uSID：SRv6 新范式

苏远超 思科首席工程师

蔡德忠 阿里巴巴基础设施首席架构师

蒋治春 腾讯资深架构师

摘要：本文介绍最新的 SRv6 创新 uSID（Micro Segment）。uSID 兼容既有的 SRv6 框架，将极大地改变 SRv6 的设计、实现和部署方式，成为 SRv6 的新范式。

一、SRv6 101

Segment Routing（以下简称 SR）指由思科院士 Clarence Filsfils 发明，并主要由 IETF SPRING（Source Packet Routing In Networking）工作组进行标准化的新一代网络传送技术。SR 基于源路由并且只在网络边缘维持状态，这使得 SR 非常适合于超大规模 SDN 部署，现已成为支持 5G、物联网、多云、微服务发展的标准网络传送技术。

Clarence 在发明 SR 的第一天，就为 SR 数据平面设计了两种实现方式（详见 RFC 8402）：一种是 SR-MPLS，其重用了 MPLS 数据平面，可以在现有 IP/MPLS 网络上增量部署；另一种是 SRv6，使用 IPv6 数据平面，基于 IPv6 路由扩展头进行扩展（SRv6 报头格式详见 IETF 草案 draft-ietf-6man-segment-routing-header-21），可以在现有 IPv6 网络上增量部署。

如果说 SR-MPLS 可以简单地认为是“下一代 MPLS”的话，那么 SRv6 则是代表了全新的思考、设计、运营网络的方式——网络即计算机（详见 IETF 草案 draft-ietf-spring-srv6-network-programming-01）。下图是笔者经常使用的一张类比示意图：

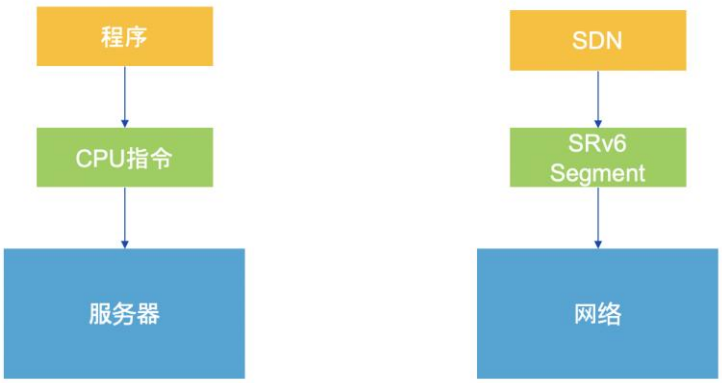

图 1 网络即计算机-X86 架构 vs SRv6 架构

- 图中左侧是大家熟悉的 X86 体系，程序最终是通过 X86 的 CPU 指令来操控服务器；右侧是 SRv6 体系，SDN 通过 SRv6 Segment 来操控网络。
- 与 X86 CPU 指令是固定的不同，SRv6 Segment 指令是可以任意扩充的（我们建议把新的 Segment 指令提交 IETF 进行标准化，但私有的 Segment 指令也是允许的），这赋予了 SRv6 极高的灵活性和极强的可扩展性。
- 目前多个 IETF 草案定义了多种 SRv6 Segment 指令，包括 Underlay 的 Segment(转发、TE)，也包括 Overlay 的 Segment(L2/L3 VPN)，还包括服务编程的 Segment(服务链)以及用于 5G 移动核心网用户面的 Segment 等。可以看出，SRv6 早已超出了 Underlay 的范畴，朝着全功能、网络级指令集演进。

还需要强调的是，SRv6 实现了网络极简：控制平面是支持 IPv6 的 IGP/BGP，转发平面则是纯 IPv6。“简单即力量”，SRv6 无疑在降低 OPEX/CAPEX 方面有着非常好的前景。

SRv6 极简和可编程两大特性得到了业界的广泛认可。事实上，SRv6 的生态系统发展的很快，如下图所示:

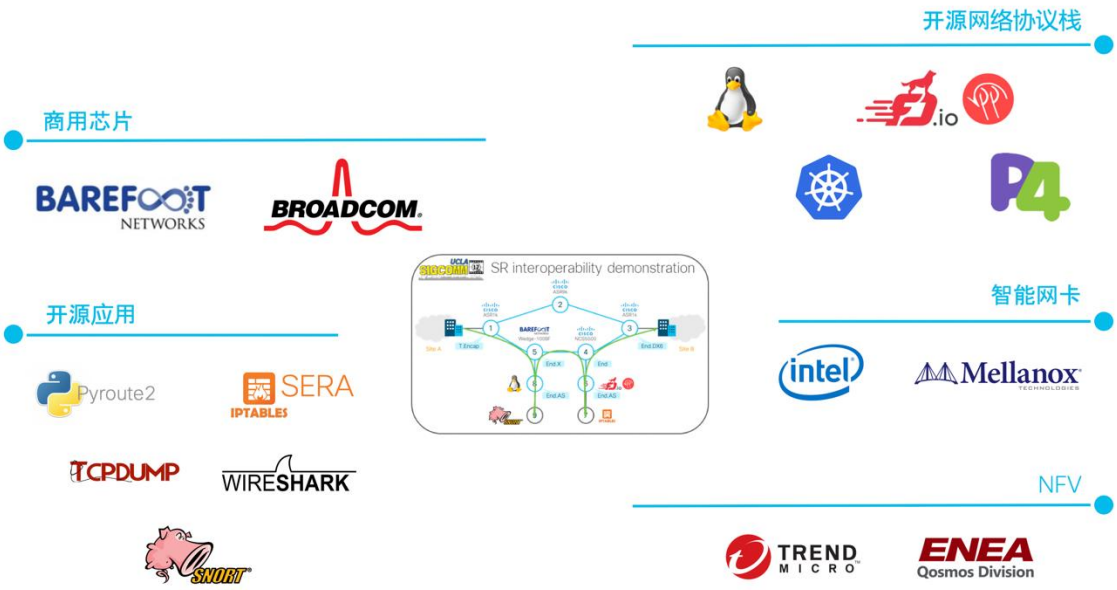

图 2 SRv6 生态系统

但 SRv6 在实际网络中的部署却很少，部署的业务也只是尽力而为的 L3VPN over SRv6，没有 SRv6 流量工程，更没有服务链这类高级功能。与之相对的是，随着 Linux/VPP/智能网卡对 SRv6 的支持，SRv6 在主机侧的应用和创新则是蓬勃发展（详见本文作者发表的 Linux SRv6 系列文章）。

SRv6 在网络侧和主机侧呈现完全不同的情况，原因是什么？如何加速网络侧的 SRv6 部署？网络侧 SRv6 与主机侧 SRv6 如何整合/联动从而实现“网络即计算机”的愿景？

二、SRv6 的阿喀琉斯之踵

前面谈到了 SRv6 在网络侧和主机侧呈现出完全不同的发展速度，根本原因是两者在查找和转发机制上存在根本的差异。

- 网络侧：ASIC/NPU 收到数据包后，把数据包存在外置的内存中。ASIC/NPU 读取固定长度的报头内容（一般是 96~128 字节），然后查找芯片本地/外部内存中的转发表，进行转发。如果报文头太长，无法在一个处理周期完成读取，则需要使用两个处理周期进行读取（Recycle），这将导致吞吐量下降一半。
- 主机侧：CPU 读取完整的（一组）数据包，查找路由表/缓存，进行转发。因此报文头的长度对主机的吞吐不会有太大的影响，当然代价就是吞吐量远低于 ASIC/NPU。

在 SR-MPLS 下，协议引入的开销较小，因此现有的大多数网络设备硬件均可以在一个处理周期内读取完 SR 报头信息，完成转发，意味着现有的硬件无须替换，只需升级软件即可支持 SR-MPLS。这是 SR-MPLS 能迅速得到大量部署的技术基础之一。

但 SRv6 引入的协议开销远大于 SR-MPLS[1]，Segment 所对应的操作也比 SR-MPLS 复杂，因此 SRv6 对网络设备提出了非常高的要求。如果按照目前的 SRv6 协议实现，要么需要替换掉绝大多数的网络设备，要么网络吞吐降低一半（Recycle），这对于很多用户而言是难以接受的。毕竟 SRv6 虽好，但如果其前期门槛是如此高的话，网络业界都会踌躇不前；缺乏网络侧 SRv6(Underlay)的支持，主机侧（APP/Overlay）的 SRv6 创新也难以大规模部署，因为无法确保端到端的用户体验——这成为了 SRv6 的阿喀琉斯之踵。

下面我们通过一个例子来具体分析 SR-MPLS 和 SRv6 在协议开销、承载效率、MTU 和对硬件芯片要求等方面的异同。假设净荷是 IPv4 报文，净荷长度是 IMIX 440B，需要经过具有 5 个 Segment 的路径转发数据包。

项目	SR-MPLS	SRv6
协议开销	4*5=20B	16*4[2] (Segment 列表)+8B(SRH)+40B(IPv6 报头)=112B

[1] 在整个传送路径上直到倒数第二跳，SRv6 数据包中的 SRH 报头始终存在且长度不会缩短（不像 SR-MPLS 每经过一个航路点会弹出相应的 Segment 标签），这就使得承载效率的问题更加严重。

[2] 总共需要 5 个 Segment，这里假设采用了 Reduce 操作，因此 Segment 列表里只包含了后 4 个 Segment，第 1 个 Segment 已经被拷贝至 IPv6 报头的目的地址字段。

项目	SR-MPLS	SRv6
承载效率	440/(20+440)=95.7%	440/(112+440)=79.7%
MTU	1520B	1612B
对硬件 NPU/ASIC 要求	最多只需读出前 20B(标签栈)+20B（IPv4 报头，用于负载均衡）报头信息即可转发 执行的操作是 SR-MPLS 定义的标准“压入”、“交换”和“弹出”标签操作，主要是 Underlay 相关	需要读出前 112B 报头信息才能转发 执行的操作类型接近 20 种，包括 Underlay、Overlay 和服务编程等，并且还在不断扩展之中
转发过程是否弹出 Segment	弹出。栈顶标签即为活动 Segment	不弹出。通过 Segment Left 指针标识出活动 Segment
负载均衡实现	一般根据净荷的 IP 5 元组进行哈希；如果 Segment 数量多，需要使用 Entropy 标签	根据外层 IPv6 报头的 Flow Label 字段进行哈希
对网络中间航路点（非头端）的要求	低。由于 SR-MPLS 的开销小，而且会逐步弹出，因此中间节点查找的难度不大。主要的挑战在于在标签栈数量多时，如何实现有效的负载均衡	高。由于 SRv6 不弹出 Segment，因此即使是中间航路点，也需要读取整个 SRv6 报头。

