6 主机 a 可以正常 ping 通主机 b

如图 6 所示，主机 a 与主机 b 可以正常互通。

在默认情况下，我们在 R1 配置的 SRv6 Policy 是经由 R1->R2->R3 进行转发，并没有经过 Service1/Service2。

我们可以通过 Wireshark 打开 R1 的 eth2 接口的抓包文件予以确认(默认为 trace12.pcap 文件)。

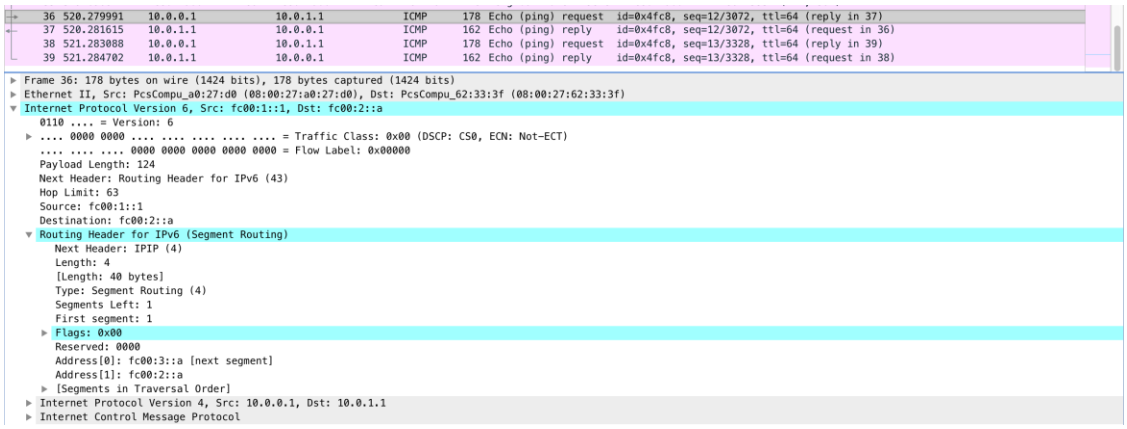

图 7 R1 上抓包结果-数据包被加上了 SRH

如图 7 所示，从主机 a(10.0.0.1)去往主机 b(10.0.1.1)的数据包，加上 SRH，Segment 列表指定了数据包经由 R2、R3 进行转发。

现在登陆到 R1 上，查看当前的 SRv6 Policy：

```
vagrant ssh r1
sudo -i
ip route list|grep seg6
```

```
root@srv6-net-prog:~# ip route list|grep seg6
10.0.1.0/24  encap seg6 mode encap segs 2 [ fc00:2::a fc00:3::a ] dev eth2 scope link
```

图 8 R1 上当前的 SRv6 Policy

然后，我们修改 SRv6 Policy，加入 2 个新的 Segment，引导流量至 Service1 和 Service2 上的 Snort 进行监测。主要注意的是，Service1 对应的 Segment 其实是 SR Proxy 配置的 End.AD4，而不是实际提供服务的 VNF 的地址（192.168.1.2）。

```
ip route del 10.0.1.0/24
ip route add 10.0.1.0/24 encap seg6 mode encap segs fc00:a::a1,fc00:
b::a1,fc00:3::a dev eth2
```

