基于高性能包处理架构 VPP 的带内网络遥测系统

潘恬[1]，林兴晨[1]，张娇[1]，黄韬[1,2]，刘韵洁[1,2]
（1. 北京邮电大学网络与交换技术国家重点实验室，北京 100876；2. 网络通信与安全紫金山实验室，江苏 南京 211111）

摘 要：面向高性能虚拟网络转发架构及设备 VPP，通过重构数据平面的流水线处理模块，设计了基于矢量数据包处理技术的带内网络遥测方案，并在此基础上通过引入源路由机制引导遥测报文走向，实现了一种带内全网遥测的扩展案例。最后，基于虚拟机组网方式，搭建网络拓扑并进行网络性能评测。实验结果表明，该遥测系统能够在 0.13 ms 精度的遥测时间间隔下覆盖被测网络链路，并能以较小代价实时监测虚拟网络的链路拥塞情况。鉴于虚拟网络设备已在数据中心内被广泛部署，该方案通过高精度的虚拟网络测量覆盖，有望为数据中心内多租户网络和网络功能虚拟化的大规模部署提供可靠性保障。

关键词：带内网络遥测；虚拟网络设备；矢量包处理；可编程数据平面
中图分类号：TN92
文献标识码：A
DOI: 10.11959/j.issn.1000−436x.2021016

In-band network telemetry system based on high-performance packet processing architecture VPP

PAN Tian[1], LIN Xingchen[1], ZHANG Jiao[1], HUANG Tao[1,2], LIU Yunjie[1,2]
1. State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
2. Purple Mountain Laboratories, Nanjing 211111, China

Abstract: An in-band network telemetry-based system was built based on VPP, a high-performance virtual network forwarding architecture/device, via reorganizing the data plane pipeline processing modules. Moreover, a network-wide telemetry mechanism was further developed via embedding source routing into the probe packet header for specifying the route of probe packets. Finally, a virtual network topology was built with performance evaluation conducted. The evaluation shows that the telemetry system can monitor the network in a high precision of every 0.13 ms, detecting link congestion in real time with minor performance overhead. Given virtual network devices have been widely deployed in data centers, the proposed scheme is expected to improve the reliability of multi-tenancy and network function virtualization in data centers via high-precision, network-wide virtual link telemetry coverage.

Keywords: in-band network telemetry, virtual network device, vector packet processing, programmable data plane

1 引言

随着云计算技术的飞速发展和云计算市场的持续繁荣，数据中心网络规模迅速增长。为了提升数据中心内硬件资源利用率以及解决高能耗问题，大量虚拟化技术被广泛使用[1]，通过硬件资源动态共享和弹性分配，可以节省硬件购买和运维成本，提高资源整体利用率，降低整个数据中心的能耗开销。数据中心的硬件资源通常包括服务器和连接它们的网络设备。服务器的虚拟化可以使单台物理机的硬件资源被多台虚拟机共享[2]。而网络设备的虚拟化则需要能支撑服务器虚拟化之后引入的大容量虚拟地址表和虚拟

收稿日期：2020−08−04；修回日期：2020−10−26
基金项目：国家重点研发计划基金资助项目（No.2019YFB1802600）；国家自然科学基金资助项目（No.61702049）
Foundation Items: The National Key Research and Development Program of China (No.2019YFB1802600), The National Natural Science Foundation of China (No.61702049)

机在物理机之间的动态迁移[3]。此外，网络设备虚拟化还需要满足不同网络业务之间的流量隔离、隐私控制、安全防护等刚性需求。近年来，基于 x86 等通用硬件开发的虚拟网络设备在数据中心网络内被大量部署[4]。这些虚拟网络设备承载了很多高速网络功能（包括隧道网关、交换机、防火墙、负载均衡器等）的软件处理，支撑并发部署多个互不干扰的网络业务，以满足用户多样化、复杂化、定制化的网络业务需求。其中 OVS（open vswitch）[5]和 VPP（vector packet processor）[6]是两款工业界广泛使用的虚拟网络设备。

随着数据中心网络规模的迅速扩张，网络故障出现的概率急剧上升，且故障感知定位的过程变得复杂而漫长。另一方面，随着数据中心网络定制化业务的逐渐增多，付费用户对网络拥塞所导致的业务服务质量下降也变得更敏感。如何高效地监控和管理数据中心网络流量，快速定位网络故障，开展精细化的流量工程正成为亟待解决的技术挑战[7-8]。在传统网络中，通常使用网络测量的手段来感知网络流量的实时变化。然而，传统的网络测量方案在数据中心网络中的应用面临新的挑战。一方面，在传统的流量监控中，运营商往往通过物理分接端口（例如分光器或流量镜像技术）来采集数据包，这些方案需要在交换机或路由器的物理端口上实施抓包动作。而在数据中心网络中，大多数流量是东西向的，甚至有可能只是从一个虚拟机通过虚拟网络设备传输到位于同一台物理机上的另一个虚拟机，这意味着流量有时不需要离开物理服务器。此时，通过物理分接端口采集数据包的方法就无法满足对虚拟网络设备的监控需求。另一方面，传统的网络测量方案一般采集网络设备中的流量统计信息，例如数据包的接收数、丢弃数、字节量等，所采集的数据类型较少且粒度较粗，无法精确实时地反映流量瞬时的拥塞状态，这就大大限制了网络故障的快速定位以及细粒度网络流量工程的有效实施。

随着协议无关转发体系架构[9]和 P4 数据平面编程语言[10]的提出，带内网络遥测（INT, in-band network telemetry）[11]作为 P4 语言的重要应用之一，能够支持低时延、细粒度的网络设备状态测量。它能将网络设备更加底层的实时状态信息，例如网络设备的包缓存队列长度、数据包的路由信息、到达或离开设备的时间戳信息等，嵌入沿路到达的数据包中，并由数据包随路携带，最终上报给远端的控制器[12]。控制器将基于采集到的实时网络拥塞信息对故障潜在位置或流量优化策略进行判断或规划[7]。由于整个端到端的遥测过程只在数据平面进行，不需要与控制平面进行频繁交互（除了最后一跳的信息采集上报），INT 大大降低了遥测时延和对控制平面的中断频率。虽然带内网络遥测的以上特性显示了其对数据中心网络实时测量的潜在价值，然而带内网络遥测依赖于协议无关转发体系架构，目前仅有少量可编程硬件交换机支持该能力，大多数虚拟网络设备均无法运行 P4 程序，以提供对带内网络遥测的直接支持。

鉴于虚拟网络设备已在数据中心内大范围部署，本文面向工业界流行的高性能开源虚拟网络架构及设备 VPP（作架构讲时，VPP 全称为 vector packet processing；作设备讲时，VPP 全称为 vector packet processor）[6]，给出针对性的带内网络遥测的设计和实现，为在虚拟网络设备中研究网络测量问题提供新的解决思路。具体地，本文基于带内网络遥测协议规范定义了相关数据包的报文格式，基于 VPP 的数据包处理框架，构造了数据平面的流水线处理模块，并在此基础上在包头引入源路由机制，实现了一种带内全网遥测方案。最后，本文基于虚拟机组网方式，搭建网络拓扑并进行了网络性能评测，实验展示了不同背景流量速率和不同探测频率下的遥测性能开销。系统实现包含 1 017 行 C++ 代码，并在 GitHub 网站进行开源。

2 带内网络遥测

2.1 研究背景

网络测量能在网络管理和运维过程中为网络故障快速定位、网络容量合理规划以及网络安全有效防护等任务提供有价值的基础数据。面对日益增长的网络流量以及复杂多样的业务场景，为了更加快速及时地反映流量瞬时变化情况，网络测量应满足精细化和实时性的探测需求，同时尽可能降低网络测量本身的开销以及对被测网络性能的影响，做到轻量化测量。

在传统的网络测量方案中，最常见的方法是基于简单网络管理协议（SNMP, simple network management protocol）[13]实施测量。SNMP 可以从网络设备侧采集设备以及流量的实时统计数据，如报文的接收数、丢弃数和错误数等。SNMP 配置简单且性能开销较低，被长期用于各类网管系统中。然而，SNMP 周期性地轮询网络设备以采集设备内部状态的行为将引入数据平面与控制平面的频繁

交互，进而导致 SNMP 具有较大的查询时延，从而无法对突发网络故障或数据平面的动态流量变化做出及时的反应，也无法对全网链路状态进行精细化的探测覆盖。尽管文献[14-15]分别提出了基于管理信息库（MIB, management information base）表组织模式和 MIB 表查询方式的性能优化，但仍然无法从根本上消除 SNMP 控制平面和数据平面频繁交互的性能瓶颈。谷歌的研究人员在 2018 年甚至提出了“SNMP is dead”的观点。根据前人对 SNMP 的性能测量[16-17]，SNMP 控制平面与数据平面的单次查询时延约为 0.4～21.1 ms；根据本文第 5 节的实验数据，基于 VPP 的带内网络遥测系统能够以 10 Mbit/s 的速度向网络系统注入探测包，从而将设备状态查询间隔降至 0.13 ms。在高速硬件可编程交换机上，带内网络遥测探测包的注入频率还能大幅提高，从而进一步提升网络状态探测精度。除了 SNMP，还有其他的一些基于被动探测方式的网络测量工具，例如 NetFlow[18]、sFlow[19]、IPFIX[20-21]等。这些工具基于 IP 流进行统计，即网络设备把报文划分为不同的流，并统计每个流的包数、字节数等信息。当流内报文传输完毕或流记录更新超时后，该流的统计结果就会被上传到收集器以便后续数据分析。基于流统计的测量方法能够有效支撑业务流量的分析、统计和计费，但它们无法获取网络设备更加底层的实时状态信息，例如包缓存的队列长度、数据包的路由信息、包到达或离开设备的时间戳等。此外，微软公司在 2015 年提出的 Pingmesh 技术能够基于端到端发送的 Ping 包，形成对整个数据中心网络的覆盖[22]。然而，Pingmesh 只能采集端到端的拥塞信息，仍然无法进一步监测网络中间设备的内部状态。综上所述，传统的网络测量方法数据采集时延较大，且可采集的网络实时数据类型较少，或无法支持在网络设备侧采集，这就大大限制了网络故障快速定位以及细粒度网络流量工程的有效实施。

