研究与开发

面向多模态网络的SONiC网元控制通道容器设计

涂化清[1]，方徐鑫[2]，朱俊[1]，邹涛[1]，李传煌[2]，隆克平[1,3]
（1. 之江实验室，浙江 杭州 311100；
2. 浙江工商大学，浙江 杭州 310018；
3. 北京科技大学，北京 100083）

摘　要：为解决SONiC（software for open networking in the cloud）交换机操作系统对多模态网络（polymorphic network，PINet）中模态适配及模态管控问题，提出了一个基于P4Runtime的SONiC网元控制通道容器p4runtime-pins，使多模态网元设备可以支持多种网络模态流表的配置。p4runtime-pins容器通过gRPC服务模块实现与控制器的连接，使用邻近网元发现算法实现控制器对链路的发现。设计了网元端口更新算法解决了网元设备在实际应用环境中存在的端口变更问题。同时，针对SONiC网元交换机中硬件转发处理单元存在的流表支持性差异问题，设计了内部流表转存和gRPC网元代理功能，实现了不同网络模态流表的部署。实验结果表明，p4runtime-pins容器资源消耗低，仅占用了1.70%的CPU资源和0.45%的内存资源。同时，部署p4runtime-pins容器的SONiC网元设备能够准确地接收并配置控制器下发的流表规则，流表配置延迟仅为0.027~0.037 s。

关键词：多模态网络；白盒交换机；交换机操作系统；网元控制通道
中图分类号：TN393
文献标志码：A
doi: 10.11959/j.issn.1000−0801.2025026

Desigh of SONiC network element control channel container for plymorphic network

TU Huaqing[1], FANG Xuxin[2], ZHU Jun[1], ZOU Tao[1], LI Chuanhuang[2], LONG Keping[1,3]
1. Zhijiang Laboratory, Hangzhou 311100, China
2. Zhejiang Gongshang University, Hangzhou 310018, China
3. University of Science & Technology Beijing, Beijing 100083, China

Abstract: To address the modal adaptation and modal control issues in polymorphic network (PINet) for software for open networking in the cloud (SONiC) switch operating system, a SONiC network element control channel container called p4runtime-pins based on P4Runtime was proposed. It enabled multi-modal network element devices to support

收稿日期：2024−11−11；**修回日期：**2025−01−10
通信作者：李传煌，chuanhuang_li@zjgsu.edu.cn
基金项目：国家自然科学基金资助项目（No. U22A2005）；国家重点研发计划项目（No. 2022YFB2901503）；中国博士后科学基金资助项目（No. 2024M75986）
Foundation Items: The National Natural Science Foundation of China (No. U22A2005), The National Key Research and Development Program of China (No. 2022YFB2901503), The China Postdoctoral Science Foundation (No. 2024M75986)

configuration of various network modal flow tables. In the p4runtime-pins container, a connection with the controller through the gRPC service module was established, a neighboring network element discovery algorithm for controller discovery of links was utilized, and a network element port update algorithm to resolve the issue of port changes in practical application environments was designed. Furthermore, to address the issue of poor support for flow table variations in the hardware forwarding processing unit of SONiC network element switches, the internal flow table dumping and gRPC network element proxy functionality was introduced in p4runtime-pins to facilitate the deployment of different network modal flow tables. Experimental results demonstrate that the p4runtime-pins container has low resource consumption, occupying only 1.70% of CPU and 0.45% of memory. Moreover, SONiC network element devices deployed with the p4runtime-pins container can accurately receive and configure flow table rules issued by the controller, with an only 0.027~0.037 s flow table configuration delay.

Key words: PINet, white box switch, switch operating system, network element control channel

0　引言

随着信息通信网络技术和网络规模的不断发展和扩大，新兴的网络业务不断涌现。而现有的网络架构无法满足用户日益增长的高质量用网需求。面对上述问题，我国邬江兴院士研究团队提出了一种网络结构全维度可定义的多模态网络（polymorphic network，PINet）[1-2]。多模态网络[3]需要支持不同协议数据包的同时处理，实现多种模态数据的接收与转发。而传统交换机由于交换机厂商的定制，整个交换机设备高度耦合，能够实现的功能有限。白盒交换机则突破了传统交换机设备的软硬件捆绑一体化设计形式，将底层硬件和上层软件操作系统进行了解耦，支持控制平面协议软件可编程及数据平面转发芯片硬件可编程[4]。将SONiC[5]作为开源的白盒交换机操作系统，突破了软硬件捆绑一体化设计，为多模态网络数据平面搭建提供了有力的支持。

然而，当SONiC操作系统应用于多模态网络环境时，面临着显著的技术挑战，尤其是有效地支持并整合多种异构的标识体系和网络架构模型。这些多模态特性要求SONiC不仅需具备高度的灵活性和可扩展性，还需解决标准化接口的问题，以便能够与不同的网络设备和应用程序兼容。SONiC采用Docker容器形式的微服务架构，能够通过交换机抽象接口（switch abstraction interface，SAI）[6]运行在不同的ASIC（application-specific integrated circuit）平台上。但原生的SONiC操作系统仅支持传统的IPv4和IPv6网络的转发控制，不支持其他网络模态（如工业标识网络、移动优先网络等）的表项配置，从而未能适配多模态网络。

目前，虽然业界对多模态网络及可编程数据平面的研究逐步展开，但并未解决上述问题。例如，董永吉等[7]提出了面向多模态网络需求的存转算一体化数据平面共性网络平台，为多模态网络中存储、转发和计算融合的需求做出了贡献。文献[8]设计了基于开源的开放网络操作系统（open network operating system，ONOS）[9]并面向BMv2（behavioral model version 2）交换机的多模态网络模态管控系统。同时，国外NG-SDN项目[10]基于开源控制器ONOS以及Stratum操作系统[11]构建可编程的网络控制体系结构。但是，Stratum交换机操作系统主要适用于高度软件定义网络（software defined network，SDN）集成、硬件独立性和高性能的解决方案和场景。相比之下，SONiC交换机操作系统更适用于大规模生产环境、模块化设计和高度硬件兼容性应用的场景。然而，国内外尚无工作对SONiC操作系统进行改进以适配多模态网络。

因此，为解决SONiC对多模态网络中模态管

控适配问题，本文设计了面向SONiC交换机操作系统的多模态网元控制通道容器p4runtime-pins，该容器使用P4Runtime协议，与开源可编程的SDN控制器相结合，通过gRPC与控制器交互的方式，实现基本链路功能以及自定义控制面流表的下发与转存。然而，在实现SONiC网元设备控制通道容器过程中面临着两个挑战。第一，网元设备在不同场景中端口的变更会影响控制器的拓扑显示以及控制信息的传输。第二，硬件转发处理单元带来的对流表支持性的差异问题，需要能够对不同处理单元进行适配。针对上述挑战，本文通过添加端口变更算法解决端口场景化变更问题，同时设计网元代理模块适配不同转发处理单元的差异。本文设计的容器结合开源SDN控制器和可编程交换机来实现多模态网络数据包的转发控制，为多模态网络网元设备管控及扩展提供了途径。