再次查看抓包文件 trace12.pcap，可以看到数据包的下一跳被重定向到了 Service1。

```
▶ Frame 362: 194 bytes on wire (1552 bits), 194 bytes captured (1552 bits)
▶ Ethernet II, Src: PcsCompu_08:cd:c8 (08:00:27:08:cd:c8), Dst: PcsCompu_63:99:21 (08:00:27:63:99:21)
▼ Internet Protocol Version 6, Src: fc00:1::1, Dst: fc00:a::a1
    0110 .... = Version: 6
  ▶ .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 140
    Next Header: Routing Header for IPv6 (43)
    Hop Limit: 63
    Source: fc00:1::1
    Destination: fc00:a::a1
  ▼ Routing Header for IPv6 (Segment Routing)
      Next Header: IPIP (4)
      Length: 6
      [Length: 56 bytes]
      Type: Segment Routing (4)
      Segments Left: 2
      First segment: 2
    ▶ Flags: 0x00
      Reserved: 0000
      Address[0]: fc00:3::a
      Address[1]: fc00:b::a1 [next segment]
      Address[2]: fc00:a::a1
    ▶ [Segments in Traversal Order]
  ▶ Internet Protocol Version 4, Src: 10.0.0.1, Dst: 10.0.1.1
  ▶ Internet Control Message Protocol
```

图 9 R1 上抓包结果–修改 SR Policy 后

接着，登陆到 SR Proxy 这台主机，名字叫“server1”。

```
vagrant ssh server1
sudo -i
srconf localsid show
```

```
root@srv6-net-prog:~# srconf localsid show
SRv6 - MY LOCALSID TABLE:
=========================================================
        SID     :       fc00:a::a1
        Behavior:       end.ad4
        Next hop:       192.168.1.2
        OIF     :       veth0
        IIF     :       veth1
        Good traffic:   [113 packets : 9492  bytes]
        Bad traffic:    [1 packets : 104  bytes]
---------------------------------------------------------
```

图 10 SR Proxy 上配置的 End.AD4 操作

可以看到 SR Proxy 配置了 End.AD4 操作。由于使用了 SREXT 模块，因此需要使用 srconf 命令。

如前所述，在 SR Proxy 上配置 End.AD4 操作的命令为（包含在 Day0 脚本中）：

```
srconf localsid add fc00:a::a1 end.ad4 ip 192.168.1.2 veth0 veth1
```

其中 fc00: a::a1 为 End.AD4 对应的 Segment，192.168.1.2 为 Non SR-aware 服务的地址。

在这里我们添加服务的方法是使用 Linux Network Namespace 虚拟出一个名字为"snort"的 VNF，此 VNF 包含有 2 个接口 veth0 和 veth1，接口地址设置见图 3。在 VNF 内安装 Non SR-aware 版本的 Snort。

首先通过 tcpdump，在 veth0 抓包

```
ip netns exec snort tcpdump -i veth0-snort
```

```
root@srv6-net-prog:/home/vagrant# ip netns exec snort tcpdump -i veth0-snort
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on veth0-snort, link-type EN10MB (Ethernet), capture size 262144 bytes
04:41:53.077068 IP 10.0.0.1 > 10.0.1.1: ICMP echo request, id 1277, seq 1, length 64
04:41:54.070538 IP 10.0.0.1 > 10.0.1.1: ICMP echo request, id 1277, seq 2, length 64
04:41:55.072315 IP 10.0.0.1 > 10.0.1.1: ICMP echo request, id 1277, seq 3, length 64
04:41:56.075362 IP 10.0.0.1 > 10.0.1.1: ICMP echo request, id 1277, seq 4, length 64
04:41:57.077458 IP 10.0.0.1 > 10.0.1.1: ICMP echo request, id 1277, seq 5, length 64
04:41:58.121165 ARP, Request who-has 192.168.1.2 tell 192.168.1.1, length 28
04:41:58.121208 ARP, Reply 192.168.1.2 is-at c2:9e:9e:e1:77:aa (oui Unknown), length 28
```

图 11 VNF 内 tcpdump 的抓包结果

如图 11 所示，在我们新建的 VNF 上收到的数据包已经没有了 IPv6 报头，而只有内层的 IPv4 报头。

接下来检查 Snort 上的规则。

```
ip netns exec snort bash
# 进入 Snort 这个 VNF 的 bash
cat /etc/snort/rules/local.rule
```

```
root@srv6-net-prog:/home/vagrant# cat /etc/snort/rules/local.rule
alert icmp any any -> any any (msg:"ICMP detected"; sid:1000)
```