随着协议无关转发体系架构和 P4 编程语言的提出，网络数据平面的可编程能力被极大释放。这使网络设备在理论上能够任意解析和编辑报文头部。随后，越来越多的 P4 程序被提出，它们定义了数据平面包处理的诸多有趣功能[11,23-26]。INT 就是其中的一个高级应用。INT 允许数据包在通过数据平面包处理流水线时，查询并携带网络设备的内部状态，例如队列长度和排队时延等。也就是说，INT 可以采集端到端转发路径上每一跳网络设备的内部状态。同时，整个状态采集过程只在数据平面进行，与控制平面尽量解耦，最终由末梢网络设备或接收终端将遥测结果上报给收集器。这种测量方式既能采集逐跳网络节点上的多种内部状态，又不需要频繁与控制平面发生交互，所以整体的测量时延大大降低。INT 自提出以来一直由 P4 联盟的工作小组维护相关的协议规范[27]。目前，INT 正受到华为、Broadcom、Barefoot 等多家设备和芯片厂商的关注。一些厂商已将 INT 功能嵌入其最新的商用交换芯片中[28-29]。

2.2　带内网络遥测原理

INT 实施框架如图 1 所示，该框架定义了以下技术术语。

1) INT 头部，即 INT 定义的报文首部字段。INT

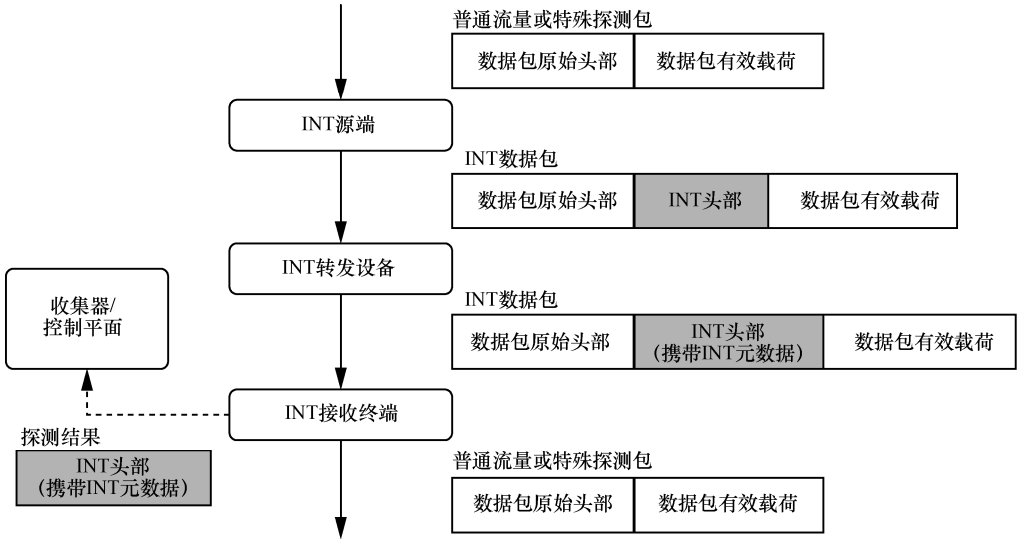

图 1　INT 实施框架

头部既可用于携带收集到的网络状态信息，又能告知支持INT的设备需要收集哪些信息以及怎样将信息写入报文中。目前 INT 规范并未强制规定 INT 头部在报文中的插入位置，只要所插入的位置能够提供足够的容纳空间即可。例如，可将 INT 头部作为 TCP/UDP 层的有效载荷，还可以基于 VXLAN GPE 协议对 INT 头部进行封装。

2) INT 元数据，即所采集到的网络状态信息。INT 元数据将被写入 INT 头部的特定位置。

3) INT 数据包，即携带 INT 头部的报文。普通流量或特殊探测包都可以被标记为 INT 数据包。若把普通流量标记为 INT 数据包，则可以进行指定用户流量的随路监测或随流监测。

4) INT 源端，即能创建 INT 数据包的实体，例如应用程序、终端主机网络协议栈、物理网卡或与发送方直接相连的 ToR（top of rack）交换机等。INT 源端负责把 INT 头部封装到数据包中。

5) INT 接收终端，即能提取 INT 头部并解析其中携带的 INT 元数据的实体（通常既可以是最后一跳网络设备，也可以是终端主机）。INT 接收终端接收 INT 数据包，解析 INT 头部，生成探测结果，并将探测结果上报给收集器。

6) INT 转发设备，即根据 INT 头部的指示，采集 INT 元数据并将其写入 INT 头部的网络设备。

INT 采用带内测量模式，其遥测过程如图 2 所示。INT 源端把 INT 头部插入原始数据包中，从而指示沿途网络设备添加所期望采集的 INT 元数据。在网络传输过程中的每一跳，INT 转发设备将根据 INT 头部的指示，在其中插入采集到的各个 INT 元数据。INT 接收终端在处理原始数据包之前会提取携带 INT 元数据的 INT 头部，并生成探测报告。该探测报告包含了数据包经过沿途网络设备时所采集到的逐跳实时网络状态信息。同时 INT 接收终端还会把探测报告发送给收集器。收集器一般部署在集中式控制器上，负责收集并分析从数据平面采集到的 INT 元数据，为控制平面后续的决策动作（如流量工程）提供有效的数据支撑。根据 INT 协议规范[27]，理论上 INT 自身并不约束支持采集的数据类型。但受到网络设备的实际硬件限制，最终 INT 能够采集到的元数据通常包括报文出入端口号（用于分析报文端到端的转发路径）、报文处理时延和设备内部排队状态（用于感知端到端的逐跳拥塞情况）。

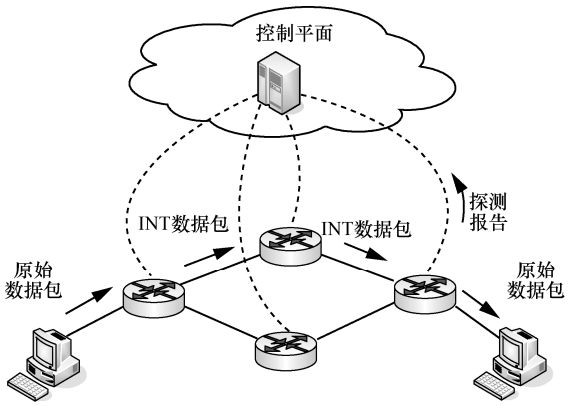

图 2 INT 的遥测过程

3 矢量数据包处理

3.1 研究背景

在数据中心网络中，虚拟网络设备被大量使用。一方面，高可靠、低时延、高吞吐率等多样化的应用需求要求网络设备支持差异化的数据包转发方式，例如多路径路由、最短路径路由、源路由等。而虚拟网络设备支持数据平面可编程，能灵活修改数据包转发方式，这满足了多样化的应用需求。另一方面，数据中心包含了海量服务器的互联，并通过虚拟机和物理机的形式对外提供服务，且不同服务之间还存在流量隔离、隐私控制、安全防护等需求。虚拟网络设备则可以通过 x86 等通用硬件以及虚拟化技术来承载诸多网络功能的软件处理，从而能够在一台物理设备上同时运行多个性能与故障互相隔离的实例，因此可以支持多个网络业务的并发部署，达到硬件资源共享的目的。

目前，已有多个开源项目提出了多种网络设备虚拟化的解决方案。OVS[5]是一个开放的多层虚拟软件交换机，其设计目的是通过软件编程的方式实现大规模的网络自动化部署，非常适合作为软件交换机部署在虚拟机管理平台上。VPP[6]是一种高效的数据包处理架构，基于该架构核心思想实现的高性能转发设备能够运行在通用 CPU 上，实现可供生产环境使用的高速虚拟软件交换机或路由器。VPP 最早由思科公司开源，发展至今已成为 Linux 基金会的开源项目 FD.io 中最核心的组件。此外，还有专门用于 P4 程序验证的开源仿真平台 BMv2[30]，它实现了具备协议无关转发架构的软件交换机，并通过运行 P4 程序，产生程序定义的转发行为。以上 3 种典型的开源虚拟网络设备均具备

良好的编程扩展能力以及一定的数据包处理性能，所以正受到越来越多网络开发者的关注。当需要为报文处理逻辑扩展新的功能或自定义新协议时，对于 OVS，需要修改其核心软件代码或定义流表规则；对于 VPP，由于其本身采用了模块化的设计思想，因此能够独立于核心代码创建新的协议功能逻辑；至于 BMv2，它是基于 P4 的转发引擎，所以只需编写并加载对应功能的 P4 代码，天然具备协议扩展能力。另一方面，在报文处理性能上，VPP 采用矢量化包处理技术，大幅降低包转发开销，所以包处理性能优于 OVS；而 BMv2 主要关注如何实现 P4 程序定义的协议无关转发功能，所以包处理性能并非它优化的重点，其处理性能远低于 VPP 和 OVS[8]。综上所述，本文拟基于 VPP 实现带内网络遥测，以求兼具高性能和可扩展性。

3.2　VPP 数据包处理架构

VPP 基于批处理思想提升报文处理效率，同时基于模块化的架构解耦添加新协议逻辑的复杂度。VPP 数据包处理架构如图 3 所示。首先，VPP 从网络 I/O 收取输入流量并将到达的数据包组织成一个个的包集合，即“数据包矢量”。然后，输入流量以数据包矢量为单位，一批批通过 VPP 的包处理结构来执行数据包的批量操作。该包处理结构是由多个图节点构成的有向图，而每个图节点则代表数据包处理流程中的某一个操作环节。通常来说，图节点可分为 2 种类型：一种是包含收发数据包操作的节点，如 dpdk-input 节点是基于 DPDK 接收数据包的节点；另一种是处理数据包的节点，如 ethernet-input 节点是解析数据包以太网头部的节点。此外，值得注意的是，VPP 对数据包的处理顺序不再是传统的标量方式（即将数据包逐个送入图节点进行处理），而是把整个数据包矢量整体送入当前图节点进行处理，处理完成后再整体送入下一个图节点。使用矢量处理相较标量处理的好处是，若以标量方式处理数据包，则每处理一个包都需要重新逐层调用协议栈处理函数，这很容易产生 CPU 指令缓存的访问时延抖动，即每个包被处理时都有可能产生一组相同的指令缓存缺失事件，这将大大影响处理效率。若以矢量方式处理，那么在一个图节点的处理过程中，由于数据包矢量中第一个报文有缓存预热作用，因此能大概率地减少包矢量中后续数据包处理时的指令缓存缺失事件，也就能以一组数据包来分摊第一个数据包引发的指令缓存缺失开销，从而整体提升数据包矢量通过图节点的效率。