1 研究背景与挑战

1.1 SONiC网元多模态控制通道

多模态网络旨在突破传统相对单一的网络协议的限制，融合多种模态标识，以适应并推动新型网络的发展需求。虽然当前存在多种寻址路由技术及体系，但这些标识网络模态均基于各自的技术体系和支撑环境。因此，需要对各标识网络模态进行不同程度的适配来实现统一的支撑。多模态网络通过以太网MAC层实现这种统一承载。在MAC层，以太网帧类型字段的0x0800和0x0809~0x081B分别表示各类模态，以太网帧模态承载示意图如图1所示，各模态的具体报文格式在网络层呈现。

SONiC采用SAI作为南向接口，开发者可以借助交换芯片厂商的软件开发工具包（software development kit，SDK）快速地适配多种硬件平台以实现多模态网络开发。SONiC操作系统架构如图2所示，用户层包括一系列轻量化容器组件；内核层配备各类驱动用于与上层容器及与底层硬件交互；交换机硬件层安装芯片、电源、风扇等各类硬件。其中，用户层根据SAI并通过ASIC驱动与交换机中的ASIC芯片进行通信。由于在多模态网络中，不同网络模态采用不同的协议数据包格式，因此不同模态的管控流表表项也存在

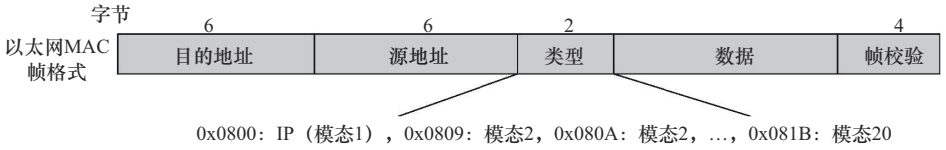


图1 以太网帧模态承载示意图

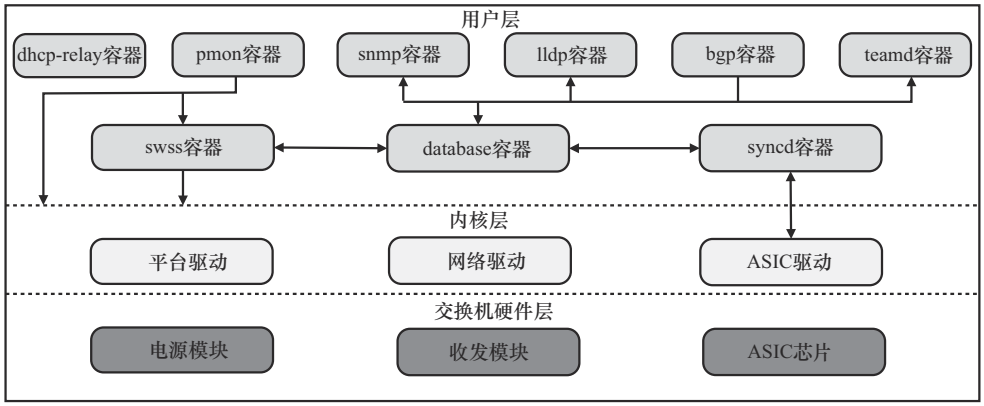


图2 SONiC操作系统架构

差异。虽然使用P4[12]协议能实现多种模态数据包的处理，然而当前SONiC默认只支持普通IPv4和IPv6网络协议，不适用于多模态网络的多模态场景。因此，为了能够在SONiC上实现对多模态网络场景下的数据包转发控制，需要通过可编程和可扩展的能力，对SONiC进行多模态表项适配。

1.2　SONiC网元多模态控制通道面临的挑战

为解决SONiC对多模态网络模态管控适配的问题，本文设计了p4runtime-pins多模态网元控制通道容器，实现SONiC网元设备的多模态流表下发与流量管控。在设计p4runtime-pins容器时需解决以下两个问题。

（1）如何解决网元设备场景化端口变更问题?

在实际网元交换机运行场景中，设备端口会产生变化。当网络拓扑发生变化时，如添加、删除或移动电缆；当端口出现故障时，如端口损坏、故障修复；交换机端口的物理连接状态会相应改变。在使用各类协议或者技术，如链路聚合和冗余技术时，端口的状态可能会根据链路聚合组的配置和冗余路径的可用性而动态变化。由于控制器需要实时反映网络中的拓扑以及设备连接状态情况，因此，在p4runtime-pins容器运行时，如何实现对SONiC网元设备端口变更状态的实时更新是个重要问题。

（2）如何适配交换机转发处理单元的差异?

在多模态网元设备中，多模态数据流量的转发可能通过底层的ASIC芯片，也可能通过在ASIC的基础上拓展的其他转发处理单元，如FPGA[13]。不同的多模态网元设备中对P4流表的支持可能不同。因此，在将P4流表下发到这些设备时，会遇到兼容性问题。对于支持P4流表下发管理的转发处理单元，p4runtime-pins可将流表信息直接部署；对于不直接支持P4流表的转发处理单元，需通过p4runtime-pins容器完成流表对管理程序输入的格式转换。因此，如何使容器能够适配转发处理单元的差异，是控制通道容器设计过程中的另一个关键问题。

2　面向多模态网络的SONiC网元容器设计

2.1　控制通道容器架构

控制通道容器系统架构如图3所示。整个控制通道由SDN控制器、p4runtime-pins容器、配置载入服务组件和ASIC可编程芯片构成。其中，p4runtime-pins容器内部包括4个模块，分别为gRPC服务端、网元发现模块、流表转存模块和网元代理模块。SONiC操作系统以及ASIC可编程芯片作为软硬件搭载于白盒网元交换机中。gRPC服务端和网元发现模块用于与控制器连接和交互，提供链路设备信息。流表转存模块用于转存来自控制器的流表信息至本地配置文件。网元代理模块为网元设备内部服务场景提供控制信息转发的功能。在下发控制操作时，控制器能够通过容器与网元交换机设备建立连接，同时容器能够将控制器下发的流表规则接收转存为相应的配置文件，并通过配置载入服务组件与database容器和syncd容器联动将表项规则下发至具体的ASIC芯片/转发处理单元。

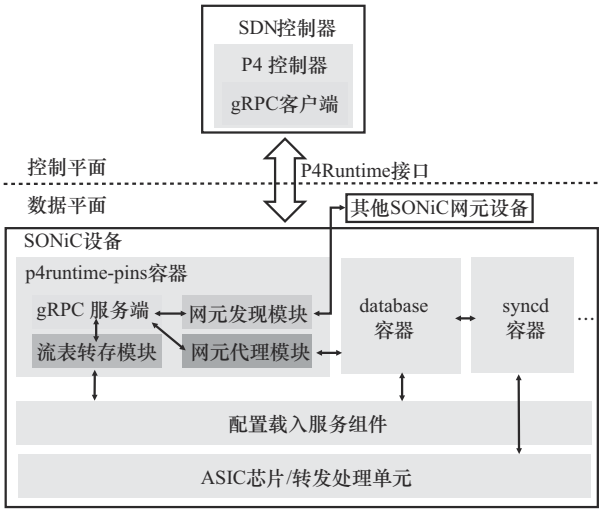


图3　控制通道容器系统架构

多模态网元管控流程如图4所示，为部署p4runtime-pins控制通道容器的SONiC网元在多模态网络中的管控过程[14]。控制器在与网元建立连接后，控制器通过与容器中的网元发现模块交互得到网络整体的拓扑结构。图4中，以IP模态为例，展示了控制器通过p4runtime-pins容器中流表接收转存或网元代理解析下发流表后，IP模态数据包转发状态的变更。