图 12 VNF 内的 Snort 配置的规则

本验证所使用的 Snort 规则如图 12 所示，该规则针对所有 ICMP 包都会发出一个告警，并显示详细信息。

接下来运行 Snort，并查看 Snort 日志：

```
snort -c /etc/snort/snort.conf -A console
```

```
[ Port Based Pattern Matching Memory ]
pcap DAQ configured to passive.
Acquiring network traffic from "veth0-snort".
Reload thread starting...
Reload thread started, thread 0x7f7409f50700 (7029)
Decoding Ethernet

        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.12 GRE (Build 325)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2018 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.6.2
           Using PCRE version: 8.35 2014-04-04
           Using ZLIB version: 1.2.8

Commencing packet processing (pid=7022)
WARNING: No preprocessors configured for policy 0.
01/17-07:43:51.136740  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
WARNING: No preprocessors configured for policy 0.
01/17-07:43:52.135117  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
WARNING: No preprocessors configured for policy 0.
01/17-07:43:53.137399  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
WARNING: No preprocessors configured for policy 0.
01/17-07:43:54.139315  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
WARNING: No preprocessors configured for policy 0.
01/17-07:43:55.141075  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
WARNING: No preprocessors configured for policy 0.
01/17-07:43:56.143176  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
WARNING: No preprocessors configured for policy 0.
WARNING: No preprocessors configured for policy 0.
WARNING: No preprocessors configured for policy 0.
01/17-07:43:57.146189  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
WARNING: No preprocessors configured for policy 0.
01/17-07:43:58.147986  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
```

图 13 VNF 内的 Snort 监测日志

可以看到不支持 SRv6 的 Snort 能够正常地监测到 ICMP 包的源地址、目的地址以及协议，这是因为数据包原有的外层 IPv6 报头在执行 End.AD4 操作时已经被 SR Proxy 去除了，此时 Snort 收到的是内层的 IPv4 数据包。

我们再看一下 SR Proxy 处理之后继续转发的数据包，使用 wireshark 打开 traceserver1.pcap

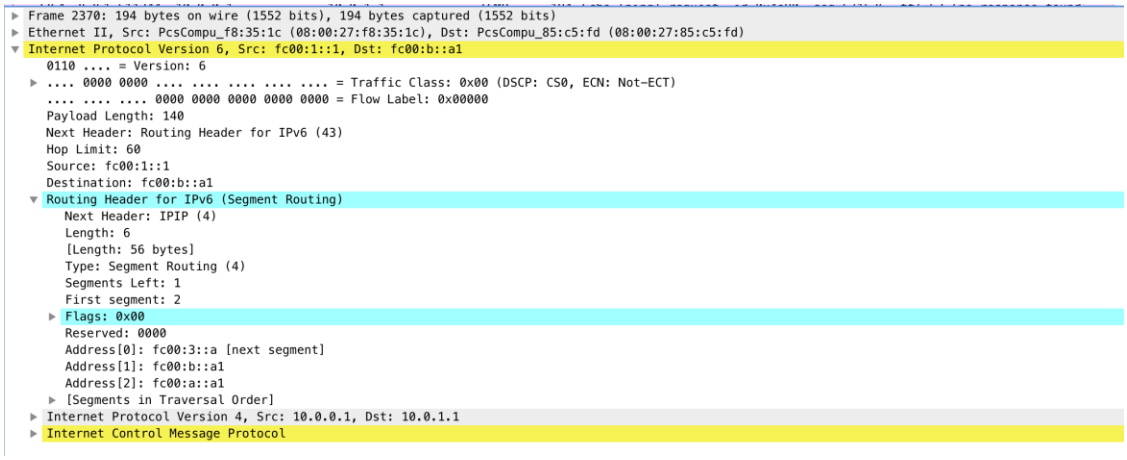

图 14 SR Proxy 上抓包结果–经过 SR Proxy 处理之后的数据包

如图 14 所示，可以看到经过 End.AD4 处理之后的数据包，在发回 SR Proxy 的时候，SR Proxy 会自动加回 IPv6 报头以及 SRH，其中 SL 和 IPv6 目的地址都已经更新。