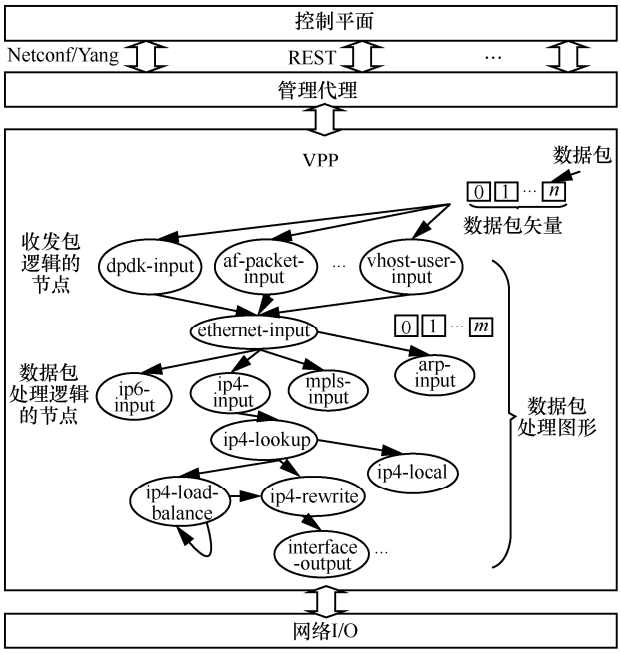

图 3　VPP 数据包处理架构

VPP 数据包处理架构具备良好的可扩展性。具体而言，VPP 数据包处理架构中模块化的图结构有助于实现协议功能的快速迭代更新。如图 4 所示，若开发者希望定义新的报文处理逻辑，只需在程序库中为 VPP 编译生成一个独立的二进制插件，并在系统中加载该插件，就能重新排列包处理图节点或引入新的图节点。该机制允许通过插件更新的方式引入新的包处理逻辑，而不需要修改 VPP 自身的核心代码。这对网络数据平面协议功能扩展来说是非常便捷的。

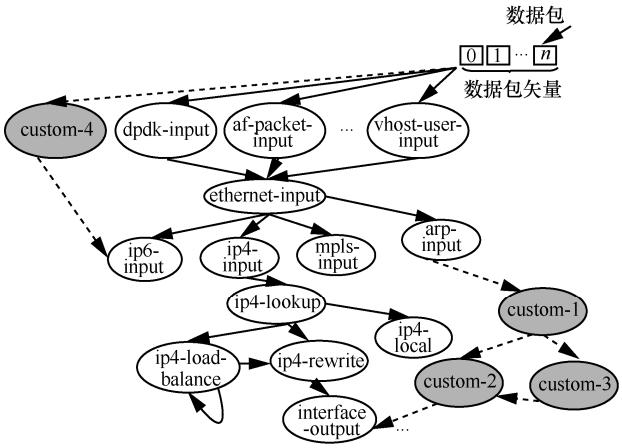

图 4　VPP 数据包处理架构的图结构的扩展

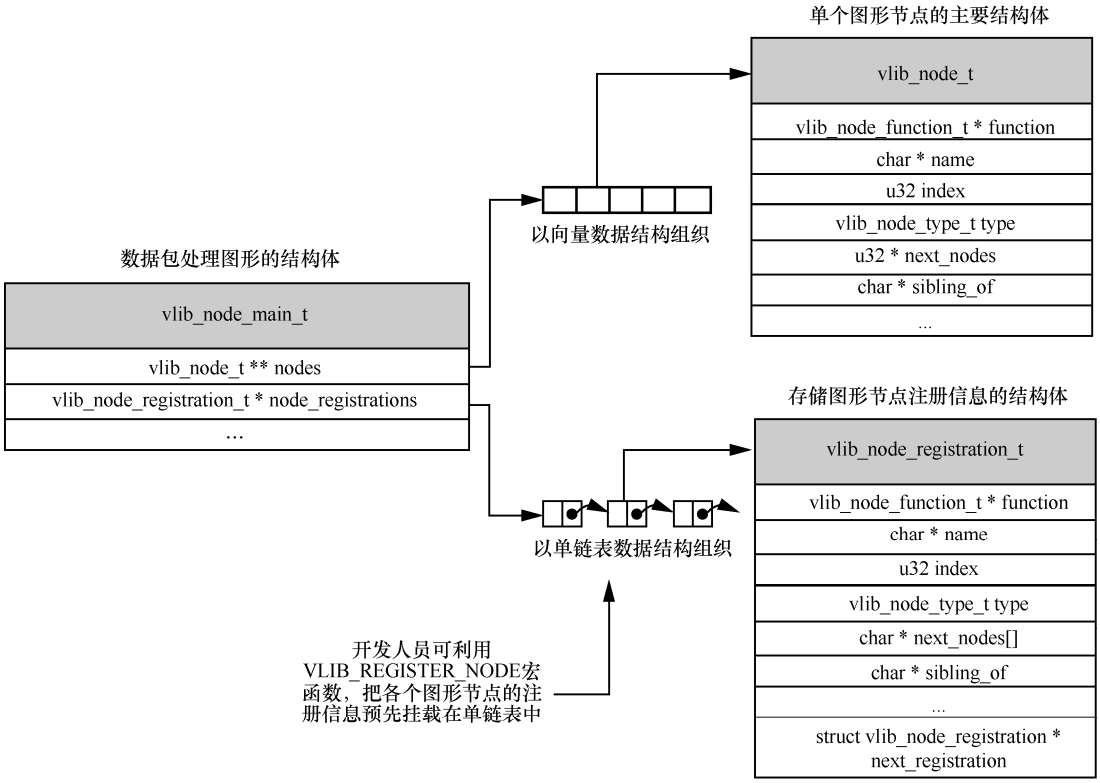

图 5　VPP 数据包处理架构的图结构的主要数据结构

3.3　VPP 图形节点结构及其初始化

如前所述，如果开发人员想要开发自定义的包处理逻辑，就需要在 VPP 数据包处理架构的图结构中建立新的图节点。

如图 5 所示，VPP 数据包处理架构的图结构由 vlib_node_main_t 结构体定义，该结构体记录了包处理图的全局信息。其中比较重要的 2 个成员变量分别为 nodes 和 node_registrations。成员变量 nodes 以向量的数据结构进行组织，存储了多个指向 vlib_node_t 结构体类型的指针，用来保存所有在 VPP 中已注册的图节点；成员变量 node_registrations 则以单链表的数据结构进行组织，链表中的各个元素类型为 vlib_node_registration_t 结构体，提供给 VPP 的初始化函数使用，用于图节点的注册。

具体来看，vlib_node_t 结构体是单个图节点的主要数据结构，保存了单个图节点的名称、节点类型、处理函数、兄弟节点及子节点等信息。vlib_node_registration_t 结构体用于图节点的注册，开发人员可以利用 VPP 提供的宏函数（VLIB_REGISTER_NODE），把各个图节点预先挂载到以单链表组织的 node_registrations 成员变量中，以供后续的初始化函数使用。图 6 是通过 VLIB_REGISTER_NODE 宏函数注册 ethernet-input 节点的示例，它定义了该图节点的处理函数等一系列注册信息。

```
vlib_node_registration_t ethernet_input_node;
/* *INDENT-OFF* */
VLIB_REGISTER_NODE (ethernet_input_node) = {
  .function = ethernet_input,
  .name = "ethernet-input",
  /* Takes a vector of packets. */
  .vector_size = sizeof (u32),
  .scalar_size = sizeof (ethernet_input_frame_t),
  .n_errors = ETHERNET_N_ERROR,
  .error_strings = ethernet_error_strings,
  .n_next_nodes = ETHERNET_INPUT_N_NEXT,
  .next_nodes = {
#define _(s,n) [ETHERNET_INPUT_NEXT_##s] = n,
    foreach_ethernet_input_next
#undef _
  },
  .format_buffer = format_ethernet_header_with_length,
  .format_trace = format_ethernet_input_trace,
  .unformat_buffer = unformat_ethernet_header,
};
/* *INDENT-ON* */
```

图 6　通过 VLIB_REGISTER_NODE 宏函数注册 ethernet-input 节点示例

VPP 在启动之后，将对各个图节点进行注册，并创建整个数据包处理架构的图结构。相关的初始化流程函数为 vlib_main()，图 7 显示了该函数的主要流程。vlib_main() 函数首先调用 vlib_register_all_static_nodes() 函数，遍历以单链表组织的 node_

registrations 成员变量，从而创建各个图节点，并把它们添加到以向量组织的节点成员变量中。接着，继续调用 vlib_node_main_init()函数，它会根据各个图节点所注册的兄弟节点及子节点等信息，建立节点之间的跳转关系，由此，创建出一张完整的数据包处理图。最后，在调用其他一些初始化配置函数之后，VPP 主程序将进入 vlib_main_loop()函数，开始执行收发数据包和图节点调度的流程。

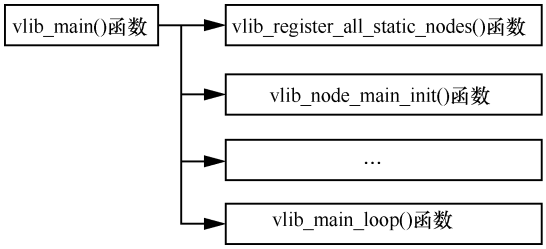

图 7　VPP 数据包处理架构的图结构的主要初始化流程

基于 VPP 提供的包处理图节点抽象机制，开发人员能够把更多的精力投入自定义图节点中协议逻辑功能的设计和实现里。

4　基于 VPP 的带内网络遥测实现

4.1　设计思路

由 INT 的实施框架可知，INT 元数据可以由用户流量携带，也可以由专门的探测包携带，二者在实现上的区别主要在于 INT 源端封装 INT 数据包的方式。目前，本文采用由专门的探测包携带 INT 数据的实现方式，即标记特殊探测包为 INT 数据包。基于此，使用额外的发包终端来发送特殊的 UDP 探测包。该 UDP 探测包被第一台 VPP 虚拟网络设备接收后，将被封装 INT 头部，从而生成真正的 INT 数据包。该台 VPP 虚拟网络设备实际上扮演了 INT 源端的功能角色。