表 1 SR-MPLS vs SRv6

从上表可以看出，SRv6 在网络可编程性和负载均衡方面有着巨大的优势，但要发挥其优势，需要迫切解决 SRv6 在协议开销、承载效率、MTU 和对硬件要求方面的问题。这几个问题，其实本质上都是同一个问题：如何提高 SRv6 Segment 效率？

三、uSID 原理

思科（Clarence）联合业界众多领先的运营商、OTT、设备厂商和芯片厂商，在 2019 年 7 月 8 日提交了 IETF 草案 draft-filsfils-spring-srv6-net-pgm-extension-usid-00。这个草案对现有 SRv6 框架做了扩展，定义了新的 Segment 类型 uSID（Micro Segment）。

uSID 可以彻底解决上述的协议开销、承载效率、MTU 和对硬件要求高方面的问题。uSID 将极大加速 SRv6 在网络侧的部署，并成为 SRv6 新范式。一个典型的 uSID 如下图所示。可见一个 128bit 的 IPv6 地址被分为 8 份，第 1 份（16bit）用于表示 uSID 块信息，另外 7 份每份用于表示一个 Segment 信息（uSID），这意味着 SRv6 Segment 效率提高了 7 倍!

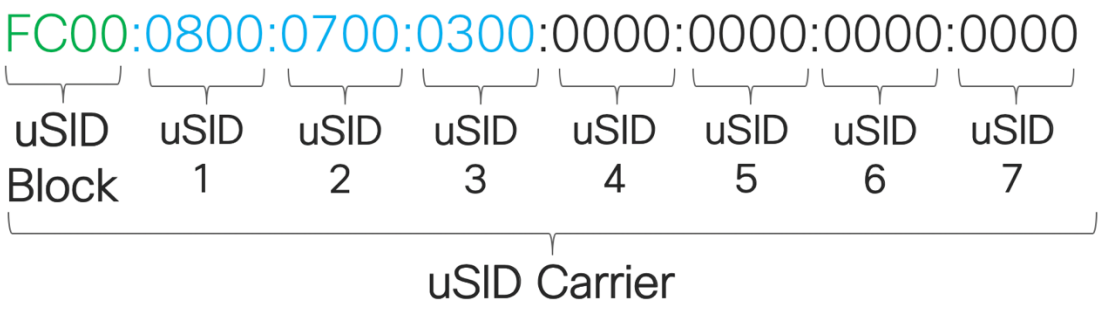

图 3 uSID 示例

3.1 uSID 概念

uSID 相关概念如下:

- uSID 承载器(uSID Carrier)：128bit 的 SRv6 Segment，其格式为<uSID 块><活动 uSID><下一个 uSID>...<最后一个 uSID><承载器结束标志><承载器结束标志>
- uSID：16bit 的 Segment ID。也可采用其他长度。
- uSID 块(uSID Block)：uSID 地址块
- 活动 uSID(Active uSID)：在 uSID 地址块后的第一个 uSID。
- 下一个 uSID(Next uSID)：在活动 uSID 之后的 uSID。
- 最后一个 uSID(Last uSID)：在第一个承载器结束标志前的 uSID。
- 承载器结束标志(End-of-Carrier)：16 进制值“0000”作为承载器结束标志。承载器内所有空闲的位置都需要用承载器结束标志来填充，因此承载器结束标志可以在一个承载器内出现多次。

uSID 的设计在一定程度上借鉴了 MPLS 封装，但保持了对 IP 路由最长匹配机制的支持。如果将 uSID 承载器中每个 16bits 的 uSID 类比于一个 32 bits 的 MPLS 标签，则可以将整个 uSID 承载器中除了 uSID 块（前 16 个 bits）以外的部分看做最多 7 个 MPLS 标签。uSID 相对于 MPLS 标签的一个优化是每个

uSID 只包含 Segment 信息（即 MPLS 标签中的 20bits 的部分），而去除了 3 bits 的 TC(Traffic Class，之前叫 EXP)、1bit 的栈底标志和 8 bits 的 TTL——TC 和 TTL 信息在 IPv6 报头字段体现，栈底标志由承载器结束标志表示。

