用P4对数据平面进行编程

作者：尼克·麦克欧文(Nick McKeown)[1,3]
金昶勳(Changhoon Kim)[2,3]
[1]斯坦福大学
[2]Barefoot Networks
[3]P4.org
译者：高荣新(Ron Kao)

关键词：P4 语言联盟　可编程数据平面

引言

软件定义网络因其使网络拥有者和运营商能够对网络行为进行编程而取得了巨大的成功。然而，其可编程性目前仅局限于网络控制平面，其转发平面在很大程度上受制于功能固定的包处理硬件。P4语言联盟 (www.P4.org) [1] 及其开源活动旨在完全摆脱网络数据平面的束缚，让网络拥有者、工程师、架构师及管理员可以自上而下地定义数据包的完整处理流程。

灵活的网络数据平面将加速网络和计算在不同子领域的创新。事实上，这是计算机历史上几次相同技术变革模式的又一次重现：廉价的可编程器件的出现必然会推动其上层应用的创新和发展，给业界带来新一轮革新。

在网络领域，这个故事才刚刚开始，并在新一代高性能可编程数据包处理芯片的助力下成为可能。在高性能领域，内置协议无关的交换架构 PISA 芯片能提供每秒 Tb[2] 级别的数据包处理速度，以及完全可编程的数据包解析与通用的“匹配 - 动作”能力。在中低性能领域，服务器级或嵌入式的中央处理器 (CPU)、图形处理器 (GPU)、现场可编程门阵列 (FPGA) 和网络处理器 (NPU) 早已能够提供每秒几十到几百 Gb 速度级别的数据包处理速度以及灵活的处理过程。

除了可编程转发芯片，我们还需要“P4”(www.p4.org)[1,3] 这样的高级语言，以不受限于具体目标设备（目标无关）的方式控制转发行为。程序员首先用 P4 定义数据包的处理流程，然后利用编译器在不受限于具体协议（协议无关）的交换机或网卡上生成具体的配置，从而实现用 P4 表达的数据包处理逻辑。程序员通过编程，可以将交换机变为一个架顶交换机 (Top-Of-Rack, TOR)、一道防火墙或一个负载平衡器，或者支持新的自动诊断功能和新的拥塞控制算法等。

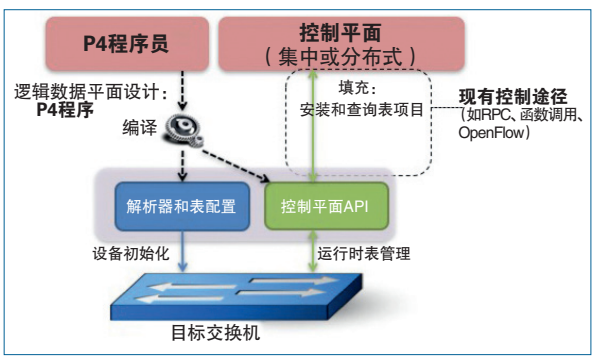

图1　P4是编程交换机数据平面的语言

图 1 显示了 P4（编程交换机数据平面，从而说明数据包如何被处理）与控制平面应用程序（在运行时生成转发表）之间的关系。需要注意的是，通

过编译 P4 程序，不仅可以在可编程的数据包处理硬件上生成具体的配置，还可以生成运行时应用编程接口，来帮助控制和数据平面之间的交互。

本文将从比较抽象的层面介绍 P4 语言和可编程数据平面技术。如想了解更多技术细节，建议从《P4 的语言规范》[4]（最初是 2014 年在 *SIGCOMM CCR* 上发表的 P4 论文 [3]）和《开源 P4 开发工具》[5] 入手。

可编程数据平面将达到什么目标?