若后续的 VPP 虚拟网络设备接收到 INT 数据包，作为 INT 转发设备，它们需要依据 INT 头部的指示，统计设备自身的 INT 元数据，并将其写入 INT 头部的对应位置，由此完成数据采集工作。

由于目前 VPP 虚拟网络设备暂时无法从自身判断 INT 数据包是否已被转发至最后一跳节点，因此也无法完成 INT 接收端的功能。本文将指定端侧服务器作为数据包接收终端，负责解析 INT 头部中记录的各个 INT 元数据。

接下来，本文基于 IPv4 和 INT 协议规范，阐述在 VPP 中设计和实现带内网络遥测的技术细节。

4.2　系统设计与实现

4.2.1　数据包格式

在标记特殊探测包为 INT 数据包的实现方式中，共出现 2 种数据包，一种是特殊 UDP 探测包，另一种是 INT 数据包。它们的包格式如下所述。

特殊 UDP 探测包由额外发包终端向 VPP 虚拟网络设备发送，使用携带特殊目的端口号（55555）的 UDP 包封装，表示它是生成 INT 数据包的原生探测包。当 INT 源端识别到携带该端口号的 UDP 包后，将其封装成 INT 数据包。此外，它的目的 IP 地址为所发往的 VPP 虚拟网络设备的网络地址。

INT 数据包的报文格式基于 IPv4 协议和 INT 规范设计，如图 8 所示。

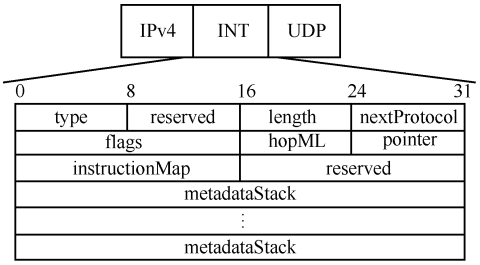

图 8　INT 头部报文格式

1) 8 bit 的 type 字段表示 INT 数据包的类型，本文定义“1”是基于特殊 UDP 探测包封装的 INT 数据包。

2) 8 bit 的 length 字段记录了 INT 头部的长度。

3) 8 bit 的 nextProtocol 字段用于区分 INT 头部封装的上层协议，它复制了原始 IPv4 头部中的协议字段号（protocol）。这是因为，本文把 INT 头部放置在 IP 层之后，使用自定义的协议号 200 来表示 IP 层之后的上层协议是 INT，所以 IP 头部中原始的协议字段号需要重新被记录在 INT 头部中，以指示 INT 头部之后的上层协议类型。

4) 16 bit 的标志位（flags）字段，包括 INT 协议的版本号等标志位信息。

5) 8 bit 的 hopML 字段，表示每一跳节点需要添加的 INT 元数据的最大长度。

6) 8 bit 的 pointer 字段，指向在 INT 元数据堆栈中下一个可填充 INT 元数据的空白空间地址。

7) 16 bit 的 instructionMap 字段，其每 bit 代表一种 INT 元数据类型，指示了每跳节点需要统计的各个 INT 元数据类型的组合，例如交换机标识号、

出入端口时间戳、包缓存队列长度等。

8) 后续的 metadataStack 即 INT 元数据堆栈，记录了在每一跳节点上采集到的各个 INT 元数据。

可以看出，对于一条端到端探测路径，如果总共有 n 跳节点，且每一跳节点需要添加的 INT 元数据的最大长度为 hopML B，则 INT 头部所占空间为（12+$n$hopML）B。

4.2.2　INT 转发设备功能

INT 转发设备完成接收 INT 数据包，采集设备本地的 INT 元数据并将其写入 INT 头部，转发 INT 数据包的功能。

如前所述，VPP 的数据包处理架构以数据包矢量为单位，通过包处理架构的图结构对包矢量进行统一处理。本文根据转发流程设计了 INT 数据包处理的 VPP 图结构，如图 9 所示。

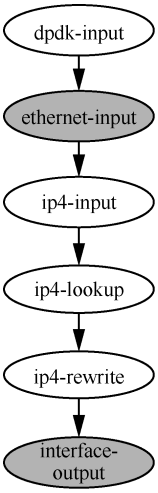

图 9　INT 数据包处理的 VPP 图结构

具体而言，图 9 的图结构扩展了 VPP 已经提供的处理普通 IPv4 数据包的简单流程。

1) VPP 虚拟网络设备通过收发数据包逻辑的相应节点接收 IPv4 数据包（如 dpdk-input 节点）。

2) 解析数据包的以太网头部（ethernet-input 节点）。

3) 解析数据包的 IPv4 头部（ip4-input 节点）。

4) 依据解析结果，查找路由表项（ip4-lookup 节点）。

5) 依据出端口信息，更新数据包头部对应的字段值（ip4-rewrite 节点）。

6) 转发数据包（interface-ouput 节点）。

为了进一步在 IPv4 协议之上实现 INT 元数据的采集工作，需要对若干原有节点的逻辑功能做相应的扩展。本文在原有的 ethernet-input 节点和 interface-output 节点中增加了部分代码，使数据包能在刚进入数据平面流水线进行处理以及处理结束时，分别采集所需的 INT 元数据。

本文采集了 3 种类型的 INT 元数据，如下所示。

1) 入端口时间戳：ingress_timestamp_s 和 ingress_timestamp_us，秒和微秒级，分别占 4 B。

2) 出端口时间戳：egress_timestamp_s 和 egress_timestamp_us，秒和微秒级，分别占 4 B。

3) 出端口 MAC 地址：switch_addr，占 6 B。

以上元数据共占 22 B，所以此时每一跳节点需要添加的 INT 元数据的最大长度为 22 B。

图 10 展示了采集 INT 元数据的程序流程，具体的 INT 元数据采集流程描述如下。

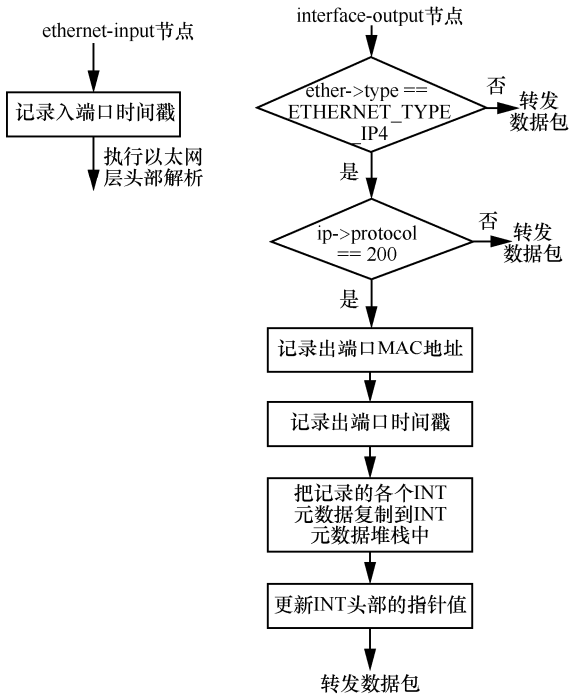

图 10　采集 INT 元数据的程序流程

在 ethernet-input 节点中执行原有解析动作之前，首先记录入端口时间戳。当数据包完成以太网层和 IP 层的流水线处理之后，进入 interface-output 节点等待转发。此时，首先判断数据包是否为 INT 数据包（ether->type == ETHERNET_TYPE_IP4 且 ip->protocol == 200），若是，则继续完成下列动作，否则直接转发数据包。当判断是 INT 数据包后，分别记录出端口 MAC 地址以及出端口时间戳。接着，根据 INT 头部的指令信息（即 instructionMap 字段）

和指针值（即 pointer 字段），把指令要求采集的各个 INT 元数据复制到 INT 元数据堆栈中的空白区域。最后，更新指针值，并继续执行数据包的转发动作。

4.2.3　INT 源端功能

INT 源端接收特殊 UDP 探测包，封装 INT 头部以生成 INT 数据包。

同样地，本文设计了如图 11 所示的 INT 源端包处理架构的图结构。具体而言，当 INT 源端接收到特殊 UDP 探测包后（dpdk-input 节点），首先执行以太网头部（ethernet-input 节点）和 IPv4 头部（ip4-input 节点）的解析操作；当解析得到的 IP 目的地址为本设备地址，且 IP 层头部协议字段号为 UDP 时，将继续解析数据包的 UDP 层（ip4-local 节点和 ip4-udp-lookup 节点），获取 UDP 头部的目的端口号；接着，根据自定义的目的端口号（55555），执行数据包头部的重新封装操作（int-probe-packet-generation 节点），使其成为 INT 数据包；最后，由于此时 INT 源端也是 VPP 虚拟网络设备，也需采集本网络设备的状态信息（即也需完成 INT 转发设备的功能），因此生成的 INT 数据包会被重新送到以太网头部的解析节点（ethernet-input 节点），继而执行前文所述的 INT 转发设备处理流程，即采集 INT 元数据并完成转发。

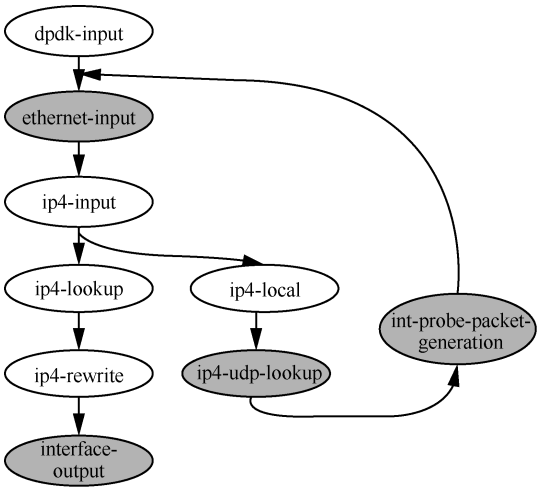

图 11　INT 源端包处理架构的图结构

受限于数据包大小，INT 源端无法在 INT 头部中预留无限大的空白 INT 元数据堆栈空间。这就希望可以由用户指定遥测过程的最大转发跳数以及所采集的 INT 元数据类型，基于此计算预留空间的大小。因此，除了封装 INT 头部之外，INT 源端还应提供相应的配置接口。