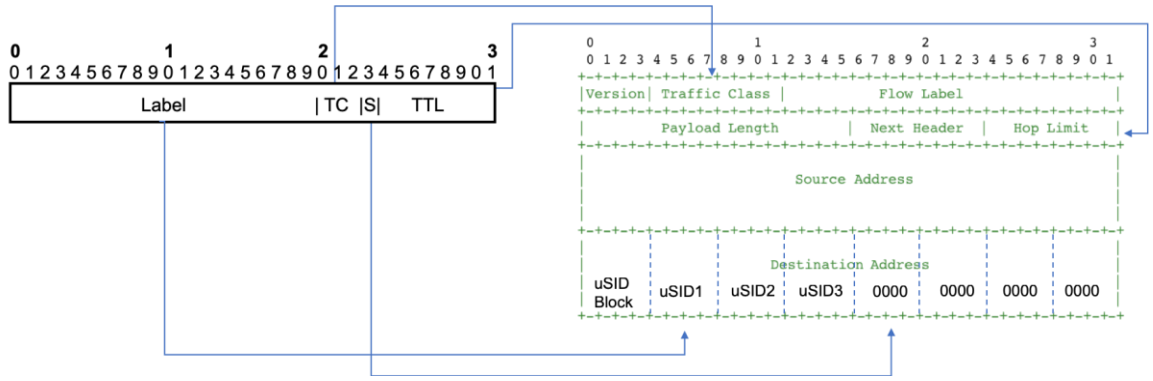

图 4 uSID 与 MPLS 封装的对比

3.2 uSID 相关操作

目前草案里只定义了一个操作 uN，功能类似于 SRv6 标准的 End 操作。

假设 uSID 块=FC00::/16(ULA 地址，详见 RFC 4193),节点 N 对应的 uSID=0N00, 则在节点 N 上 uN 操作会与以下两条 FIB 条目相关联:

- FC00:0N00/32 绑定至“Shift & Forward（位移 & 转发）”伪代码
- FC00:0N00:0000/48 绑定至 End 操作（弹出 uSID 承载器，处理下一 SRv6 Segment)

其中“Shift & Forward”伪代码如下:

- 把 uSID 承载器中第 32 至 127 位的内容拷贝至第 16-111 位（即二进制左移 16 位，注意从 0 开始编号）。
- 把第 112 至 127 位的内容置为 16 进制的“0000”（承载器结束标志)
- 在 FIB 中查找更新后的目的地址（即 uSID 块和活动 uSID 组合起来的前缀)
- 按照匹配的条目转发数据包

uN 操作中设备转发表项设计的巧妙构思，可以将 IP 路由最长匹配的优势发挥得淋漓尽致。

- 设备利用一长一短两条 FIB 条目进行最长匹配查找，若匹配到长条目（uSID 块+活动 uSID+承载器结束标志），说明被处理的 uSID 是 uSID 承载器中的最后一个 Segment，从而执行 uSID 承载器的弹出；若匹配到短条目（uSID 块+活动 uSID，下一个 uSID 不是承载器结束标志），说明被处理的 uSID 不是整个 uSID 承载器的最后一个 Segment，则执行“Shift & Forward”操作。uSID 对栈底标志的处理，更接近于 MPLS，而不是常规的 SRv6 使用的 Segment Left 指针。
- uSID 在转发时则完全继承了 SRv6 的特点。不同 uSID 之间完全是标准的 IPv6 路由，遵循 IP 路由最长匹配原则。因此，用户完全可以只通告 uSID 块的汇总路由(甚至只通告默认路由)给末端接入网络，而无须像 MPLS 一样一定要建立端到端 LSP 才能转发。这无疑将极大简化了全网的路由设计和减少 了 FIB 条目。需要注意的是，uSID 操作形成的 Segment 是固定可预知的(uSID 块+活动 uSID)，无须建立和维护复杂的映射表，这使得硬件设备实现起来非常简单。

由此可见，uSID 操作是非常简单的，现有的硬件芯片完全可以完成；并且由于 uSID 将 SRv6 Segment 效率提高了 7 倍，因此硬件芯片只需要读取有限的包长就可以完成转发（详见第 3.4.2 节）。

3.3 uSID 转发流程示例

以下基于 IETF 草案 draft-filsfils-spring-srv6-net-pgm-extension-usid-00 中的示例，介绍 uSID 转发流程。

本示例将介绍如何基于 uSID 实现低延迟 IPv4 L3VPN 业务，示例拓扑如下图所示:

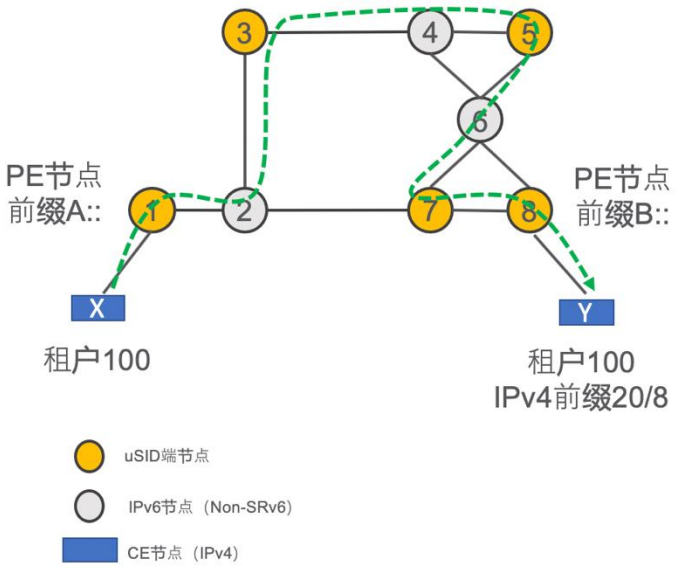

图 5 uSID 转发示例-低延迟 IPv4 L3VPN over SRv6 业务

假设 uSID 块=FC00::/16,节点 N 对应的 uSID=0N00。

图中节点 1-节点 8 位于同一 IGP 域中，节点 1 和节点 8 是 SRv6 PE 节点。此 IGP 域中所有链路的 IGP 度量相等，链路 3->4->5->6->7 具有更低的延迟。节点 1/节点 3/节点 5/节点 6/节点 7/节点 8 均支持 uSID，且都在 IGP 中通告 FC00:0N00::/32 路由。