可编程数据平面将带来广泛而巨大的影响。我们预计几乎所有的网络领域参与者都将从中获益。

可编程数据平面有助于网络系统供应商进行更快速的迭代开发，迅速推出新的功能，甚至直接通过打补丁修复现有产品中发现的数据平面程序漏洞。数据平面的灵活性使得各个供应商能够在系统功能和性能上差异化，以有别于其他厂商。最根本的是，这些系统供应商可以从软件产业过去几十年已发展成熟的软件编程理论、实践和工具中受益。

可编程数据平面也可以帮助网络拥有者（例如在线服务提供商、运营商和企业）实现最适合其自身需求的具体网络行为。此外，许多大型运营商拥有的大量软件开发人员能够轻松学会对数据平面的网络设备进行编程、测试和调试，将网络变成一个可编程的平台，以一个完全可编程的方式来管理网络。例如，他们可以开发自定义的网络监控、分析和诊断系统，实现前所未有的网络可视化和相关性，从而大幅度降低网络的运营成本。他们还可以协同优化网络以及在网络上运行的应用程序，来保证最佳的用户体验。

对于网络芯片供应商，可编程数据平面使他们能专注于设计并改进那些可重用的数据包处理架构和基本模块，而不是纠缠特定协议里错综复杂的细节和异常行为。而且，一旦证明这些架构和基本模块可行，供应商就可以在多代交换芯片的设计中重复使用它们，不必为客户不断产生的新需求而反复修改。

对于网络和分布式系统的研究人员，可编程的数据平面为他们实现并验证新想法（尤其是那些引入新的数据平面行为的项目）提供了新的契机。因为只有“合适”的硬件才能真正应对实际部署中所有具有挑战性的需求，如高端口速度、全线速发送小数据包、低延迟、高端口数等。

可编程的数据平面技术具有如下优点：

新功能 可以快速和频繁地开发新的网络功能，比如新的自定义功能或标准的数据包头规范和转发行为。

降低复杂性 可以去掉冗余的而只保留网络必需的数据平面功能。例如，大多数大型数据中心网络通常只使用L2 & L3转发、ECMP（等价多路径）、LAG（链路聚合）和 ACL（访问控制列表）。其他复杂的协议，比如 MPLS（多协议标签交换）、QoS（服务质量）和 IP 多播等，都是多余的负担。在设备的数据和控制平面中仅保留有用的协议，可以减少潜在的漏洞、被攻击的可能性和运维的复杂性。

有效利用资源 通过去掉不必要的功能，可以释放所耗费的物理资源（例如内存、纵横式交换矩阵和计算逻辑单元），然后重新分配给必要的功能，以实现最大的效能。

增强可视化 当前，只具有固定功能的数据包处理硬件并且能够生成的网络监控信息的数量和质量都非常有限。这是因为监测、分析和诊断等功能往往被认为是一种事后的补救措施，所以其优先级总是低于数据包转发功能。一个无法根据特定协议转发数据包的交换机芯片在市场上是毫无竞争力的，而一个能转发数据包但只提供较少或低质量检测信息的交换机芯片在市场上则有一定的竞争力。因此，考虑到要在有限的硬件资源（如尺寸和功耗）上实现各种数据平面的功能，固定功能硬件的设计者要始终优先设计转发功能，而后才是监控功能。有了可编程数据平面，网络拥有者可以控制转发和监控功能之间的平衡，因为其真正想要的转发功能往往只是硬件厂商提供的功能列表中的一个很小的子集。此外，网络拥有者还可以明确地定义所需的监控功能的语义，以保证监控信息的相关性。

模块化 我们可以通过重用别人开发的P4代码库来实现自己想要的转发行为。这将大大简化和方便个人用户的P4开发进程。

可移植性 当开发了一个P4程序后，我们可以为不同目标硬件重新编译代码，将此程序重复用于不同类型的目标上。这可以显著提升编程、测试和调试整个网络行为的效率，因为我们只要在所有的目标设备上使用相同的P4程序就可以保证数据和控制平面之间接口的统一。因此，控制平面可以被重复使用，而无须或只须最少的改动。

拥有自己的知识产权 P4程序员不需要与硬件厂商分享转发逻辑（功能）。这使得开发一个P4程序能得到相应的回报，从而激发网络拥有者们自主创新的热情。

协议无关的交换机架构

可编程的数据平面技术具有很多优点，但所有这些都是基于廉价的可编程数据包处理器。虽然中低端性能的数据包处理器——CPU、现场可编程门阵列和网络处理器——早已存在，并已被用来实现具有复杂数据平面功能的网络系统，但这还远远不够。如果要想具备和现有的功能固定的数据包处理硬件相似的性价比，我们还需要实现高性能的系统芯片级可编程数据包处理器。