下表总结了数据包经过 SR Proxy 前后的变化情况。

项目	进入 SR Proxy	进入 VNF	出 VNF	出 SR Proxy
IPv6 目的地址	fc00:a::a1	无	无	fc00:b::a1
SL	2	无	无	1
Segment 列表	[fc00:3::a,fc00:b::a1,fc00:a::a1]	无	无	[fc00:3::a,fc00:b::a1,fc00:a::a1]
IPv4 数据包	ICMP 10.0.0.1->10.0.1.1	ICMP 10.0.0.1->10.0.1.1	ICMP 10.0.0.1->10.0.1.1	ICMP 10.0.0.1->10.0.1.1

表 1 数据包经过 SR Proxy 前后的对比

5.2.2 验证 SR Proxy 上的 Non SR-aware 服务(iptables)

在这里我们继续做个测试，使用 Linux iptables 实现一个简单的防火墙功能。官方版本的 Linux iptables 目前是 Non SRv6-aware 的。

```
# 切换到Snort 的VNF
vagrant ssh server1
sudo -i
ip netns exec snort bash
# 配置iptables 规则
# 丢弃从10.0.0.0 去往10.0.1.0 的ICMP 数据包
iptables -I FORWARD --source 10.0.0.0/24 --destination 10.0.1.0/24
--protocol icmp -j DROP
# 查看iptables 状态
iptables -L -nvx
```

使用命令查看 iptables 的状态，如下图：

```
root@srv6-net-prog:~# iptables -L -nvx
Chain INPUT (policy ACCEPT 0 packets, 0 bytes)
    pkts      bytes target     prot opt in     out     source               destination

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
    pkts      bytes target     prot opt in     out     source               destination
       9      756 DROP       icmp --  *      *       10.0.0.0/24          10.0.1.0/24

Chain OUTPUT (policy ACCEPT 0 packets, 0 bytes)
    pkts      bytes target     prot opt in     out     source               destination
```

图 15 在 VNF 添加 iptables 规则后的统计信息

如图 15 所示，可以看到所添加的 iptables 规则成功匹配到了从 10.0.0.0 去往 10.0.1.0 的 ICMP 包，并实施了丢弃(DROP)操作。

```
64 bytes from 10.0.1.1: icmp_seq=723 ttl=63 time=3.22 ms
64 bytes from 10.0.1.1: icmp_seq=724 ttl=63 time=4.08 ms
64 bytes from 10.0.1.1: icmp_seq=725 ttl=63 time=4.00 ms
64 bytes from 10.0.1.1: icmp_seq=726 ttl=63 time=3.75 ms
^C
--- 10.0.1.1 ping statistics ---
735 packets transmitted, 726 received, 1% packet loss, time 735624ms
rtt min/avg/max/mdev = 2.406/4.019/55.895/2.214 ms
```

图 16 在 VNF 添加丢弃 ICMP 防火墙规则后，主机 a 无法 ping 通主机 b

回到主机 a，我们可以看到已经 ping 不通主机 b 了，再次证明了 SR 可以实现包含 Non SR-aware 服务的服务链。

现在删除这条防火墙规则，继续接下来的测试。

```
iptables -D FORWARD 1
```

5.2.3 验证 Server2 上的 SR-aware 服务(Service2)