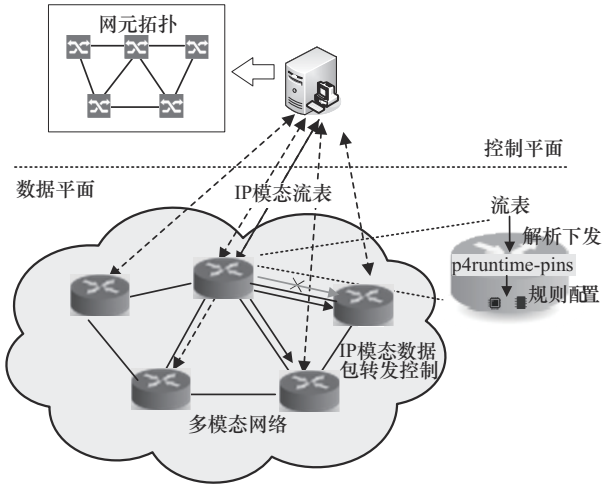


图4　多模态网元管控流程

2.2　gRPC服务端

gRPC服务端是整个容器的基础，控制平面控制器和网元设备的基本通信通过该模块实现。gRPC服务通过P4Runtime协议[15]实现，允许控制平面与数据平面分离，使控制器可以与不同厂商的交换机和路由器进行交互，而不依赖特定的硬件设备。容器内gRPC服务通过P4Runtime提供的协议缓冲区（protocol buffers，ProtoBuf）文件p4runtime.proto中预定义的gRPC服务方法来实现服务端的建立。

gRPC服务端模块流程如图5所示。在服务开启前配置对应的服务IP地址及端口，该端口需要保持与SDN控制器中的gRPC服务开启的端口一致。随后对p4info的表项及动作信息进行读取，此处p4info文件为控制器中通过P4编译的多模态数据包及表项动作定义的P4文件所得。在读取完成后通过实例开启gRPC服务，在服务与客户端控制器建立连接后通过执行获取/设置转发管道配置方法读取并配置转发管道信息。随后gRPC服务一直处于监听服务状态。

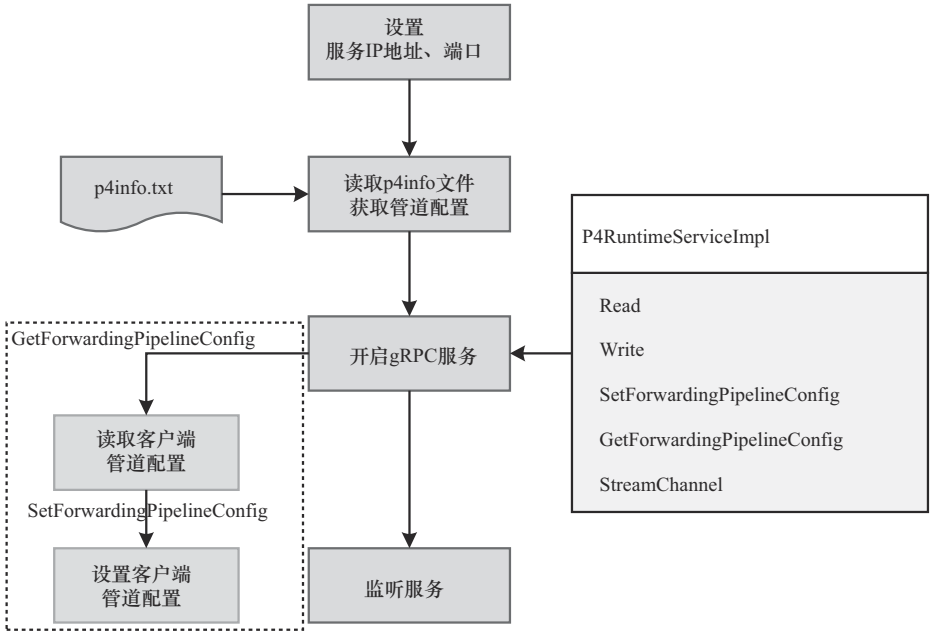


图5　gRPC服务端模块流程

2.3　邻近网元发现

服务端的构建能够完成控制器与容器的gRPC通信，但是作为网元设备的控制器，需要在网络中起到发现、识别和跟踪网络中其他设备的作用。网络中网元状态的捕获是一个动态的问题，网元设备的状态时刻发生变化。

邻近网元发现功能如图6所示，控制器需要获取网元交换机的连接状态，即图6中实线部分的连接。同时，不同设备的连接状态需要实时反馈给控制器。如果网络中有网元设备断开连接，则控制器需及时发现。当网元设备启动时，控制器同样需要捕获其状态，实现网元设备更新。首先，本节描述邻近网元发现实现过程，然后，通过网元端口队列更新算法解决第1.2节中提到的端口变更问题。

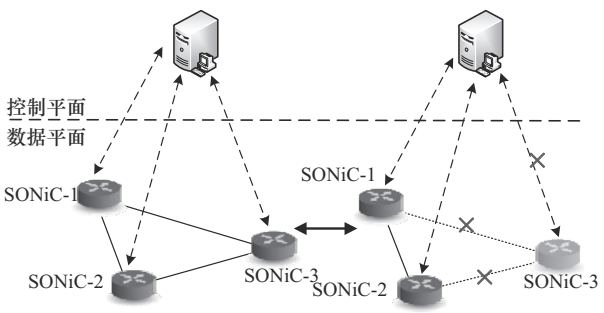


图6　邻近网元发现功能

在数据平面中，网元设备通过数据网口与不同的设备进行数据传输。同时每个网元设备又是一个gRPC服务端，与SDN控制器相连。在控制器端，有集成链路层发现协议（link layer discovery protocol，LLDP）功能的LLDP Link Provider程序，该程序通过监听LLDP数据包向控制器核心提供网络中的链路状态。因此，本文基于控制器的LLDP Link Provider程序，在p4runtime-pins容器中设计了pcap程序来实现控制器对网元设备的发现功能。

LLDP包转发过程如图7所示，在控制器层面，控制器在与网元设备建立gRPC服务时，会获取连接设备的端口开启状态。在得到已开启的端口号后，控制器中的LLDP Link Provider程序通过得到的开放端口号，将其存入具备LLDP功能的数据包中，并向建立连接的设备发送封装后的LLDP数据包。对于每个部署容器的网元设备，当容器随着设备启动时，容器除了开启gRPC服务，同时将扫描当前设备中的所有开启端口，并通过pcap为每一个已开启的端口挂起一个线程。该pcap线程设有过滤器，主要作用为过滤从对应端口输入的数据包，并从中捕获LLDP数据包。