综上，本文把 INT 源端的功能实现划分为以下 2 个模块，即指令配置模块和 INT 数据包生成模块。

1) 指令配置模块

指令配置模块负责提供配置指令和初始化 INT 头部模板。配置指令可暴露给用户或控制平面调用，根据配置参数来计算 INT 头部中需要预留的空白 INT 元数据堆栈空间大小。INT 头部模板则在配置指令下发后被初始化，以便 INT 数据包生成模块据此模板来重新封装 UDP 探测包，生成 INT 数据包。

借助 VPP 提供的开发接口，可以实现 CLI 配置指令以及指令响应函数。指令格式为 int header <add> <maxhop hop_value> <insmap map_value>。其中，add 表示该指令是 INT 头部配置指令；maxhop 表示遥测过程的最大跳数；insmap 表示每跳节点需要统计的 INT 元数据类型，对应 INT 头部格式中 16 bit 的 instructionMap 字段。

指令响应函数的程序流程如图 12 所示。首先，解析 CLI 配置指令，得到各参数值；接着，先删除原有的 INT 头部模板（即上一轮创建的头部模板全局变量），再依据指令参数值，生成新的模板。

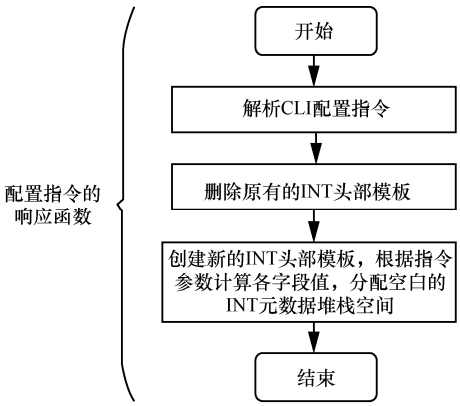

图 12　指令响应函数的程序流程

2) INT 数据包生成模块

当接收到特殊 UDP 探测包后，该模块依据 INT 头部模板，对探测包执行重新封装操作，以生成真正的 INT 数据包。如图 11 所示，本文在包处理架构的图结构中增加了新的图节点（即 int-probe-packet-generation 节点，该节点的注册函数如图 13 所示），并在 ip4-udp-lookup 节点中通过解析 UDP 目的端口号是否为 55555 来判断是否跳转

到该新节点进行处理。INT 数据包生成模块的程序流程如图 14 所示。其中，本文借助 VPP 提供的开发接口，为自定义的 UDP 目的端口号注册了新的节点跳转关系。此外，在 int-probe-packet-generation 节点中，首先，为 UDP 探测包分配足够的 INT 头部空间；其次，将 INT 头部模板中的内容复制到 UDP 探测包对应的空间中；接着，更新 UDP 探测包 IPv4 头部中的 protocol 字段值为 200（标识上层协议为 INT），并将原始值记录到 INT 头部中；最后，重新计算 IPv4 头部校验和，再把数据包发往 ethernet-input 节点，以便继续执行后续的 INT 元数据采集工作。

```
vlib_node_registration_t int_probe_packet_generation_node;
/* *INDENT-OFF* */
VLIB_REGISTER_NODE (int_probe_packet_generation_node) = {
  .function = int_probe_packet_generation,
  .name = "int_probe_packet_generation",
  /* Takes a vector of packets. */
  .vector_size = sizeof (u32),
  .type = VLIB_NODE_TYPE_INTERNAL,
  .n_errors = INT_PROBE_PACKET_N_ERROR,
  .error_strings = int_probe_packet_error_strings,
  .n_next_nodes = INT_PROBE_PACKET_N_NEXT,
  .next_nodes = {
#define _(s,n) [INT_PROBE_PACKET_NEXT_##s] = n,
    foreach_int_probe_packet_next
#undef _
  },
};
/* *INDENT-ON* */
```

图 13 int-probe-packet-generation 节点的注册函数

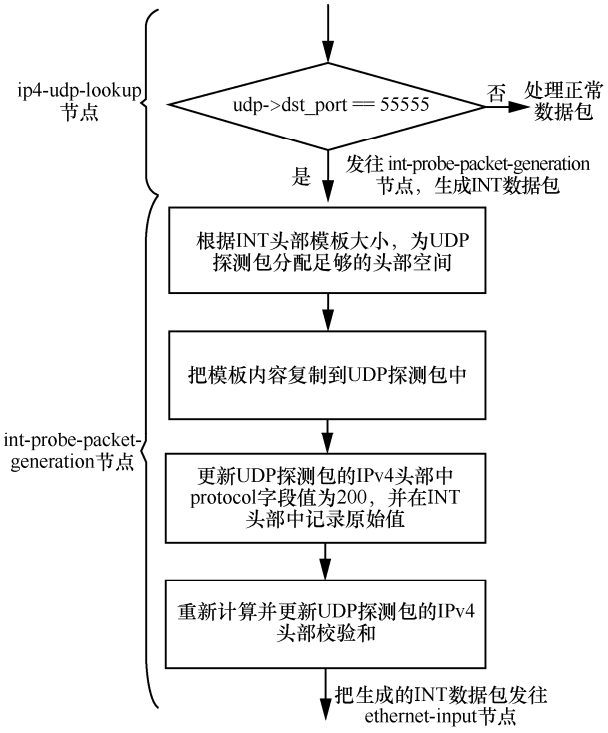

图 14 INT 数据包生成模块的程序流程

4.3 扩展案例：带内全网遥测

4.3.1 带内全网遥测机制简介

INT 数据包的遥测路径取决于该数据包如何在网络中被转发，即取决于网络设备中的路由表项。若想由用户控制遥测路径，则可以为 INT 数据包添加源路由[31]，使 INT 数据包可以预先被指定转发时的下一跳地址。基于这种源路由机制可以主动规划全网范围的遥测路径，以实时获取覆盖全网的所有网络设备及链路的状态信息[32]。下面，本文基于 VPP 给出一种带内全网遥测的实现案例。

图 15 给出了一种带内全网遥测机制，该机制包含数据平面和控制平面的实现。在数据平面上，INT 源端向网络周期性地发送 INT 数据包。这些 INT 数据包同时还会被封装源路由标签栈。当 INT 转发设备接收到它们时，除了采集 INT 元数据并写入 INT 头部之外，还需解析源路由标签栈，获取下一跳 INT 转发设备的地址。综上，INT 数据包在网络中的遥测路径由其携带的源路由信息唯一指定。

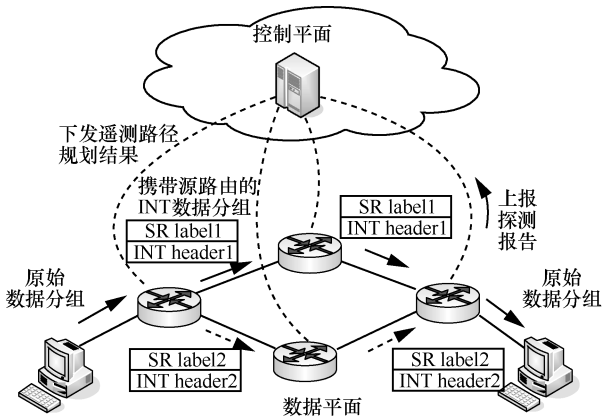

图 15 一种带内全网遥测机制

源路由标签栈的内容则需由网管人员或控制器预先进行规划。也就是说，控制平面除了分析遥测结果之外，还要借助完整的网络拓扑图进行全局遥测路径的设计。然后，控制平面把计算出的各条遥测路径下发到数据平面的 INT 源端。INT 源端据此信息封装 INT 数据包中的源路由标签栈。

4.3.2 方案实现思路

基于前文设计的基于 VPP 的 INT 功能方案，本节阐述带内全网遥测机制的实现思路。

为了在 IPv4 数据包中增加源路由信息，可借助 IPv4 头部的可选字段（option）。图 16 展示了源路由标签栈的报文格式，介绍如下。

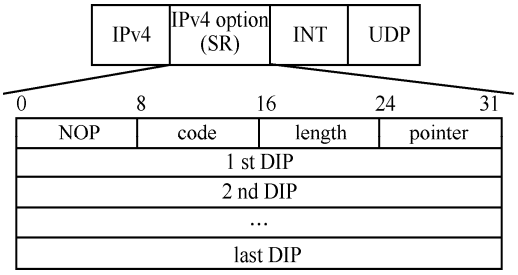

图 16　源路由标签栈的报文格式

1) 8 bit 的 NOP 字段是填充字符，使后面的字段能与 4 B 长度对齐。

2) 8 bit 的 code 字段标识了可选字段的选项类型，其中 137 表示严格的源路由选项。

3) 8 bit 的 length 字段记录了整个可选字段的长度。

4) 8 bit 的 pointer 字段记录了指向下一个可使用的路由地址的指针。

5) 后续的 DIP 标签栈以 4 B 为单位，记录着数据包在网络中传输时被指定的每一跳节点的网络地址。

对于 INT 转发设备，为了解析源路由标签栈，可以在包处理架构的图结构中增加新的图节点（如图 17 所示的 int-sr-forwarding 节点，该节点的注册函数如图 18 所示）。新的图节点位于 ip4-input 节点和 ip4-lookup 节点之间，其程序流程如图 19 所示，具体描述如下。在 ip4-input 节点中解析 IPv4 头部之后，判断其 protocol 字段值是否为 200（自定义协议号，表示上层为 INT 协议），若是，则把数据包发往新增加的 int-sr-forwarding 节点；否则说明该数据包为普通数据包，继续执行 IP 层后续的操作。在 int-sr-forwarding 节点中，首先分别判断 IPv4 头部中的可选字段是否为源路由选项（sr->code == 137），以及可选字段存储的指针是否已经超出头部长度（sr->pointer < sr->length）；其次，根据指针值获取源路由标签栈中指定的下一跳 IP 地址，并将其复制到 IPv4 头部的目的地址字段中；接着，更新可选字段中的指针值；最后，把数据包发往 ip4-lookup 节点，继续执行后续的路由表查找操作。