数据包会被继续引导至 Service2，即 SR-aware 版本的 Snort。

登陆到 Service2 所在的主机，名字叫“server2”。

```
vagrant ssh server2
sudo -i
tcpdump -i eth1
```

```
root@srv6-net-prog:~# tcpdump -i eth1
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
10:22:56.371038 IP6 fc00:1::1 > fc00:b::a1: srcrt (len=6, type=4, segleft=1, last-entry=2, tag=0, [0]fc00:3::a, [1]fc00:b::a1, [2]fc00:a::a1) IP 10.0.0
.1 > 10.0.1.1: ICMP echo request, id 20173, seq 1, length 64
10:22:56.371063 IP6 fc00:1::1 > fc00:3::a: srcrt (len=6, type=4, segleft=0, last-entry=2, tag=0, [0]fc00:3::a, [1]fc00:b::a1, [2]fc00:a::a1) IP 10.0.0.
1 > 10.0.1.1: ICMP echo request, id 20173, seq 1, length 64
10:22:57.373281 IP6 fc00:1::1 > fc00:b::a1: srcrt (len=6, type=4, segleft=1, last-entry=2, tag=0, [0]fc00:3::a, [1]fc00:b::a1, [2]fc00:a::a1) IP 10.0.0
.1 > 10.0.1.1: ICMP echo request, id 20173, seq 2, length 64
10:22:57.373301 IP6 fc00:1::1 > fc00:3::a: srcrt (len=6, type=4, segleft=0, last-entry=2, tag=0, [0]fc00:3::a, [1]fc00:b::a1, [2]fc00:a::a1) IP 10.0.0.
1 > 10.0.1.1: ICMP echo request, id 20173, seq 2, length 64
10:22:58.375778 IP6 fc00:1::1 > fc00:b::a1: srcrt (len=6, type=4, segleft=1, last-entry=2, tag=0, [0]fc00:3::a, [1]fc00:b::a1, [2]fc00:a::a1) IP 10.0.0
.1 > 10.0.1.1: ICMP echo request, id 20173, seq 3, length 64
10:22:58.375796 IP6 fc00:1::1 > fc00:3::a: srcrt (len=6, type=4, segleft=0, last-entry=2, tag=0, [0]fc00:3::a, [1]fc00:b::a1, [2]fc00:a::a1) IP 10.0.0.
1 > 10.0.1.1: ICMP echo request, id 20173, seq 3, length 64
10:22:59.377982 IP6 fc00:1::1 > fc00:b::a1: srcrt (len=6, type=4, segleft=1, last-entry=2, tag=0, [0]fc00:3::a, [1]fc00:b::a1, [2]fc00:a::a1) IP 10.0.0
.1 > 10.0.1.1: ICMP echo request, id 20173, seq 4, length 64
```

图 17 在 Server2 主机上的 tcpdump 抓包结果

如图 17 所示，在 Server2 上抓到的数据包是带有 IPv6 和 SRH 报头的。若使用官方版本的 Snort 将无法监测到内层的 IPv4 数据包内容。

因此我们需要运行修改过的 SR-aware 版本的 Snort。

```
snort -c /etc/snort/snort.conf -A console -i eth1
```

```
[ Port Based Pattern Matching Memory ]
pcap DAQ configured to passive.
Acquiring network traffic from "eth1".
Reload thread starting...
Reload thread started, thread 0x7fa3d30c3700 (20116)
Decoding Ethernet

        --== Initialization Complete ==--

   ,,_     -*> Snort! <*-
  o"  )~   Version 2.9.11.1 GRE (Build 268)
   ''''    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
           Copyright (C) 2014-2017 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using libpcap version 1.6.2
           Using PCRE version: 8.35 2014-04-04
           Using ZLIB version: 1.2.8

Commencing packet processing (pid=20111)
WARNING: No preprocessors configured for policy 0.
01/17-10:34:24.309746  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
WARNING: No preprocessors configured for policy 0.
01/17-10:34:24.309764  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
WARNING: No preprocessors configured for policy 0.
01/17-10:34:25.308877  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
WARNING: No preprocessors configured for policy 0.
01/17-10:34:25.309006  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
WARNING: No preprocessors configured for policy 0.
01/17-10:34:26.311325  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
WARNING: No preprocessors configured for policy 0.
01/17-10:34:26.311367  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
WARNING: No preprocessors configured for policy 0.
01/17-10:34:27.312688  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
WARNING: No preprocessors configured for policy 0.
01/17-10:34:27.312705  [**] [1:1000:0] ICMP detected [**] [Priority: 0] {ICMP} 10.0.0.1 -> 10.0.1.1
```