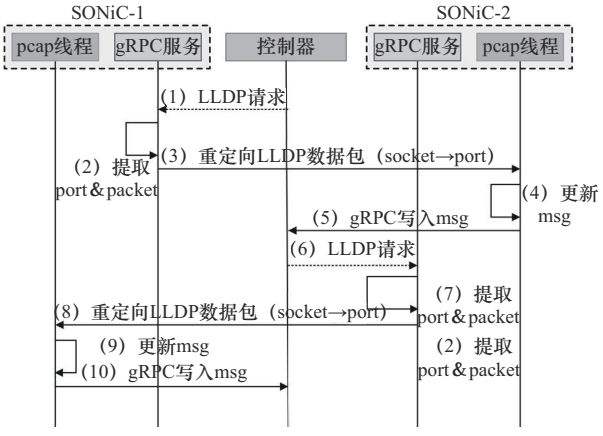


图7　LLDP包转发过程

通过创建pcap端口抓包线程，能够实现LLDP数据的转发与回传，完成网元状态更新。但是，网元设备的端口连接状态有可能发生变化，当网络中发生旧网元断开连接，或新网元的添加时，与之相连的网元端口会发生变化，此时如果不能更新端口状态并设置相应的pcap端口抓包线程，则会导致变更的网元无法连接，从而在控制器端无法成功识别更新。

针对网元设备端口变更问题，本文通过监控线程并应用网元设备网络端口更新队列动态监控网元设备情况，网络端口队列更新算法如算法1所示，在线程开始时，程序将读取当前网元设备所有的网口信息，并创建对应网口的pcap线程一

并缓存至网口设备队列中。在初始化原始网口信息后，端口监控线程开始运行，在线程执行时，程序将会监测网元设备的端口状态，当设备的网络端口发生变更时，对缓存队列数据进行更新。

算法1 网络端口队列更新算法

输入 DevVector（交换机网口队列），ThreadVector（网口对应的pcap线程队列），DevDifferFlags（网口开启状态列表）

输出 更新后的DevVector，ThreadVector，DevDifferFlags

初始化临时设备变量tmpDevs并对其升序；

初始化标志全局标志Devchange为false；

for 每台设备在tmpDevs中 **do**

if 设备不在DevVector中 **then**

设置标志队列DevDifferFlags对应项false；

全局Devchange置为true；

end if

end for

if Devchange与设备未变更 **then**

return DevVector，ThreadVector，DevDifferFlags

end if

for 每个队列中的端口 **do**

记当前端口、变更端口分别为d_{cur},d_{new}，新旧状态为f_{cur}、f_{new}，新线程为n_{pcap}；

更新当前端口$d_{cur}\leftarrow(d_{cur}\wedge d_{new})\vee(\neg d_{cur}\wedge d_{new})$;

更新当前状态$f_{cur}\leftarrow(f_{cur}\wedge f_{new})\vee(\neg f_{cur}\wedge f_{new})$;

ThreadVector$\leftarrow n_{pcap}$；

end for

return DevVector，ThreadVector，DevDifferFlags

2.4 流表接收转存

网元设备流量的管控通过流表进行操作。控制器经网元发现后，获得拓扑信息，同时控制器能够通过类似REST API（representational state transfer application program interface）等接口对指定网元设备来下发流表控制信息。但是，REST API中提供的接口通过gRPC客户端调用写入方法以请求的方式下发流表规则至服务端。因此，在服务端需要对控制端与流表下发请求解析提取，将对应控制信息以配置读取服务组件能够读取的指定格式转存至配置文件。

流表表项配置格式如图8所示。针对流表转存的格式，为了方便存取并提高可读性，本文将芯片接口字段、多模态表名、匹配类型、匹配字段和具体的匹配信息组合并根据图8格式转存。

```
p4 = bfrt.ipv4_ipv6_pwl_8l80_asic. pipe
p4.IngressPipeImpl.ipv4_exact.add_with_ipv4_drop(dst_addr=0x0b0b6f0b)
p4.IngressPipeImpl.ipv4_exact.del_with_ipv4_drop(dst_addr=0x0b0b6f0b)
```

图8 流表表项配置格式

此外，网元设备服务组件通过定位和读取转存文件，并调用SDK实现搭载SONiC操作系统的网元设备自定义流表规则的部署。在流表请求解析提取至配置文件过程中，由于流表请求由控制器下发，请求量由网络管理人员依据对网络设备调整情况下发，同一时刻可能存在同一流表规则的增删等冗余操作，这些操作不仅会增加资源的浪费，而且会影响流表部署速度。本文使用流表缓存队列来解决上述问题，流表缓存队列更新算法如算法2所示。首先初始化提取output中的表名（table）、类型（type）、匹配内容（match）等信息，在新的流表请求解析至output后，通过对比当前队列内流表与output的差异情况，分别对其进行操作。若output中的流表配置规则在队列中存在，则不做入队操作。若解析后的流表配置规则在队列中不存在，则依次对比其表名、类型以及匹配内容等字段信息。table不存在表示新表，则直接入队；若table存在，则比较type类型，若type类型不同，表示同一张表的增删操

作，则删除队列中的相应指令项，对当前的output不做入队操作。通过添加流表缓存队列，能够实现对解析后流表的统一管理以及在避免资源浪费的同时提高流表的解析部署速度。

算法2　流表缓存队列更新算法

输入 output（解析输出流表规则），flowsCasheQueue（流表缓存队列）

输出 更新后的flowsCasheQueue

初始化临时参数table，type，match；

提取参数信息table，type，match←output；

for 每个在flowsCasheQueue中的表项**do**

　if output在flowsCasheQueue中 **then**

　　continue“；”

　else

　　if table与表项相同 **then**

　　　if type与表项不同，match相同 **then**

　　　　flowsCasheQueue中删除表项“；”

　　　else if match与表项不同**then**

　　　添加output至flowsCasheQueue“；”

　　　end if

　　else

　　　添加output至flowsCasheQueue“；”

　　end if

　end if

end for

return flowsCasheQueue

2.5　网元代理功能

在多模态网络中，网元有核心级与用户级之分，不同设备的转发处理单元存在差异。对于ASIC芯片和不同转发处理单元，对P4流表的支持程度有差异，对于不支持P4流表格式的芯片和转发处理单元，流表接收转存模块通过将从控制器中接收的模块进行转存并格式化为芯片或者其他转发处理单元能够直接管理的格式。对于支持P4流表格式的芯片和转发处理单元，则通过网元代理模块内部转发至对应的转发单元中。因此，本节通过设计网元代理模块来解决转发硬件差异场景下的控制信息转发带来的挑战。

为了实现代理功能，本文通过P4Runtime中提供的gRPC通信存根构建代理gRPC通道。代理启动过程如图9所示，在第2.2节的gRPC服务端开启的过程中，通过添加的代理配置文件，以及代理服务类，新增代理服务启动流程。其中，在配置文件中，通过定义target_flow配置字段，能够过滤指定流表控制规则，提高网元代理的可操作性。同时，容器主程序通过扫描指定路径下的代理配置文件解析并提取代理信息，以创建gRPC代理客户端与目标IP地址的网元设备的gRPC服务端的通信。