网络硬件行业的最新技术趋势表明，实现这样的硬件是可行的[2,6,7]，并且相对于固定功能设计，可编程性带来的额外开销可以做到足够低。这种数据包处理器的一种通用架构是协议无关的交换机架构(Protocol-Independent Switch Architecture, PISA)。

图2展示了PISA的高层架构。PISA是一种具有完全可编程数据包解析器和通用“匹配-动作”单元的高速数据包处理器。关于可编程解析器的技术在近期已有广泛研究[8]，且最早的P4论文中也提供了一个案例来说明如何使用三元匹配(ternary matching)能力实现有限状态机。通用的“匹配-动作”逻辑是通过静态随机存储器(SRAM)、三态内容寻址存储器(TCAM)、寄存器、灵活的哈希(Hash)生成器和支持通用数据包处理指令集的计算逻辑单元来实现的。博斯哈特(Bosshart)等人最近提出了一种通用“匹配-动作”逻辑的详细设计[2]。PISA中有大量“匹配-动作”单元，且所有这些单元都是相同的。这种同质化使得PISA成为一个极佳的编译器目标，编译器可以灵活编程每个“匹配-动

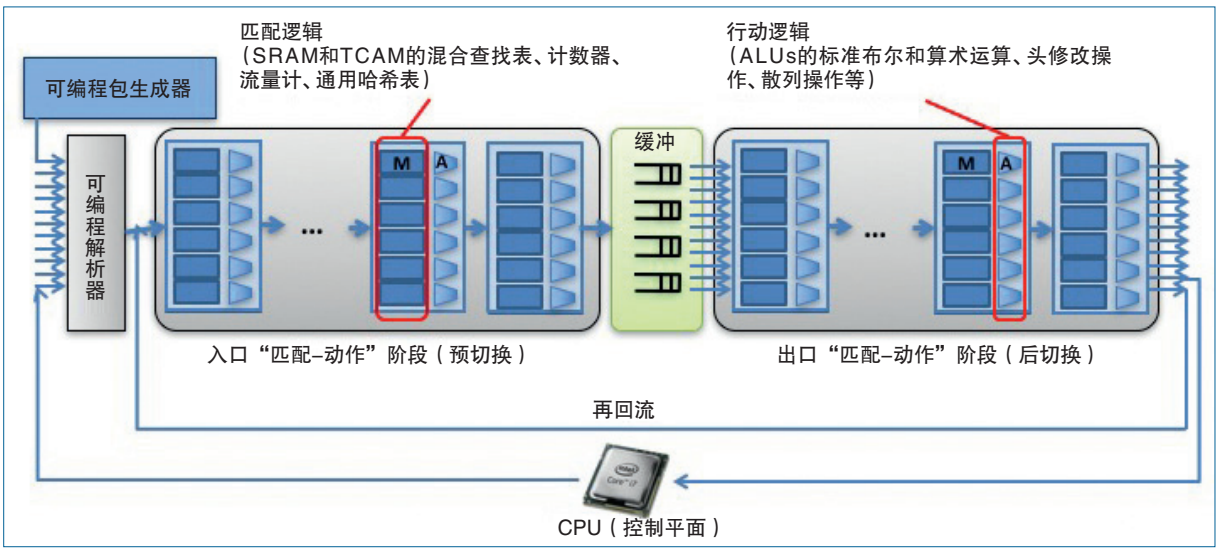

图2 协议无关的交换机架构。到达PISA交换机的数据包先由可编程的解析器解析，再通过入口侧一系列的“匹配-动作”阶段，然后经由队列系统交换，由出口“匹配-动作”阶段再次处理，最后重新组装发送到输出端口

作”单元来实现需要的数据包处理功能。需要注意的是，解析器和“匹配 - 动作”单元必须在被编程后才能做具体的数据包处理工作；实际上，没有配置的 PISA 芯片不会实现任何数据平面协议。这就是为什么我们称之为协议无关架构。