图 18 在 Server2 上运行 SR-aware 版本的 Snort 的监测结果

支持 SRv6 的 Snort 会跳过 SRH 报头直接读取内层的 IPv4 数据包内容，从而实现服务链，这里不需要配置 End.AD 这类操作。

5.2.4 验证 Server2 上的 Non SR-aware 服务(iptables)

新建一条与 5.2.2 小节中一样的 iptables 规则进行对比。

```
vagrant ssh server2
sudo -i
# 配置 iptables 规则
# 丢弃从 10.0.0.0 去往 10.0.1.0 的 ICMP 数据包
iptables -I FORWARD --source 10.0.0.0/24 --destination 10.0.1.0/24
--protocol icmp -j DROP
iptables -I INPUT --source 10.0.0.0/24 --destination 10.0.1.0/24 --
protocol icmp -j DROP
# 查看 iptables 状态
iptables -L -nvx
```

```
root@srv6-net-prog:~# iptables -L -nvx
Chain INPUT (policy ACCEPT 46 packets, 2236 bytes)
    pkts      bytes target     prot opt in     out     source               destination
       0        0 DROP       icmp --  *      *       10.0.0.0/24          10.0.1.0/24

Chain FORWARD (policy ACCEPT 0 packets, 0 bytes)
    pkts      bytes target     prot opt in     out     source               destination
       0        0 DROP       icmp --  *      *       10.0.0.0/24          10.0.1.0/24

Chain OUTPUT (policy ACCEPT 35 packets, 2764 bytes)
    pkts      bytes target     prot opt in     out     source               destination
```

图 19 在 Server2 添加 iptables 规则后的统计信息

如图 19 所示，可以看到 INPUT 和 OUTPUT 都统计到了数据包，但因为数据包最外层具有 IPv6 报头，因此 iptables 规则没能匹配成功，ping 流量也不会被丢弃。

六、总结与展望

在本文第一篇的基础上，本篇对 Linux SRv6 服务链进行了详细的解析和验证，基于开源的 Linux 内核模块 SREXT，实现了服务链上同时支持 SR-aware 服务和 Non SR-aware 服务。

从结果来看，Linux 内核模块 SREXT 已经能比较好地支持服务链的操作。SR 实现服务链本质上是基于自身的流量工程能力，服务节点（或代理）本质上只是 Segment 列表中的一个 Segment，因此我们可以采用 SR 流量工程的 OAM 手段来管理服务链，这极大地降低了服务链本身的管理开销。

在本文第三篇中，我们将会介绍用 FD.IO/VPP 来实现 SRv6，以提供高性能的 Linux SRv6 解决方案，同时还会把 FD.IO/VPP 与物理网络设备通过 SRv6 结合起

来，从而构建出虚拟/物理一体、Overlay/Underlay 融合的高性能 SRv6 网络—我们坚定地认为这是未来网络发展的方向！

【参考文献】

1. SRH draft: https://tools.ietf.org/html/draft-ietf-6man-segment-routing-header-15
2. SRv6 draft：https://tools.ietf.org/html/draft-filsfils-spring-srv6-network-programming-06
3. SR Service Programming draft: draft-xuclad-spring-sr-service-programming-01
4. Segment Routing 的相关资料：https://segment-routing.net
5. SRv6 Linux 的相关资料/教程：https://segment-routing.org
6. SREXT 的相关资料/教程：https://github.com/netgroup/SRv6-net-prog
7. SR-SFC-DEMO 的相关资料：https://github.com/SRouting/SR-sfc-demo