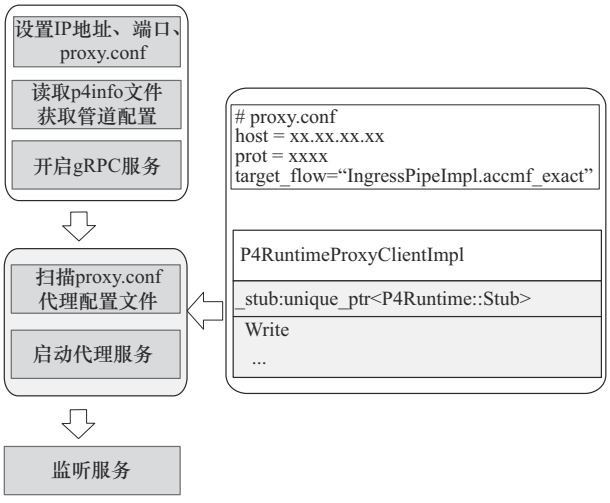


图9　代理启动过程

代理转发过程如图10所示，网元代理通道开启后，具有代理功能的网元设备可认为是一个代理客户端。来自控制器的流表POST/DELETE请求到达网元代理后，根据配置文件中的IP地址和端口号转发至指定的网元设备。具体的转发流程如下，以控制器通过POST下发流表为例。控制器下发流表时，控制器处的gRPC客户端会以JSON流表格式通过POST将请求发送至指定的设备，若代理未配置，则请求将根据第2.4节的流表接收转存的流程解析并提取请求中的流表内容，正常转存部署。代理存在时，程序根据tar-

get_flow过滤所需的流表名，程序将接收到的请求通过代理通道转发至配置文件中的目的IP地址网元设备。此时，在目的网元设备中的gRPC服务器会收到经转发的请求，由于目的网元设备并未配置代理文件，设备容器内只包含一个gRPC服务，程序将正常根据gRPC请求处理方法处理流请求对请求中的流表解析并转存。

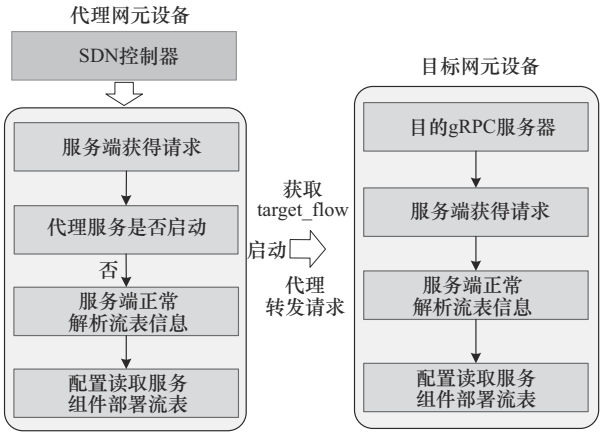


图10　代理转发过程

3　测试与评估

本节中，通过对实体交换机进行全面的场景测试来评估控制通道容器p4runtime-pins的效果和性能，实验结果表明：

（1）在网元端口变更场景中，配置p4runtime-pins的网元设备能实现端口状态的更新；

（2）配置p4runtime-pins容器的SONiC网元在转发处理单元存在差异的情况下，仍能实现多模态网络数据包的正确转发控制；

（3）与SONiC操作系统中原有容器进行性能对比，p4runtime-pins容器CPU占用率为1.70%，内存占用率为0.45%，较其他容器的CPU和内存占用率表现更稳定；

（4）流表下发后流量转换性能方面，测试设备中流表下发后流量的变更切换延迟为0.027~0.037 s，流量调整时间稳定。

3.1　实验设置

本文使用C++语言，在P4Runtime协议提供的gRPC接口上实现面向可编程交换机的SONiC操作系统的控制通道容器p4runtime-pins，使用搭载3.2 GHz Intel Core i7 8700 CPU、16 GB内存的服务器作为ONOS控制器终端，该服务器运行Ubuntu 20.04操作系统。测试环境为搭载盛科CTC8180（TsingMa.MX）交换芯片的核心级多模态网元PINE-B1000-P48C8A。该芯片支持高达2.4 Tbit/s的交换容量，还具备从10 GB到100 GB等多种速率端口转发的能力，充分满足了不同网络场景下的高性能需求。网元通过开放网络安装环境（open network install environment，ONIE）安装SONiC操作系统。

3.2　网元端口变更实验

为了验证第1.2节中提到的在实际场景中网元交换机设备端口变更所涉及的问题。本文第一组实验针对第2.3节中图6所描述的场景进行网元端口变更测试。在实验中，SONiC-1、SONiC-2、SONiC-3分别通过Ethernet0、Ethernet1与相邻网元设备进行连接。其中，SONiC-1的0号端口连通SONiC-2，SONiC-3的0号端口连通SONiC-2，SONiC-3的0号端口连通SONiC-1。

在此场景下，手动断开SONiC-3的物理线路，对应3个设备的端口均会发生变更。SONiC-1端口变更情况如图11所示，该图展示了所有端口状态信息，包括线路（Lanes）、速率（Speed）、最大传输单元（MTU）、校验码（FEC）、别名（Alias）、虚拟链路（Vlan）、操作状态（Oper）、管理状态（Admin）、类型（Type）和非对称优先级流量控制（Asym PFC），在这些字段中，本文重点关注Oper字段。在端口未变更前，设备网口状态中Oper参数为up表示端口开启；当端口发生变更后，设备端口状态变更为down。此状态同步至控制器中影响拓扑的变更。

admin@SONiC-1~$ show interfaces status

Interface	Lanes	Speed	MTU	FEC	Alias	Vlan	Oper	Admin	Type	Asym PFC
Ethernet0	76	25G	9100	none	eth-0-1	trunk	up	up	SFP/SFP+/SFP28	N/A
Ethernet1	77	25G	9100	none	eth-0-2	trunk	up	up	SFP/SFP+/SFP28	N/A
Ethernet2	78	25G	9100	none	eth-0-3	trunk	down	up	SFP/SFP+/SFP28	N/A
Ethernet3	79	25G	9100	none	eth-0-4	trunk	down	up	N/A	N/A
Ethernet4	80	25G	9100	none	eth-0-5	trunk	down	up	N/A	N/A
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Ethernet24	100	25G	9100	none	eth-0-25	trunk	down	up	N/A	N/A
Ethernet25	101	25G	9100	none	eth-0-26	trunk	down	up	N/A	N/A
Ethernet26	102	25G	9100	none	eth-0-27	trunk	down	up	N/A	N/A

admin@SONiC-1~$ show interfaces status

Interface	Lanes	Speed	MTU	FEC	Alias	Vlan	Oper	Admin	Type	Asym PFC
Ethernet0	76	25G	9100	none	eth-0-1	trunk	up	up	SFP/SFP+/SFP28	N/A
Ethernet1	77	25G	9100	none	eth-0-2	trunk	down	up	SFP/SFP+/SFP28	N/A
Ethernet2	78	25G	9100	none	eth-0-3	trunk	down	up	SFP/SFP+/SFP28	N/A
Ethernet3	79	25G	9100	none	eth-0-4	trunk	down	up	N/A	N/A
Ethernet4	80	25G	9100	none	eth-0-5	trunk	down	up	N/A	N/A
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
Ethernet24	100	25G	9100	none	eth-0-25	trunk	down	up	N/A	N/A
Ethernet25	101	25G	9100	none	eth-0-26	trunk	down	up	N/A	N/A
Ethernet26	102	25G	9100	none	eth-0-27	trunk	down	up	N/A	N/A