对于 INT 源端，除了为特殊 UDP 探测包添加 INT 头部之外，还需为其封装源路由标签栈。由于前文已经在 INT 源端中设计了指令配置模块和 INT 数据包生成模块，因此只要对这 2 个模块进行适当的扩展，就能支持源路由标签栈的封装操作。

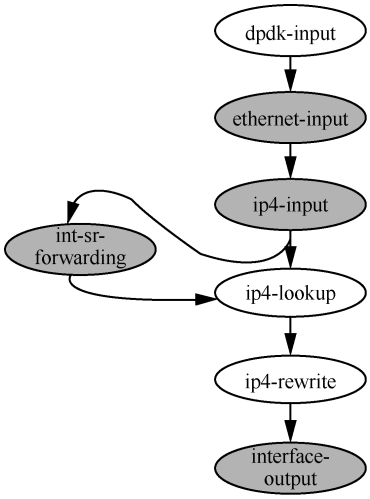

图 17　扩展案例中 INT 转发设备报文处理逻辑对应的 VPP 图结构

```
vlib_node_registration_t int_sr_forwarding_node;
/* *INDENT-OFF* */
VLIB_REGISTER_NODE (int_sr_forwarding_node) = {
  .function = int_sr_forwarding,
  .name = "int_sr_forwarding",
  /* Takes a vector of packets. */
  .vector_size = sizeof (u32),
  .type = VLIB_NODE_TYPE_INTERNAL,
  .n_errors = INT_FORWARDING_N_ERROR,
  .error_strings = int_forwarding_error_strings,
  .n_next_nodes = INT_FORWARDING_N_NEXT,
  .next_nodes = {
#define _(s,n) [INT_FORWARDING_NEXT_##s] = n,
    foreach_int_forwarding_next
#undef _
  },
};
/* *INDENT-ON* */
```

图 18　int-sr-forwarding 节点的注册函数

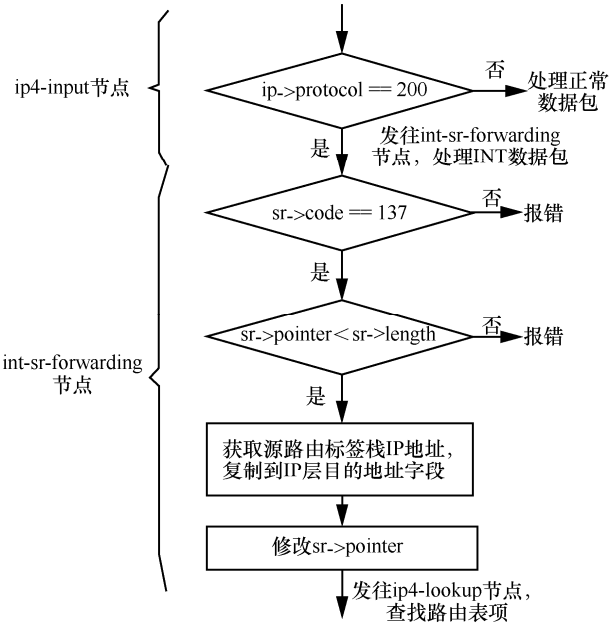

图 19　INT 转发设备解析源路由标签栈的程序流程

一方面，由于源路由标签栈的内容由控制平面下发，因此可以扩展指令配置模块中的指令，以暴露给控制平面调用。增加源路由信息后的配置指令格式为 int header <add> <next ip4_addr1, next ip4_addr2, ⋯> <maxhop hop_value> <insmap map_value>。

新格式中增加了 next 参数，它按序记录了源路由中每一跳节点的 IP 地址。相应地，在指令响应函数的程序流程图中，也需解析 next 参数值，据此创建源路由标签栈模板，并在模板中填充指定的各跳节点的地址。

此外，在 INT 数据包生成模块中，新增加的 int-probe-packet-generation 节点除了为特殊 UDP 探测包封装 INT 头部之外，还应该继续在 IPv4 头部的 option 字段中复制源路由标签栈模板的内容，使该 INT 数据包能够携带源路由信息。

5　系统功能和性能评测

5.1　实验环境

基于 VPP，本文实现了上述带内全网遥测机制。系统包括 1 017 行 C++代码，并在 GitHub 网站开源。下面，基于虚拟机组网的方式，对系统进行网络仿真实验，以验证遥测结果的有效性。仿真实验环境包括一台具有 Intel i5-6500 四核 CPU、16 GB 内存以及 1 TB 硬盘的台式计算机。在这台物理机上，本文又基于 VirtualBox 虚拟机，创建了如图 20 所示的网络拓扑。该拓扑由 3 个 VPP 虚拟网络设备（VPP_1、VPP_2、VPP_3）、一个控制面终端、一个 UDP 发包终端（P_1）以及 2 个主机终端（H_1、H_2）组成。它们分别运行在具有单核 CPU、2 GB 内存、Ubuntu 16.04 操作系统的 VirtualBox 虚拟机中，并使用由虚拟机提供的虚拟网卡以及桥接模式进行组网。

5.2　实验场景及目的

基于不同的实验场景，本文希望回答以下问题。1) 单个 VPP 节点在转发普通流量和 INT 流量时的性能怎么样？本文也将比较 VPP 与同类软件（如 OVS 和 BMv2）的转发性能差异。2) INT 探测包能否真正地感知到因为背景流量速率变化引发的网络拥塞？且 INT 探测包自身的采样频率设置为多少是合适的？其中，实验场景 2)的背景流量速率设置会参考当前实验环境的虚拟网络链路最大带宽的测量结果。

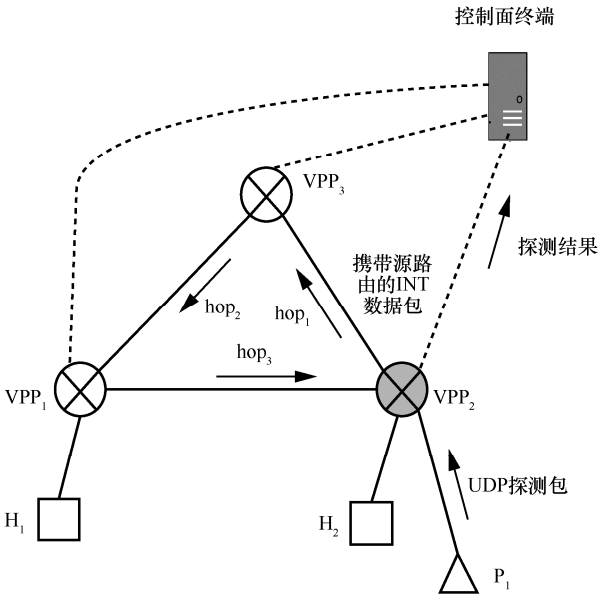

图 20　仿真实验网络拓扑

5.3　测试流量生成方法及流量路径

首先，UDP 发包终端（P_1）借助 Iperf 发包工具向 VPP 虚拟网络设备（VPP_2）发送具有特殊目的端口号（55555）的 UDP 探测包。然后，VPP_2 作为 INT 源端，接收特殊 UDP 探测包，并依据 CLI 指令配置内容，为探测包封装 INT 头部以及源路由标签栈。实验中设置的源路由路径为 $hop_1 \rightarrow hop_2 \rightarrow hop_3 \rightarrow$控制面终端。此时，3 个 VPP 虚拟网络设备（$VPP_1$、$VPP_2$、$VPP_3$）都将执行 INT 转发设备功能，以采集本设备的 INT 元数据。接着，在控制平面终端接收到 INT 数据包后，将执行 INT 接收终端功能，即解析 INT 头部，获取逐跳节点统计的 INT 元数据。本文依据 Raw Socket 提供的 API，使用 C++语言实现了该解析功能。最后，主机终端（H_1、H_2）也借助 Iperf，向网络中发送特定速率的 UDP 背景流量，这将引起沿路网络转发设备（VPP_1、VPP_2）内部排队处理时延的增加。在对单台 VPP 设备进行转发性能测试时，本文也会直接由 P_1 向 VPP_2 发送普通 UDP 流量。

5.4　实验过程及结果分析

5.4.1　VPP 单节点的转发性能

首先，本文测量单个 VPP 节点在分别转发普通流量和 INT 流量时的性能。实验过程中，P_1 向 VPP_2 分别发送 2 种不同类型的 UDP 包：一种为普通 UDP 包，即 VPP_2 按照正常数据包的处理流程执行转发操作；另一种为前文所述的特殊 UDP 探测包，即 VPP_2 作为 INT 源端，为其封装源路由标签栈以及

INT 头部，并将采集本设备的 INT 元数据嵌入 INT 头部中，最后完成转发操作。为了公平比较，普通 UDP 包以及 INT 数据包的包长度均为 162 B。在不同发送速率下，本文使用 Linux top 命令分别测量了 VPP_2 处理这 2 类数据包的 CPU 利用率，实验结果如图 21 所示。实验结果表明，随着输入包速率的增大，VPP 虚拟网络设备的 CPU 利用率也在上升。此外，由于增加了封装额外包头部信息和采集 INT 元数据等操作，相比转发普通 UDP 流量，处理和转发 INT 流量需要消耗更高的 CPU 算力。经统计，CPU 利用率平均增加的百分比约为 25.6%。值得注意的是，以上增加部分包括 INT 源端生成源路由标签栈和 INT 头部的开销，如果单纯执行 INT 的源路由转发操作，根据在 VPP_3 上进行的测量，这部分增加比例会降低到 20.6%。在实际部署时为了降低 INT 的开销，可以考虑使用基于采样的 INT 头部添加策略进行 INT 开销分摊，避免为每个包添加 INT 头部所带来的较大开销。