PISA 从两个并行的维度保证高速率的数据包处理。第一，多个数据包处理阶段以流水线的方式工作。也就是说，每一个阶段可以相互独立地处理少量的数据包，因此在特定时间内多个数据包可以在流水线的不同阶段同时存在，从而提高芯片的处理速度。由于这些阶段都循序安排，PISA 还可以处理彼此依赖的数据包任务（如表格查找和包头修改）。例如，假设用户希望实现的是一个包头字段的先写后读的操作，编译器可以定义写的操作给 i 阶段的“匹配 - 动作”单元，另外定义“读”的操作到 i+1 阶段的单元。第二，在每个阶段，有大量的“匹配 - 动作”单元。这让编译器可以分配独立(即“可并行”) 的数据包处理操作到同一阶段的“匹配 - 动作”单元，同样提高了处理速度。

除了可编程解析器和通用的“匹配 - 动作”流水线，PISA 还有一些有价值的基本模块。经流水线修改过的数据包头在传输到输出端口之前需要重新组合。这个逆向解析过程同样受益于灵活性，因此 PISA 引入了可编程的逆解析操作。为了提高可编程性，PISA 还提供了一条回流路径使一些特殊数据包能够被多次反馈到解析器和流水线。编译器也可以使用这个回流功能来实现传统芯片因只有固定的查找修改单元数量而无法完成的复杂逻辑。PISA 还可以嵌入可编程数据包生成器，使 CPU（控制平面）可以将频繁或周期性的数据包生成操作转移到数据包生成器。特定的包头值或实例的出现也可以触发数据包生成，从而进一步丰富可编程性。这与一个通用 CPU 事件驱动的处理能力（例如中断）相似。PISA 还提供了连接数据和控制平面之间的高带宽通道。

P4语言简介

要编程 PISA 和其他类型的可编程数据包处理器，需要网络工程师和运营商所熟悉的领域和一种专用的编程语言。P4 就是这样一种语言，其目标如下：

协议无关性 网络设备不应该与任何特定的网络协议捆绑在一起。相反，对于任何所需要的网络数据平面协议和数据包处理行为，程序员都可以使用 P4 进行表达。

目标无关性 程序员无须关心底层硬件细节，即可描述数据包处理功能。

现场可重配置能力 程序员应在部署交换机后能对数据包的处理方式进行再次修改。

P4程序定义

P4 程序主要由如下组件构成：

包头 定义描述一系列字段的顺序和结构，包括字段的宽度和字段值的限制。

解析器 定义如何识别数据包内的包头和有效的包头序列。

表 “匹配 - 动作”表是执行数据包处理的机制，P4 程序定义了在表内可以匹配的字段和可以执行的操作。

动作 基于一系列预先定义的和协议目标无关的简单基本操作，P4 可以构建复杂的自定义动作，并且可以在“匹配 - 动作”表中使用。

控制程序 决定以什么顺序用“匹配 - 动作”表处理数据包，一个简单的命令式程序就可以描述“匹配 - 动作”表之间的控制流。

以下是关键组件的一些简单实例。

包头类型实质上是由成员字段组成的有序列表，每个字段有其名称和长度。每一种包头类型都有对应的包头实例来存储具体的包头数据。P4 也允许建立模型将数据包元数据（metadata，任何不包含在该数据包内，而是衍生自数据包包头或底层硬件的数据）作为包头类型。下面是以太网和 IPv4 包头类型及其实例。

```
header_type ethernet_t {
  fields {
    dst_addr : 48; // width in bits
    src_addr : 48;
```

```
        ethertype : 16;
    }
}
header ethernet_t ethernet;
header_type ipv4_t {
    fields {
        version : 4;
        ihl : 4;
        diffserv : 8;
        totalLen : 16;
        identification : 16;
        flags : 3;
        fragOffset : 13;
        ttl : 8;
        protocol : 8;
        hdrChecksum : 16;
        srcAddr : 32;
        dstAddr: 32;
    }
}
header ipv4_t ipv4;
```