图 11　SONiC-1 端口变更情况

3.3　差异性转发单元适配实验

为验证部署 p4runtime-pins 容器的网元交换机能够兼容转发单元的差异，本文第二组实验通过对部署不同转发单元的网元设备下发 IPv4 模态、IPv6 模态、工控标识模态（powerlink，PwL）、接入级移动标识模态（access mobility first，AccMF）、核心级移动标识模态（core mobility first，CoreMF）这 5 类与控制器 P4 文件中所定义的数据包格式相关的流表规则，并通过 Python 程序与 Wireshark 工具进行打流与抓包测试。同时，观察 SONiC 系统中的 ACL 规则表项。对于直接支持 P4 流表的转发单元，控制器下发可直接部署。流表表项配置结果如图 12 所示，图 12 中上半部分为 ACL 表项描述，表示通过控制器下发的流表生产的 UDF 表项的类型和描述。其中，Binding 字段和 Udf-groups 字段表示模态的缺省端口和 UDF 组别，通过平台服务组件配置。图中下半部分表示各模态表项对应的转发规则。对于直接支持 P4 流表的转发单元，控制器下发可直接部署。每条规则通过 p4runtime-pins 生产的配置文件经设备服务组件下发网元设备。以 AccMF_EXACT 表项为例，Rule 表示转发规则，源地址标识符（src_aid）为 0x65、目的地址标识符（dst_aid）为 0x 66 和 0xc9 的模态数据包分别由 3 号端口和 2 号端口转发。在表项中匹配字段和端口字段根据芯片支持的字节填充位数。

在 PINE-B1000-P48C8A 设备中用于测试网口为 Ethernet0、Ethernet1 和 Ethernet2，分别对应 1 号、2 号和 3 号端口，因此在 Action 字段中，重定向端口分别对应 Ethernet2 和 Ethernet1。Match 字段表示 32 位匹配数据及其掩码格式。对于不直接支持 P4 流表配置的转发单元，即 CTC8180 芯片，控制器下发的流表信息将通过容器代理和转存功能进行配置。同样以 AccMF_EXACT 表项为例，部署其 src_aid 为 0x65d，dst_aid 为 0x 66 的流量规则，则需要通过代理功能转发至指定设备中并转存，流表表项配置结果如图 13 所示。转存文件读取配置如图 14 所示，通过图 14 中设备服务组件的流程进行自动读取，最终配置至 SONiC 网元设备的 ACL 表项和规则中，AccMF 表项配置结果如图 15 所示。

```
~$ show acl table
Name              Type  Binding     Description    Stage   Udf-groups
----------------  ----  ----------  -------------  ------  ----------------------------
ACCMF_EXACT       UDF   Ethernet0   accmf_exact    ingress ACCMF_EXACT_UG_8
COREMF_EXACT      UDF   Ethernet0   coremf_exact   ingress COREMF_EXACT_UG_8
                        Ethernet22
PWL_EXACT         UDF   Ethernet0   pwl_exact      ingress PWL_EXACT_UG_8
IPV4_EXACT        UDF   Ethernet0   ipv4_exact     ingress IPV4_EXACT_UG_8
                        Ethernet56
IPV6_EXACT        UDF   Ethernet0   ipv6_exact     ingress IPV6_EXACT_UG_8
~$ show acl rule
Table             Rule                                                Priority  Action                  Match
----------------  --------------------------------------------------  --------  ----------------------  ----------------------------------------------------------------------------------------------------
ACCMF_EXACT       3_0x00000066_0x0003_0x00000065                             3  REDIRECT:Ethernet2      UDF_DATAS:0xffff00000065000000066ffffffffff/0x0000ffffffffffffffff000000000000
ACCMF_EXACT       2_0x000000c9_0x0002_0x00000065                             2  REDIRECT:Ethernet1      UDF_DATAS:0xffff00000065000000c9ffffffffff/0x0000ffffffffffffffff000000000000
COREMF_EXACT      3_0x03eb_0x0003                                            3  REDIRECT:Ethernet2      UDF_DATAS:0x03ebffffffffffffffffffffffff/0xffff00000000000000000000000000000
COREMF_EXACT      2_0x03ea_0x0002                                            2  REDIRECT:Ethernet1      UDF_DATAS:0x03eaffffffffffffffffffffffff/0xffff00000000000000000000000000000
PWL_EXACT         3_0x123456789012_0x0003                                    3  REDIRECT:Ethernet2      UDF_DATAS:0x123456789012ffffffffffffffff/0xffffffffff00000000000000000000
PWL_EXACT         2_0x123456789011_0x0002                                    2  REDIRECT:Ethernet1      UDF_DATAS:0x123456789011ffffffffffffffff/0xffffffffff00000000000000000000
IPV4_EXACT        3_0x0a000004_0x0003                                        3  REDIRECT:Ethernet2      UDF_DATAS:0xffff0a000004ffffffffffffffff/0x0000ffffffff00000000000000000000
IPV4_EXACT        2_0x0a000003_0x0002                                        2  REDIRECT:Ethernet1      UDF_DATAS:0xffff0a000003ffffffffffffffff/0x0000ffffffff00000000000000000000
IPV6_EXACT        3_0x0003_0x12340000000000005678901234567892                3  REDIRECT:Ethernet2      UDF_DATAS:0x00000000000056789012345678927fff/0xffffffffffffffffffffffff0000
IPV6_EXACT        2_0x0002_0x12340000000000005678901234567891                2  REDIRECT:Ethernet1      UDF_DATAS:0x00000000000056789012345678917fff/0xffffffffffffffffffffffff0000
```

图12　流表表项配置结果（1）

```
~$ cat/tmp/flowtable.conf
p4.IngressPipeImpl.accmf_exact.add_with_accmf_set_egress_port(dst_aid=0x000000c9,src_aid=0x00000065,port=0x0002)
```

图13　流表表项配置结果（2）

```
##########ACCMF_EXACT generate acl json info:
2024-12-19 15:30:29.058530| DEBUG : generate_8180_commands_udf: 894| {
    "ACL_TABLE": {
        "ACCMF_EXACT": {
            "policy_desc": "accmf_exact",
            "stage": "INGRESS",
            "ports": [
                "Ethernet0"
            ],
            "type": "UDF",
            "udf_groups": "ACCMF_EXACT_UG_8"
        }
    }
}
2024-12-19 15:30:29.058875| DEBUG : generate_8180_commands_udf: 895| {
    "ACL_RULE|ACCMF_EXACT": {
        "3_0x00000066_0x0003_0x00000065": {
            "PRIORITY": "3",
            "PACKET_ACTION": "REDIRECT:Ethernet3",
            "UDF_DATAS": "0xffff00000065000000066ffffffffff/0x0000ffffffffffffffff000000000000"
        }
    }
}
2024-12-19 15:30:29.060138| DEBUG : generate_8180_commands_udf: 914|

##########ACCMF_EXACT generate acl cli info:
2024-12-19 15:30:29.510889| DEBUG : sonic_getstatusoutput_one_time: 158| sudo sonic-cfggen -j /tmp/genereate_accmf_exact_acl_table_dict.json -w
2024-12-19 15:30:30.069517| DEBUG : sonic_getstatusoutput_one_time: 158| sudo sonic-cfggen -j /tmp/genereate_accmf_exact_acl_rule_dict.json -w
2024-12-19 15:30:31.344083| DEBUG : sonic_getstatusoutput_one_time: 158| show acl rule | grep ACCMF_EXACT
2024-12-19 15:30:31.344458| DEBUG : sonic_getstatusoutput_one_time: 158| ACCMF_EXACT 3_0x00000066_0x0003_0x00000065 3 REDIRECT:Ethernet3 UDF_DATAS:0xffff000000
 6500000066ffffffffff/0x0000ffffffffffffffff000000000000
```