图中节点 X 和节点 Y 是 IPv4 CE 节点，不属于 SRv6 域。

现在要基于 uSID 为 IPv4 数据包(X,Y)[3]提供低延迟的 L3VPN 业务。转发流程如下:

- PE 节点 1 把 IPv4 数据包(X,Y)封装入 IPv6 报头内，并转发给节点 3。

  此 IPv6 报头的目的地址=FC00:0300:0500:0700::，SRH= (B:8:D0::; SL=1; NH=4)。其中 FC00:0300:0500:0700::是承载器，用于编码 3->4->5->6->7 的低延迟路径（注意不需要把路径中的每一跳进行编码，而只需要对航路点进行编码，以充分发挥 IP 路由 ECMP 能力）；B:8:D0::是表示 End.DT4 操作（Per-VRF IPv4 VPN）的常规 SRv6 Segment。

  此时完整的数据包格式是：(A1::, FC00:0300:0500:0700::)(B:8:D0::; SL=1; NH=4)(X, Y)。

[3] （X,Y）表示源地址=X，目的地址=Y

由于节点 3 通告了 FC00:0300::/32 路由，因此数据包将根据 IGP 路由转发给节点 3。

- 节点 2 在从节点 1 去往节点 3 的 IGP 最短路径上，因此节点 2 会收到 PE 节点 1 发出的上述数据包。由于目的地址 FC00:0300:0500:0700::不是节点 2 上的地址，因此根据 RFC 8200 的规定，节点 2 不处理扩展头，而只是简单地查自身路由表把数据包发给节点 3。

- 节点 3 收到数据包(A1::, FC00:0300:0500:0700::)(B:8:D0::; SL=1; NH=4)(X, Y)，FC00:0300::/32 在本地 SID 表中的 uN 操作匹配，节点 3 执行“Shift & Forward”操作：把目的地址更新为 FC00:0500:0700::；查找路由表，匹配到最长条目 FC00:0500::/32，于是把数据包转发给节点 5。

  此时完整的数据包格式是：(A1::, FC00:0500:0700::)(B:8:D0::; SL=1; NH=4)(X, Y)

- 节点 4 在从节点 3 去往节点 5 的 IGP 最短路径上。节点 4 的处理与节点 2 类似，只是简单地按照 IGP 路由把数据包转发至节点 5。

- 节点 5 收到数据包(A1::, FC00:0500:0700::)(B:8:D0::; SL=1; NH=4)(X, Y)，FC00:0500::/32 在本地 SID 表中的 uN 操作匹配，节点 5 执行“Shift & Forward”操作：把目的地址更新为 FC00:0700::；查找路由表，匹配到最长条目 FC00:0700::/32，于是把数据包转发给节点 7。

  此时完整的数据包格式是：(A1::, FC00:0700::)(B:8:D0::; SL=1; NH=4)(X, Y)

- 节点 6 在从节点 5 去往节点 7 的 IGP 最短路径上。节点 6 的处理与节点 2 类似，只是简单地按照 IGP 路由把数据包转发至节点 7。

- 节点 7 收到数据包(A1::, FC00:0700::)(B:8:D0::; SL=1; NH=4)(X, Y)，FC00:0700:0000::/48 与本地 SID 表中的 uN 操作匹配（注意由于 IP 路由最长匹配的特性，此时不是匹配 FC00:0700::/32 这个条目），因此节点 7 执行支持 PSP[4]和 USD[5]的 End 操作：将 SL 递减 1，SL=0，把 IPv6 报头的目的地址更新为 Segment 列表[0]，即 B:8:D0::；由于此时 SL=0，节点 7

[4] PSP=Penultimate Segment Pop，倒数第二个 Segment 弹出

[5] USD=Ultimate Segment Decapsulation，最终 Segment 解封装

执行 PSP 操作，把 SRH 弹出；根据新的目的地址 B:8:D0::查找路由表，把数据包沿最短路径转发至节点 8。

此时完整的数据包格式是：(A1::, B:8:D0::)(X, Y)。

- 节点 8 收到数据包(A1::, B:8:D0::)(X, Y)，执行 End.DT4 操作：去掉外层的 IPv6 报头，在相应的 VRF 表中查找 VPN 前缀路由并转发。

3.4 uSID 优势

3.4.1 互操作性

uSID 完全遵循 srv6-network-programming 框架，并非是重新发明一套体系。事实上，uSID 只是一系列符合 srv6-network-programming 框架的新指令。

正是由于 uSID 本质上只是一个新的 SRv6 Segment，因此 uSID 可以与常规的 SRv6 Segment 互操作，也可以与纯 IPv6 节点(Non-SR)互操作。例如在第 3.3 节中，PE 节点 1 生成数据包(A1::, FC00:0300:0500:0700::)(B:8:D0::; SL=1; NH=4)(X, Y)，此数据包使用 uSID 编码路径，沿途经过纯 IPv6 节点（节点 2/节点 4/节点 6），最后在节点 8 基于常规 SRv6 Segment 完成标准的 End.DT4 操作。

uSID 与常规 SRv6 Segment、纯 IPv6 转发的互操作能力使用户可以在现网增量部署 uSID，无需对现有的硬件做全面的替换；利用 IPv6 提供的可达性，轻松实现 uSID 的跨域部署。典型的互操作场景如下图所示:

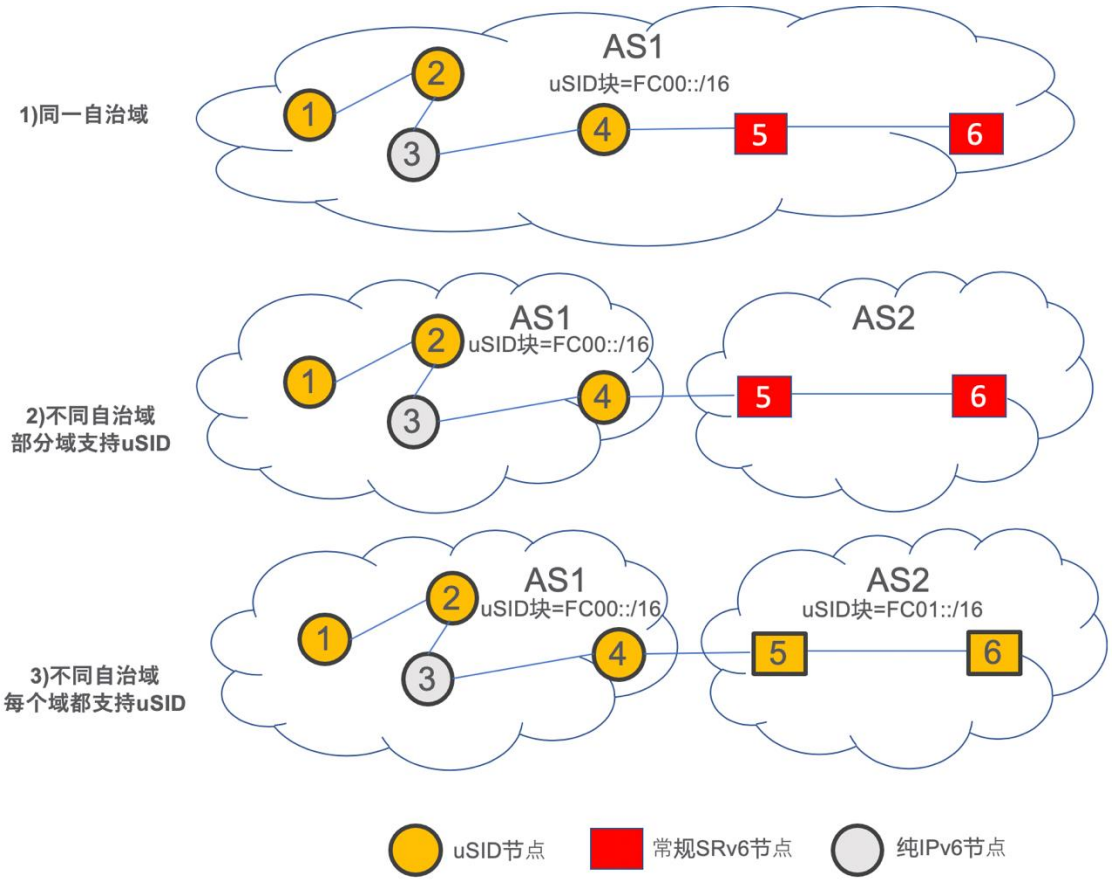

图 6 uSID 互操作性示意

由于现网部署 SRv6 的案例很少，因此如果业界能迅速地接纳并部署 uSID，那么未来很可能是第三种互操作场景成为主流。

3.4.2 协议开销 & MTU

节省协议开销能提高转发效率、节省投资。带宽能更有效地用于转发净荷，而不是用于承载报头开销。

节省协议开销的另外一个重要意义是解决 MTU 问题。如果使用 uSID，即便是需要采用 7 个 uSID 编码路径（对于很多应用场景已经足够），其协议开销也只有 40B，与 IPv4 VxLAN 的开销相近。这对于现网部署至关重要，因为很多时候无法配置沿途所有设备的 MTU，而 MTU 不匹配容易导致吞吐下降甚至是流量中断。

下面将针对 uSID vs 常规 SRv6 Segment 以及 uSID vs VxLAN over SR-MPLS 分别进行分析。

3.4.2.1 uSID vs 常规 SRv6 Segment

1. Segment 列表全部使用 uSID

每个 uSID 承载器可以携带 7 个 uSID，因此如果用于编码路径的 uSID 数量不超过 7，则只需要把各个 uSID 依次填入 uSID 承载器对应的位置，并把 uSID 承载器拷贝到 IPv6 报头字段即可，不需要使用 SRH。此时的协议开销是 40B，同样数量的 Segment，采用常规 SRv6 Segment 的协议开销是 16*6+8+40=144B，协议开销节省率=(144-40)/144=72.2%。

如果需要使用超过 7 个 uSID，则需要引入 SRH。例如对于 21 个 uSID 的情况，uSID 协议开销是 16*2+8+40=80B，同样数量的 Segment，采用常规 SRv6 Segment 的协议开销是 16*20+8+40=368B（如此大的协议开销实际上无法使用），协议开销节省率=(368-80)/368=78.3%。