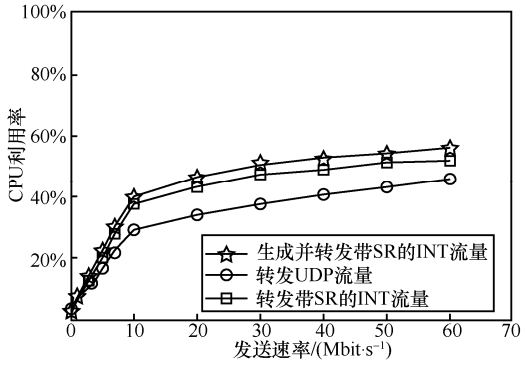

图 21　单台 VPP 虚拟网络设备的 CPU 利用率随流量变化情况

此外，为了横向比较，本文对 VPP、OVS 和 BMv2 的单跳转发性能也进行了测量。转发流量为由 Iperf 产生的 100 Mbit/s 的流量，3 类设备均设置为 L3 最长前缀匹配转发，包含单条转发表项，收发包终端与转发设备直接相连，测量结果如表 1 所示。由表 1 可知，具备矢量包处理能力的 VPP 在 100 Mbit/s 流量冲击下具有最高的吞吐率、最低的时延和最小的丢包率。OVS 在这 3 项指标中均接近 VPP 但仍然存在一定差距。具体而言，OVS 和 VPP 吞吐率较接近，但 OVS 的转发时延达到了 VPP 的 2 倍多，且产生了 2 倍的丢包率。相比之下，BMv2 的转发性能较有限，并在 100 Mbit/s 流量的冲击下产生了 23.4%的丢包，且转发时延达到了惊人的 10.8 ms，这和 BMv2 内部使用软件模块逼真模拟 P4 交换机的硬件流水线的设计实现机制有关。BMv2 为了模拟 P4 交换机的硬件流水线，在自身内部引入了多级的计算和缓存单元，且计算单元完全通过软件指令模拟真实硬件转发行为，这就使数据包处理时延大幅增加。但因为 BMv2 内部包含大量缓存，吞吐率和丢包率降低的数量级相对时延增加来说仍然较有限。

表 1　VPP、BMv2、OVS 单跳转发测量结果

转发设备	吞吐率/(Mbit·s^{-1})	时延/ms	丢包率
VPP	99.9	0.014	0.019%
OVS	98.6	0.038	0.038%
BMv2	75.6	10.8	23.4%

5.4.2　虚拟网络的链路带宽测量

由于是在单个物理机上采用虚拟机组网，每个虚拟机可分配到的 CPU 及内存等资源较有限，这导致了组网规模较小，且无法满足非常高速的网络转发性能需求。本文借助 Iperf，对虚拟网络的实际链路带宽进行了测量。在如图 20 所示的网络拓扑下，本文从 H_1 主机终端发送 UDP 报文到 H_2 主机终端，测量结果显示虚拟网络转发 UDP 流量的链路最大带宽约为 100 Mbit/s。在 100 Mbit/s 的发包速率下，产生了 0.021%的轻微丢包，如果超过这个发包速率，链路丢包率将急剧增大，如图 22 所示。

```
linxchen@vpp:~$
linxchen@vpp:~$ iperf -u -c 10.2.2.1 -b 100M -t 10
------------------------------------------------------------
Client connecting to 10.2.2.1, UDP port 5001
Sending 1470 byte datagrams
UDP buffer size:  208 KByte (default)
------------------------------------------------------------
[  3] local 10.1.1.1 port 57052 connected with 10.2.2.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec   118 MBytes  99.0 Mbits/sec
[  3] Sent 84175 datagrams
[  3] Server Report:
[  3]  0.0-10.0 sec   118 MBytes  99.0 Mbits/sec   0.010 ms   18/84174 (0.021%)
[  3]  0.0-10.0 sec  1 datagrams received out-of-order
linxchen@vpp:~$
```

图 22　虚拟网络中 UDP 流量的最大带宽测量值
（无丢包或轻微丢包）

5.4.3　INT 网络状态探测实验过程描述

在验证 INT 探测包效果的实验中，本文仍然借助 Iperf 发包工具，从 H_1 主机终端向 H_2 主机终端发送不同速率的 UDP 背景流量。同时，通过改变 P_1 向 VPP_2 发送特殊 UDP 探测包的速率，来控制 INT 探测频率。如前所述，目前系统仅支持采集 3 种类型的 INT 元数据。其中，可依据出端口 MAC 地址

来定位采集数据的网络设备，由出入端口时间戳来计算数据包在网络设备中的单跳处理时延，从而初步判断当前网络设备的负载压力。

本文通过改变背景流量速率和INT探测速率这2个变量，分别完成了3组实验，并观察INT数据包采集到的各个虚拟网络设备的单跳处理时延变化情况。为了增强结论的可靠性，本文对每组实验重复进行了10次。考虑到网络的UDP最大（无丢包）带宽约为100 Mbit/s，在每组实验中，在15~25 s时间段内发送速率为90 Mbit/s的UDP背景流量，在30~40 s时间段内发送速率为200 Mbit/s的UDP背景流量（尽可能耗尽链路带宽）。此外，第一组的INT探测速率为100 kbit/s（最大带宽的0.1%），第二组的 INT 探测速率为 1 Mbit/s（最大带宽的1%），第三组的INT探测速率为10 Mbit/s（最大带宽的10%）。每组实验中的INT数据包大小均为162 B。为了方便观察，本文只绘出了拓扑中 hop_2（VPP_3）和 hop_3（VPP_1）两跳的处理时延，3组实验结果的一次记录分别如图23~图25所示。另外，表2还分别列出了10次重复实验下，3组实验在不同背景流速率下统计出的 hop_3 单跳平均处理时延。

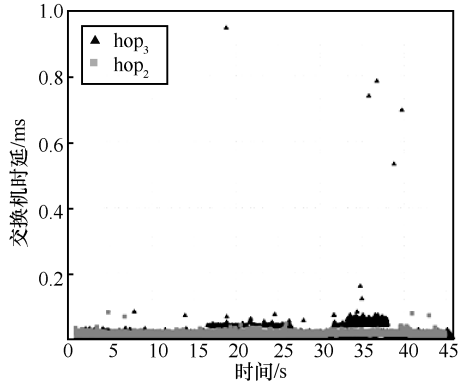

图23　100 kbit/s的INT探测速率下的交换机时延测量结果

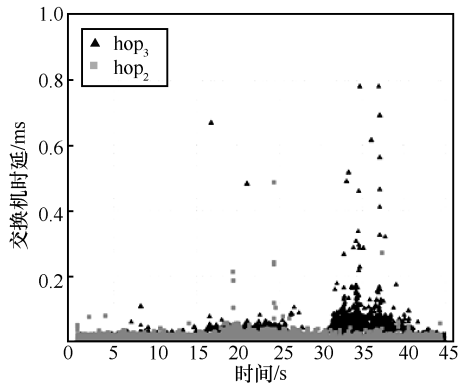

图24　1 Mbit/s的INT探测速率下的交换机时延测量结果

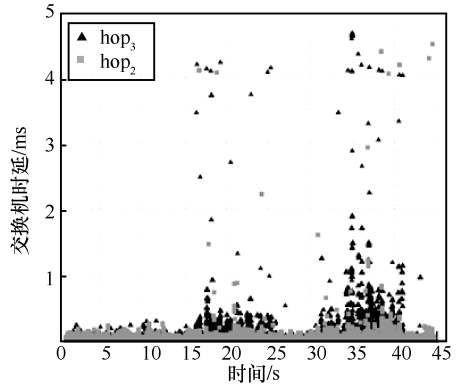

图25　10 Mbit/s的INT探测速率下的交换机时延测量结果

表2　　3组实验的 hop_3 单跳平均处理时延

背景流量	100 kbit/s INT探测速率/μs	1 Mbit/s INT探测速率/μs	10 Mbit/s INT探测速率/μs
无背景流量(3~8 s)	9.7	6.1	15.9
90 Mbit/s背景流量(15~25 s)	13.6	10.8	52.9
200 Mbit/s背景流量(30~40 s)	48.1	41.8	137.9

5.4.4　INT网络状态探测实验结果分析

图23~图25中的每个数据点表示在对应的出端口时间戳上，INT数据包所采集到的单跳网络设备处理时延（通过出端口时间戳减去入端口时间戳得到）。由于端口时间戳的时间精度是微秒级的，因此INT能够提供较高精度的探测结果。在每组实验中，由于背景流量经过的路由路径为 $VPP_1 \rightarrow VPP_2$，而不经过 VPP_3，因此 hop_2（VPP_3）的处理时延总是较低且稳定，基本不受背景流量的影响。在图24和图25中，个别 hop_2 数据点的增高主要是在高负载压力下，运行在同一台资源受限物理机上的各个虚拟机之间互相竞争资源所导致的。而由于背景流量最终会经过 hop_3（VPP_2），当背景流量显著增长，甚至超过图 22 中虚拟网络中测试得到的网络最大带宽时，就会出现明显的拥塞现象。此时背景流量数据包与INT探测包共同在设备队列中经历较长的排队过程，INT探测包能够将该排队时延精准记录并传送到控制面终端上。

从这3组实验 hop_3（VPP_1）的数据点可以看出，在不发送背景流量时，hop_3 的处理时延与 hop_2 基本一致；而在15~25 s、30~40 s时由于引入背景流量，导致 hop_3 的处理时延有所增高。特别地，当背景流量速率为200 Mbit/s时（30~40 s，背景流量速率已经大于网络最大带宽），INT探测包实时采集得到的

处理时延也相应地大幅增加，由此能判断出此时 hop_3 链路已经出现拥塞。另外，如表 2 所示，经过计算，在第二组实验 1 Mbit/s 的 INT 探测速率下，由于背景流量的变化（无背景流量、90 Mbit/s、200 Mbit/s），采集到的 INT 数据包所携带的 hop_3 单跳平均处理时延分别为 6.1 μs、10.8 μs、41.8 μs。由此也可以判断出，当背景流量为 200 Mbit/s 时，hop_3 链路出现了较严重的交换机内部队列排队现象。

此外，通过对比各组实验结果，本文还观察到不同的 INT 探测速率对探测效果的影响。例如，通过对比 100 kbit/s 和 1 Mbit/s 的实验结果（图 23 和图 24）可以看出，更高的 INT 探测速率在一定程度上有利于提高探测结果的精度。但是，INT 探测速率越高，其所占用的网络带宽也越大，且 INT 探测包在设备队列中的排队行为也将大大提升背景流量的转发时延。如表 2 所示，在 10 Mbit/s 的 INT 探测速率下，在无背景流量、90 Mbit/s 背景流量、200 Mbit/s 背景流量时，计算得到的 hop_3 单跳平均处理时延分别为 15.9 μs、52.9 μs、137 μs，均大大高于 1 Mbit/s 的 INT 探测速率下测得的 6.1 μs、10.8 μs、41.8 μs。这也说明如果 INT 探测速率太大，INT 数据包本身在网络设备中的处理开销就会大幅提高，使转发处理时延大大增加，反而不利于提升探测结果的精确度。因此，在实际部署中，需要依据具体的探测精度需求和探测开销评估，设计合适的 INT 探测速率。