图14　转存文件读取配置

```
~$ show acl table
Name              Type  Binding     Description    Stage   Udf-groups
----------------  ----  ----------  -------------  ------  ----------------------------
ACCMF_EXACT       UDF   Ethernet0   accmf_exact    ingress ACCMF_EXACT_UG_8
~$ show acl rule
Table             Rule                                Priority  Action                  Match
----------------  ----------------------------------  --------  ----------------------  ----------------------------------------------------------------------------------------------------
ACCMF_EXACT       3_0x00000066_0x0003_0x00000065             3  REDIRECT:Ethernet2      UDF_DATAS:0xffff00000065000000066ffffffffff/0x0000ffffffffffffffff000000000000
```

图15　AccMF表项配置结果

3.4　p4runtime-pins容器资源消耗评估

在SONiC操作系统中，本身带有不同功能的容器结构，如syncd、swss、database等。其中，syncd为SONiC中管理ASIC的主要容器，其核心进程与托管Redis数据库的database容器沟通，加载SAI接口库并与其交互，完成ASIC的初始化，

配置和状态上报的处理等。在p4runtime-pins和设备服务组件配置流表的过程中，也需与syncd和database等容器直接或间接交互。

为了对比本文提出的容器与SONiC自身容器的差异，并判断控制通道容器对SONiC系统资源的影响，本文针对p4runtime-pins容器部署前后SONiC中各容器的CPU和内存占用情况进行分析。p4runtime-pins部署前后各容器CPU资源消耗如图16所示，p4runtime-pins部署前后各容器内存资源消耗如图17所示。其中，左右柱形分别表示p4runtime-pins容器部署前后各容器的CPU和内存平均占用情况。第一个柱形表示p4runtime-pins容器的资源占用情况。实验结果表明，网元设备中各容器在p4runtime-pins容器部署前后CPU和内容的占用情况并无较大差异。同时，p4runtime-pins容器CPU占用率为1.70%，内存占用率为0.45%，相较于其他容器的CPU占用率正常，内存占用率较低。

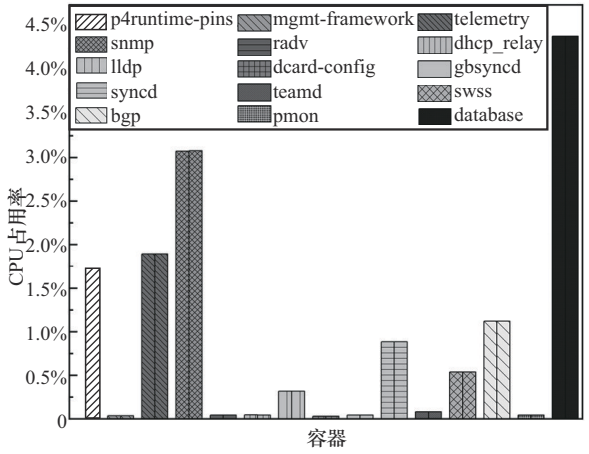


图16　p4runtime-pins部署前后各容器CPU资源消耗

此外，本文还测试了p4runtime-pins容器运行10 min内每秒钟的CPU和内存变化，并以每分钟作为采样点，取该采样点1 min内的数据均值为采样值。10 min内各容器CPU变化如图18所示，10 min内各容器内存变化如图19所示，从图中可以得出，p4runtime-pins容器的CPU和内存占用率数值上与上文结果一致，同时CPU占用率与其他容器相比波动不明显，较为稳定。各容器内存占用率稳定，图19中，结果因波动数值较小，表现接近水平。

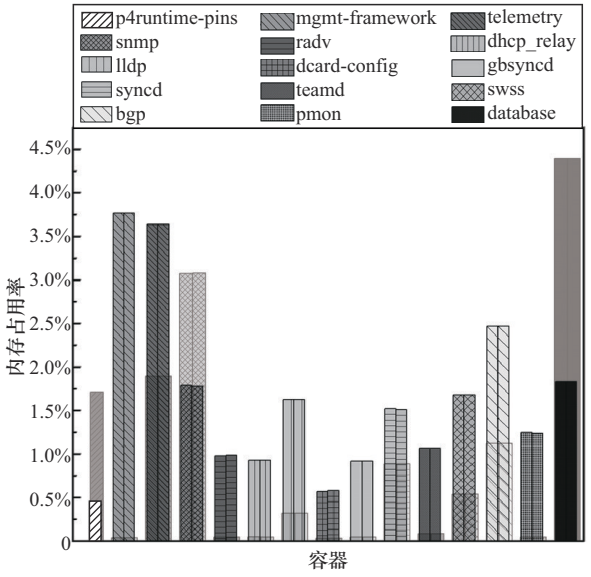


图17　p4runtime-pins部署前后各容器内存资源消耗

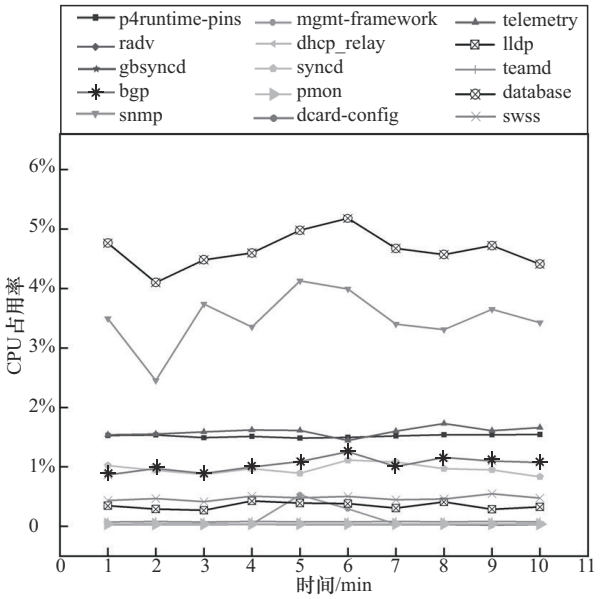


图18　10 min内各容器CPU变化

虽然容器本身有着不高的资源占用，但是p4runtime-pins作为控制通道接口容器，对于来自控制端下发流表的接收准确性及性能也至关重要。为了验证流表接收及部署性能，本文还对比

了控制器下发流表至网元设备中准确部署流表后，流经设备中流量的变更切换速度。

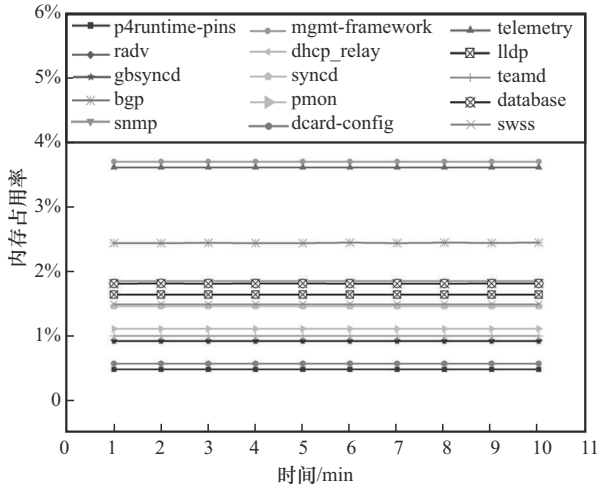


图19　10 min内各容器内存变化

流表切换动态调整时间如图20所示，图中横坐标表示多模态流表类型，纵坐标表示在流量测试中各模态在控制端流表下发后，流量按照流表规则变更转发端口的动态调整时间，即流表切换动态调整时间。实验结果表明，各个模态的流表切换动态调整时间在0.027~0.037 s，调整时间相对稳定。