一般地，uSID 的协议开销可以用以下公式来计算:

$$\text{uSID协议开销}=\begin{cases}40 & T<=7\\ 16*[(T-1)/7]+8+40 & T>7\end{cases}$$

其中 T 是 uSID 的总数，[T]表示 T 的整数部分。

因此，uSID 方案相比于常规 SRv6 Segment 的协议开销节省率可用以下公式计算:

$$\text{协议开销节省率}=\begin{cases}(2*T-1)/(2*T+4)*100\% & T<=7\\ ((T-1)-[(T-1)/7])/(T+2)*100\% & T>7\end{cases}$$

2. 混合使用 uSID 和常规 SRv6 Segment

在第 3.3 节所述示例中，使用 uSID 实现低时延 IPv4 L3VPN 业务的协议开销=16*1+8B+40B=64B[6]；如果使用常规的 SRv6 Segment，则协议开销为 16*3[7]+8B+40B=96B。两者对比，uSID 方案的协议开销节省 1/3。

更一般地，假设 Segment 列表的长度为 L(Reduce 操作之前)，其中 K 个 Segment 是 uSID，另外 L-K 个 Segment 是常规 SRv6 Segment，则 uSID 方案相比于 uSID 和常规 SRv6 Segment 混用的协议开销节省率按以下公式计算:

$$\text{协议开销节省率}=\left(\sum_{i=1}^{K} uSID_{\#}-K\right)/\left(\sum_{i=1}^{K} uSID_{\#}+L-K+2\right)*100\%$$

其中 $uSID_{\#}$表示第 i 个 uSID 承载器里包含的 uSID 数量，$uSID_{\#}$<=7。

显然 uSID 承载器越多、uSID 承载器里包含的 uSID 越多，协议开销节省得越多。

如果每个 uSID 承载器里只含有一个 uSID，则退化为常规 SRv6 Segment 的情况。

需要说明的是，以上关于协议开销的计算只是理论上的。事实上当常规 SRv6 Segment 数量超过一定数量后，硬件芯片已经无法在一个处理周期内处理。这时已经不是效率问题了，而是性能问题。

3.4.2.2 uSID vs VxLAN over SR-MPLS

目前，很多用户在数据中心内采用 VxLAN 实现 Overlay，在骨干网采用 SR-MPLS 实现 Underlay，因此当需要跨数据中心提供 SLA 业务时，需要把 VxLAN 封装入 SR-TE。

SRv6 则是天生整合了 Overlay 和 Underlay，无须 在 Overlay 和 Underlay 间进行复杂的交接，这极大地降低了网络复杂性和业务复杂性（详见本文作者发表的 Linux SRv6 系列文章）。而 uSID 则是将 SRv6 这一优势发挥到极致：通过一个 uSID 实现 Overlay，其余 uSID 实现 SR-TE，在多数情况下（用于编码路径的 uSID 数量<=6），甚至不需要 SRH，而只需外层加一个 IPv6 报头!

[6] 后续 uSID 草案会进一步降低协议开销

[7] 总共需要 4 个 Segment（3 个航路点+1 个 End.DT4），这里假设采用了 Reduce 操作，因此 Segment 列表里只包含了后 3 个 Segment（2 个航路点+1 个 End.DT4），第 1 个 Segment 已经被拷贝至 IPv6 报头的目的地址字段。

下图显示了 uSID vs IPv4 VxLAN over SR-MPLS vs IPv6 VxLAN over SR-MPLS 协议开销对比情况:

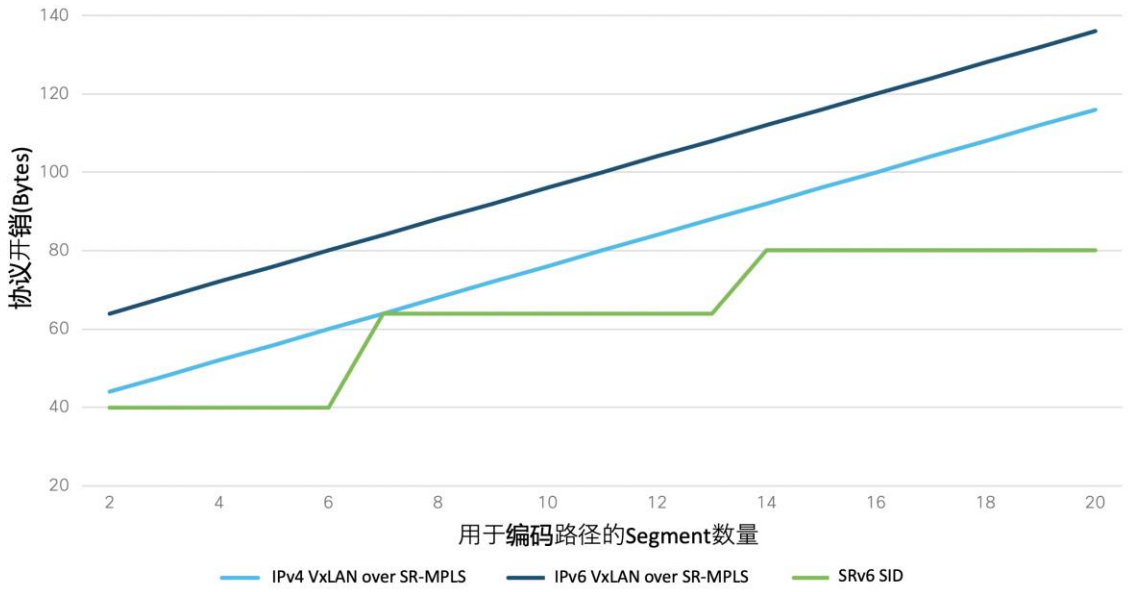

图 7 uSID 与 VxLAN over SR-MPLS 的协议开销对比

由上图可见，SRv6 uSID 在任何情况下的协议开销都要低于 VxLAN over SR-MPLS。尤其是当基础设施迁移至 IPv6，VxLAN 承载在 IPv6 之上时，uSID 的优势更加明显，这个优势随着 Segment 数量的增多不断放大，最大可以达到 50B 左右。请不要小看这 50B 的差异，对于 IMIX 长度的数据包，这意味着 10%的吞吐差异；这也意味着在硬件芯片处理上的巨大差异（是否需要 Recycle），进而影响网络性能；同时也意味着可能在 MTU 处理上的截然不同。

3.4.3 可扩展性

1. uSID 数量的可扩展性

uSID 长度为 16bit，则每个域（uSID 块）支持 uSID 数量=2^{16}=65K。如果觉得 65K 个 uSID 不够，则可以通过分配额外的 uSID 块来获得更多的 uSID。例如 FC00::/16 这个 uSID 块支持 65K 个 uSID，而在 FC/8 里含有 256 个类似的 uSID 块，因此 FC/8 总共可以支持的 uSID 数量=256*65K=14M。

如果还是不足够（例如对于物联网的场景），可以设置 uSID 长度为 32bit，此时每个域(uSID 块)支持的 uSID 数量=2^{32}=4.3B。

可见 uSID 数量具备足够的可扩展性。

2．流量工程路径的可扩展性

uSID 继承了 SR 与生俱来的对海量流量工程路径的支持能力，同时还具备对超长流量工程路径的支持能力。这是因为 uSID 只需要很少的开销就可以支持数十个航路点，而这用常规的 SRv6 Segment 是无法做到的。

当 SRv6 流量工程的头端推向主机，流量工程路径是从一端主机发起至另一端主机时，往往需要许多的航路点来编码路径以穿越数据中心、城域网、骨干网、对等互联网络等，此时 uSID 对超长流量工程路径的支持将发挥关键作用。

3．功能的可扩展性

当前的 uSID 草案中只定义了 uN 操作，在后续草案中会定义更多的 uSID 操作，扩展 uSID 的功能。

另外，uSID 与常规的 SRv6 Segment 可以无缝结合，可以像乐高积木一样，把 uSID 的各种操作和 SRv6 Segment 的各种操作任意组合起来，从而完成各种复杂的功能。

3.4.4 简化控制平面和转发平面

1．控制平面简化

uSID 节点只需通告 uN 操作对应的前缀，网络中其他节点就可通过在路由表中查找 uSID 块+活动 uSID 构成的前缀把数据包路由至 uSID 节点，路由设计非常简单。

无需复杂的映射关系用于把 uSID 对应到可路由的前缀，也无需任何的路由扩展，现有的 IGP/BGP 就可胜任，非常简单。

2．转发平面简化

基于 IP 路由最长匹配进行查找和转发，成熟可靠。

在大多数情况下，IPinIP 就足够，不需要使用 SRH；即使是使用 SRH，uSID 的协议开销也比 VxLAN over SR-MPLS 小，这使得绝大多数硬件可以在一个处理周期完成转发，不用 Recycle，这极大地简化了转发平面的设计和实现。

“Shift & Forward”对硬件要求低，容易实现。事实上思科在 1 年前就可以基于商业芯片 Broadcom Jericho 演示 uSID 的线速转发。

无需查找额外的映射表，简化设计，提高效率。

五、总结与展望

对于尚未部署 SRv6 的网络，我们建议直接部署 uSID；对于少量已经部署 SRv6 的网络，我们建议迁移至 uSID，由于 uSID 支持与 IPv6/SRv6 网络的无缝互操作，因此往 uSID 的迁移可以逐步进行。我们相信未来绝大多数的 SRv6 部署应该是基于 uSID 的，uSID 将是 SRv6 新范式。产业链的聚合有利于加速 SRv6 发展。

我们相信，当 uSID 解锁了 Underlay 的 SRv6 能力后，更多的应用场景将被激活，例如基于 SRv6/uSID 打造 Underlay 和 Overlay 一体的智能网络平台。未来 uSID 发展空间巨大，大有可为。

诚然，uSID 还刚起步，目前业界厂商也只有演示代码实现，尚未正式支持。还需要业界各方一起努力，不断完善 uSID 相关操作，打造强健的生态系统，积极推动现网部署。

未来已来!

【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：VPN、流量工程和服务链（第一篇）：https://www.sdnlab.com/22842.html
5. 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
6. Think in SRv6：极简的编程化网络: https://www.cisco.com/c/dam/assets/global/CN/solutions/industry/segment_sol/enterprise/programs/2018/cin_fy19q1_sp_workshop_ppt.pdf