一旦定义了包头实例，即可表达数据包的解析逻辑：硬件将输入的字节序列转换为有效的包头实例集合（也称为数据包的解析后表达）的过程。有限状态机的概念很适合模拟该过程，所以 P4 借用了这个概念。在有限状态机的模型下，底层的数据包处理器经过多个解析状态，从预先定义的启动状态开始，在预先定义的终止状态结束。下面的例子说明了如何描述以太网和 IPv4 包头的解析逻辑。

```
parser start {
    return parse_ethernet;
}
parser parse_ethernet {
    extract(ethernet)
    return switch(latest.ethertype) {
        case 0x8100: parse_vlan;
        case 0x0800: parse_ipv4;
        // Other cases
    }
}
parser parse_ipv4 {
    extract(ipv4);
    return select(latest.protocol) {
        IP_PROTOCOLS_IPHL_ICMP :parse_icmp;
        IP_PROTOCOLS_IPHL_TCP :parse_tcp;
        IP_PROTOCOLS_IPHL_UDP :parse_udp;
        IP_PROTOCOLS_IPHL_GRE :parse_gre;
        IP_PROTOCOLS_IPHL_IPV4 : parse_ipv4_in_ip;
        IP_PROTOCOLS_IPHL_IPV6 : parse_ipv6_in_ip;
        // Other cases
        default: ingress;
    }
}
```

P4 的表和动作是“匹配 - 动作”过程模型中的关键概念。P4 表是一个包含很多条目的逻辑结构。包头字段（如匹配键）与表中的条目按照与包头字段相关联的特定匹配语义进行匹配。匹配的语义包括完全匹配、三元匹配、最长前缀匹配和范围匹配。一经匹配或失配（没有匹配项），就执行一个与数据包及相关元数据对应的动作。

在 P4 中，所有动作都是用户自定义的。一个动作其实就是一个自定义的执行序列，包含若干个基本模块指令，配合在运行时由控制平面传递给动作的参数并以此来处理包头字段或元数据。P4 语言规范 [4] 中定义了标准的基本模块指令集，而下层目标设备还可以提供更多的指令。下面是如何定义一个 IPv4 路由表和相关的动作来实现 IPv4 路由的示例。

P4 语言规范规定的基本模块指令集包括：数据包处理运算符（如添加、删除或修改包头）、基本的算术运算符、哈希运算符和统计跟踪运算符（如计数、测量）。

```
#define IPV4_LPM_TABLE_SIZE 65536
table ipv4_fib_lpm {
    reads {
        metadata.vrf : exact;
        ipv4.dstAddr :lpm;
```

```
    }
    actions {
      on_miss;
      fib_hit_nexthop;
    }
    size : IPV4_LPM_TABLE_SIZE;
  }
  action on_miss() {
      // no op
  }
  action fib_hit_nexthop() {
    modify_field(metadata.fib_hit, TRUE);
    modify_field(metadata.fib_nexthop, nexthop_
index);
    add_to_field(ipv4.ttl, -1);
  }
```

一旦定义好包头、解析器、表和动作，剩下的任务则是指定从一个表到下一个表的控制流。控制流是一个由条件语句和表的引用组成的命令式程序。下面是如何实现一个自定义的控制流的例子：数据包首先经过 L2 转发表 (12_fwd)，然后可能经过 L3 路由表（ipv4_fib_lpm 和 ipv6_fib_lpm），这取决于数据包的以太网目的地址是否匹配路由器自身的 MAC 地址（通过查找所属路由器的 router_mac 表）。根据 IP 包头类型（IPv4 或 IPv6），数据包将经过不同的 L3 路由表。最后采用访问控制列表来决定是否允许数据包通过。

```
control ingress {
  // L2 forwarding
  apply(l2_fwd);
  // L3 routing
  apply(router_mac) {
    hit {
      if (valid(ipv4)) {
        apply(ipv4_fib_lpm);
      } else {
        if (valid(ipv6)) {
          apply(ipv6_fib_lpm);
        }
      }
    }
  }
  // ACL
  apply(acl);
}
```

编译器功能

P4 编译器本质上是将在 P4 程序中表达的数据平面的逻辑翻译成一个在特定可编程数据包处理硬件上的具体物理配置。因此，编译器后端部分自然与其支持的硬件目标紧密结合，而其前端部分则可以在各个 P4 可编程目标之间通用。这就意味着一个 P4 程序的具体实现可根据被编译的目标而改变。