5.5　与其他网络测量方法的比较分析

综上，本节对本文提出的方法与其他网络测量方法进行比较分析。对于非 INT 方法来说，存在测量效率和测量功能的缺陷。例如基于 Poll 模式的 SNMP 控制平面对数据平面的数据采集时延往往为 0.4~21.1 ms[17]。而本文方案的探测时间间隔在目前的机器配置下能够轻松达到 0.13 ms。对于 NetFlow、sFlow、IPFIX 等方法[18-21]，本文方案可以探测到交换机端口的拥塞和排队信息，而不仅仅局限于包计数或流量计数等宏观信息。相较于 Pingmesh[22]，本文方案除了能够监测路径拥塞状态，还能监测路径上逐跳交换机的拥塞状态。对于基于协议无关转发架构实现的 P4 硬件交换机[9,10,33]来说，本文方案更好地支持虚拟网络设备，而不依赖于协议无关转发架构本身。对于 BMv2 软件交换机[30]来说，表 1 显示 VPP 的转发性能远高于 BMv2，这也意味着本文方案可以在实际生产环境部署，而不是仅仅局限于网络功能验证的场景。

6　结束语

虚拟网络设备在数据中心内被大量使用。在虚拟网络设备上增加低时延、高精度的带内网络遥测功能有助于在大规模数据中心内开展实时流量可视化、故障快速定位、细粒度流量工程等网络优化任务。作为 P4 的重要应用之一，带内网络遥测依赖于协议无关转发架构，目前仅有少量可编程硬件交换机支持该能力，大多数虚拟网络设备均无法运行 P4 程序，以提供带内网络遥测的直接支持。本文首先基于开源虚拟网络设备 VPP，设计了带内网络遥测的实现方案；然后基于带内网络遥测协议规范定义了相关数据包的报文格式，基于 VPP 的数据包处理框架构造了数据平面的流水线处理模块，并在此基础上实现了一种带内全网遥测机制的扩展案例；最后基于虚拟机组网方式，搭建网络拓扑并进行了网络性能测试，通过实验展示了不同背景流量速率和不同探测频率下的遥测性能数据。

参考文献：

[1] BARI M F, BOUTABA R, ESTEVES R, et al. Data center network virtualization: a survey[J]. IEEE Communications Surveys & Tutorials, 2012, 15(2): 909-928.

[2] DANIELS J. Server virtualization architecture and implementation[J]. XRDS: Crossroads, The ACM Magazine for Students, 2009, 16(1): 8-12.

[3] NELSON M, LIM B H, HUTCHINS G. Fast transparent migration for virtual machines[C]//USENIX Annual Technical Conference, General Track. Berkeley: USENIX Association, 2005: 391-394.

[4] EGI N, GREENHALGH A, HANDLEY M, et al. Towards high performance virtual routers on commodity hardware[C]//Proceedings of the 2008 ACM CoNEXT Conference. New York: ACM Press, 2008: 1-12.

[5] PFAFF B, PETTIT J, KOPONEN T, et al. The design and implementation of open vswitch[C]//12th USENIX Symposium on Networked Systems Design and Implementation. Berkeley: USENIX Association, 2015: 117-130.

[6] BARACH D, LINGUAGLOSSA L, MARION D, et al. High-speed software data plane via vectorized packet processing[J]. IEEE Communications Magazine, 2018, 56(12): 97-103.

[7] JIA C, PAN T, BIAN Z, et al. Rapid detection and localization of gray failures in data centers via in-band network telemetry[C]//NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium. Piscataway: IEEE Press, 2020: 1-9.

[8] 刘争争, 毕军, 周禹, 等. 基于 P4 的主动网络遥测机制[J]. 通信学报, 2018, 39(Z1): 162-169.
LIU Z Z, BI J, ZHOU Y, et al. Paradigm for proactive telemetry based on P4[J]. Journal on Communications, 2018, 39(Z1): 162-169.

[9] BOSSHART P, GIBB G, KIM H S, et al. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN[J].

ACM SIGCOMM Computer Communication Review, 2013, 43(4): 99-110.
[10] BOSSHART P, DALY D, GIBB G, et al. P4: programming protocol-independent packet processors[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(3): 87-95.
[11] KIM C, SIVARAMAN A, KATTA N, et al. In-band network telemetry via programmable dataplanes[C]//Proceedings of the Symposium on SDN Research. New York: ACM Press, 2015: 1-2.
[12] SONG E, PAN T, JIA C, et al. INT-filter: mitigating data collection overhead for high-resolution in-band network telemetry[C]//GLOBECOM 2020-2020 IEEE Global Communications Conference. Piscataway: IEEE Press, 2020: 1-6.
[13] CASE J D, FEDOR M, SCHOFFSTALL M L, et al. RFC1157: simple network management protocol (SNMP)[R]. IETF, 1990-05.
[14] BREITGAND D, RAZ D, SHAVITT Y. SNMP GetPrev: an efficient way to browse large MIB tables[J]. IEEE Journal on Selected Areas in Communications, 2002, 20(4): 656-667.
[15] CHUN J K, CHO K Y, CHO S H, et al. Network management based on PC communication platform with SNMP and mobile agents[C]//Proceedings of 22nd International Conference on Distributed Computing Systems Workshops. Piscataway: IEEE Press, 2002: 222-227.
[16] ANDREY L, FESTOR O, LAHMADI A, et al. Survey of SNMP performance analysis studies[J]. International Journal of Network Management, 2009, 19(6): 527-548.
[17] PRAS A, DREVERS T, VAN DE MEENT R, et al. Comparing the performance of SNMP and Web services-based management[J]. IEEE Transactions on Network and Service Management, 2004, 1(2): 72-82.
[18] ESTAN C, KEYS K, MOORE D, et al. Building a better NetFlow[J]. ACM SIGCOMM Computer Communication Review, 2004, 34(4): 245-256.
[19] PHAAL P, PANCHEN S, MCKEE N. RFC3176: InMon corporation's sFlow: a method for monitoring traffic in switched and routed networks[R]. IETF, 2001-09.
[20] QUITTEK J, ZSEBY T, CLAISE B, et al. Requirements for IP flow information export (IPFIX)[R]. RFC 3917 (informational), IETF, 2004-10.
[21] 马云龙, 张千里, 王继龙. 基于 IPFIX 的网络流量日志系统[J]. 通信学报, 2013, 34(Z2): 5-8.
MA Y L, ZHANG Q L, WANG J L. Network traffic analysis system based on IPFIX[J]. Journal on Communications, 2013, 34(Z2): 5-8.
[22] GUO C, YUAN L, XIANG D, et al. Pingmesh: a large-scale system for data center network latency measurement and analysis[C]//Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication. New York: ACM Press, 2015: 139-152.
[23] LI Y, MIAO R, LIU H H, et al. HPCC: high precision congestion control[C]//Proceedings of the ACM Special Interest Group on Data Communication. New York: ACM Press, 2019: 44-58.
[24] KATTA N, HIRA M, KIM C, et al. Hula: scalable load balancing using programmable data planes[C]//Proceedings of the Symposium on SDN Research. Piscataway: IEEE Press, 2016: 1-12.
[25] JIN X, LI X, ZHANG H, et al. Netcache: balancing key-value stores with fast in-network caching[C]//Proceedings of the 26th Symposium on Operating Systems Principles. Piscataway: IEEE Press, 2017: 121-136.
[26] MIAO R, ZENG H, KIM C, et al. Silkroad: making stateful layer-4 load balancing fast and cheap using switching ASICs[C]//Proceedings of the Conference of the ACM Special Interest Group on Data Communication. New York: ACM Press, 2017: 15-28.
[27] The P4.org Applications Working Group. In-band network telemetry (int) dataplane specification (version 1.0)[R]. (2018-04)[2020-08-04].
[28] Huawei. World's first iFIT pilot on the 5G transport network successfully completed by beijing Unicom and Huawei[R]. (2019-06-28) [2020-08-04].
[29] Broadcom. Broadcom trident 4 delivers disruptive economics for enterprise data center and campus networks[R]. (2019-06-11) [2020-08-04].
[30] PANTELOPOULOS A. Towards accurate simulations of programmable dataplanes[D]. Zurich: ETH Zurich, 2017.
[31] SUNSHINE C A. Source routing in computer networks[J]. ACM SIGCOMM Computer Communication Review, 1977, 7(1): 29-33.
[32] PAN T, SONG E, BIAN Z, et al. INT-path: towards optimal path planning for in-band network-wide telemetry[C]//IEEE INFOCOM 2019-IEEE Conference on Computer Communications. Piscataway: IEEE Press, 2019: 487-495.
[33] 朱祖勍, 孔嘉伟, 牛彬, 等. 基于深度学习的面向 IP-over-EON 的可编程跨层网络业务性能感知系统[J]. 通信学报, 2019, 40(11): 171-179.
ZHU Z Q, KONG J W, NIU B, et al. DL-assisted programmable multilayer network application awareness system for IP-over-EON[J]. Journal on Communications, 2019, 40(11): 171-179.

[作者简介]

潘恬（1987− ），男，江苏常熟人，博士，北京邮电大学副教授、硕士生导师，主要研究方向为高速可编程网络、数据中心网络、低轨卫星网络等。

林兴晨（1995− ），男，福建宁德人，北京邮电大学硕士生，主要研究方向为高速可编程网络、数据中心网络等。

张娇（1986− ），女，河北保定人，博士，北京邮电大学副教授、博士生导师，主要研究方向为数据中心网络、传输控制协议、网络功能虚拟化等。

黄韬（1980− ），男，重庆人，博士，北京邮电大学教授、博士生导师，主要研究方向为未来网络体系架构、软件定义网络、信息中心网络等。

刘韵洁（1943− ），男，山东烟台人，中国工程院院士，主要研究方向为未来网络体系架构、网络操作系统、网络融合与演进、软件定义网络等。