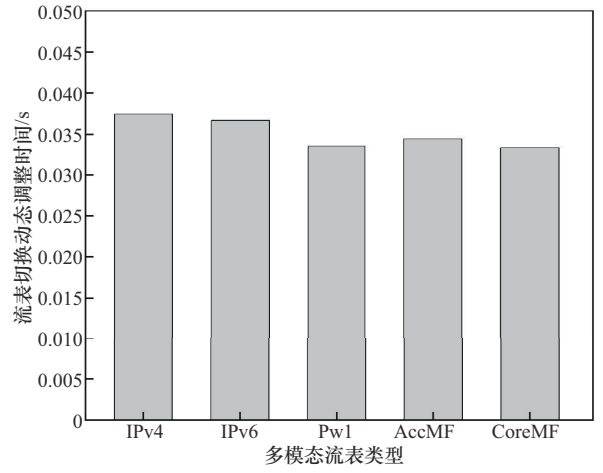


图20　流表切换动态调整时间

4　结束语

本文设计了面向SONiC交换机操作系统的多模态网元控制通道容器p4runtime-pins。该控制通道容器通过P4Runtime协议提供的gRPC接口，构建gRPC服务器与搭载P4Runtime协议框架的SDN控制器建立gRPC通信，能够自定义实现多模态数据包的控制信息接收配置。实验结果表明，p4runtime-pins能实现多模态网络数据包流表的成功接收与配置。与SONiC原生容器相比，p4runtime-pins资源占用少，流量的变更切换延迟仅为0.027~0.037 s。

参考文献：

[1] 胡宇翔, 伊鹏, 孙鹏浩, 等. 全维可定义的多模态智慧网络体系研究[J]. 通信学报, 2019, 40(8): 1-12.
HU Y X, YI P, SUN P H, et al. Research on the full-dimensional defined polymorphic smart network[J]. Journal on Communications, 2019, 40(8): 1-12.

[2] 李丹, 胡宇翔, 邬江兴. 新型网络技术创新发展战略研究[J]. 中国工程科学, 2021, 23(2): 15-21.
LI D, HU Y X, WU J X. Innovative development strategy of new network technologies[J]. Strategic Study of CAE, 2021, 23(2): 15-21.

[3] 黄韬, 刘江, 霍如, 等. 未来网络体系架构研究综述[J]. 通信学报, 2014, 35(8): 184-197.
HUANG T, LIU J, HUO R, et al. Survey of research on future network architectures[J]. Journal on Communications, 2014, 35(8): 184-197.

[4] 张成林, 宋玲玲. 面向未来网络的白盒交换机体系综述[J]. 信息技术与网络安全, 2022, 41(3): 1-8.
ZHANG C L, SONG L L. A review on architecture of white box switches for future networks[J]. Information Technology and Network Security, 2022, 41(3): 1-8.

[5] SCANO D, GIORGETTI A, SGAMBELLURI A, et al. Hierarchical control of SONiC-based packet-optical nodes encompassing coherent pluggable modules[C]//Proceedings of 2021 European Conference on Optical Communication (ECOC). Piscataway IEEE Press, 2021: 1-3.

[6] PIASETZKY Y, KADOSH M, PRITSAK M, et al. Switch asic programmability in hybrid mode[C]//Proceedings of 2018 IEEE 26th International Conference on Network Protocols (ICNP). Piscataway IEEE Press, 2018: 448-449.

[7] 董永吉, 胡宇翔, 崔鹏帅. 存转算一体的多模态网络共性平台技术研究[J]. 中兴通讯技术, 2022, 28(1): 16-20, 74.
DONG Y J, HU Y X, CUI P S. Research on multi-modal net-

work common platform technology with storage-transfer-calculation integration[J]. ZTE Technology Journal, 2022, 28(1): 16-20, 74.

[8] 郜帅, 侯心迪, 刘宁春, 等. 多模态网络环境异构标识空间管控架构研究[J]. 通信学报, 2022, 43(4): 26-35.
GAO S, HOU X D, LIU N C, et al. Research on heterogeneous identifier namespace management and control architecture in polymorphic network environment[J]. Journal on Communications, 2022, 43(4): 26-35.

[9] BERDE P, GEROLA M, HART J, et al. ONOS: towards an open, distributed SDN OS[C]//Proceedings of the Third Workshop on Hot Topics in Software Defined Networking.[S.l.:s.n.], 2014: 1-6.

[10] PETERSON L, CASCONE C, O'CONNOR B, et al. Software-defined networks: a systems approach[M]. Princeton: Princeton University Press, 2021.

[11] O'CONNOR B, TSENG Y, PUDELKO M, et al. Using P4 on fixed-pipeline and programmable stratum switches[C]//Proceedings of the 2019 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). Piscataway: IEEE Press, 2019: 1-2.

[12] BOSSHART P, DALY D, GIBB G, et al. P4[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(3): 87-95.

[13] 董永吉, 王钰, 袁征. 基于FPGA的万兆以太网UDP_IP硬件协议栈设计与实现[J]. 计算机应用研究, 2022, 39(8): 2465-2468.
DONG Y J, WANG Y, YUAN Z. Design and implementation of 10G Ethernet UDP/IP hardware protocol stack based on FPGA[J]. Application Research of Computers, 2022, 39(8): 2465-2468.

[14] ALBAB K D, DILORENZO J, HEULE S, et al. SwitchV: automated SDN switch validation with P4 models[C]//Proceedings of the ACM SIGCOMM 2022 Conference. New York: ACM Press, 2022: 365-379.

[15] STUBBE H , GALLENMÜLLER S ,SIMON M , et al. Exploring data plane updates on p4 switches with P4Runtime[J].Computer Communications,2024,225(9): 44-53.

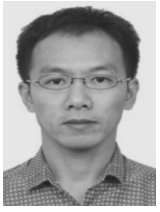
[作者简介]

涂化清（1995– ），女，博士，之江实验室在站博士后、助理研究员，主要研究方向为软件定义网络、新型网络架构、多模态网络等。

方徐鑫（1999– ），男，浙江工商大学硕士生，主要研究方向为多模态网络、可编程网络。

朱俊（1981– ），男，博士，之江实验室高级工程师，主要研究方向为新型网络体系结构、软件定义网络、网络资源管理、网络协议设计优化等。

邹涛（1974– ），男，博士，之江实验室研究员，主要研究方向为多模态网络技术、新型网络体系架构。



李传煌（1980– ），男，博士，浙江工商大学教授、硕士生导师，主要研究方向为软件定义网络、开放可编程网络、系统性能预测与分析模型、算力网络、智慧网络。

隆克平（1968– ），男，博士，之江实验室高级研究专家，北京科技大学教授、博士生导师，主要研究方向为新一代网络技术、光互联网关键技术、无线通信技术、人工智能与大数据。