PISA 编译器可以从 P4 程序的包头和解析器定义中导出解析器和重组器的配置。P4 采用的状态机概念使得此映射变得相对容易。PISA 编译器可以从表、动作和控制流的定义中导出“匹配 - 动作”阶段的配置。编译器首先分析每个包头字段、元数据和状态对象在表和动作之间的所有依赖关系。基于这个结果，识别出可放置在相同阶段同时运行的表和动作，以及那些由于依赖性而必须顺序执行的表和动作。同时编译器还应考虑到其他由特定目标带来的相关限制，如可用的表内存、计算逻辑单元和携带数据包包头的寄存器等。

阿尼路达 (Anirudh) 等人最近推出了一些可重用的技术，用于分析由状态对象（寄存器）造成的复杂的依赖关系，介绍了如何将一系列指令映射到不同功能的计算逻辑单元上 [9]。拉万亚 (Lavanya) 等人的文章介绍了如何将所需的表放置在有资源限制的特定目标设备上，同时使这些表能够符合其 P4 程序中所描述的依赖关系 [10]。

现状和展望

P4 自首次发表 [3] 以来就引起了工业界和学术界的极大关注。业内专家和主要学术团队受其启发

组建了 P4 语言联盟 [1]，并正在推广该语言，以期形成全行业通用的数据平面编程框架 [4]。目前已有许多具体的 P4 案例研究和语言演进的技术评论，发表在相关论文和报告 [11~13] 中。

P4语言联盟

P4 语言联盟是一个开源社区，由工业界和学术界成员组成。它有两个目标：(1) 定义 P4 语言的正式规范；(2) 维持开放源码的 P4 开发工具和 P4 的参考程序 [5]。所有资料都在非常宽松的 Apache 2.0[14] 许可下通过 www.P4.org 免费使用和重新发布。

P4 语言联盟通过语言设计工作组产生正式的语言规范，组织成员的代表参与工作组的活动。当前得到广泛支持的语言规范是 1.0.2 版本。P4 语言联盟已经用此版本发布了一个 P4 参考程序（取名为“switch.p4”），能实现各种流行的标准数据平面协议和功能，包括 L2 和 L3（IPv4 和 IPv6）转发、虚拟局域网 (VLAN)、生成树协议 (STP)、等价多路径、链路聚合、虚拟路由和转发 (VRF)、IP 组播、多协议标签交换、各类隧道协议（如虚拟可扩展局域网、通用路由封装、IP-in-IP 和 Q-in-Q）、数据包镜像、服务质量控制、访问控制、RPF 验证、传输协议（TCP、UDP 等），甚至还有一个 OpenFlow 功能的数据平面。尽管具有如此丰富的功能 [1]，截止到 2016 年 3 月，switch.p4 只有约 7000 行的 P4 代码。这表明，P4 在可读性和抽象性之间达到了很好的平衡，这对网络工程师和网络系统设计者极具吸引力。阿尼路达等人描述了在原有的 P4 论文基础上，如何增加语言结构以实现许多有实际应用的数据平面协议 [11]。

最新的预发布的语言规范为 1.1.0 版 [15]，相对 1.0.2 版本增加了以下功能：

强类型 语言中的所有数据类型、有效的运算符和操作规则都已明确定义，动作参数的类型亦经明确分类以避免歧义。

表达式 最新版本支持通用（一元、二元、三元、按位运算及逻辑运算）运算符组成的表达式。这使得语言有更强的表达能力且使用方便，无须支持更多的基本模块操作。

顺序执行语义 在 1.0.2 版本，用户自定义的动作是用并行执行的语义来解释的，这使得 P4 的开发者很难推理动作的实际行为。而 1.1.0 版本改用顺序执行语义，但这并不妨碍在实际数据包处理的时候并发执行用户定义的动作。我们应该让编译器，而不是 P4 的开发者，利用 P4 程序和硬件的能力并行执行用户定义的动作。

除了维护语言规范和开源参考实例外，P4.org 还举办各种活动（如 P4 教程 [16]、研讨会 [17,18] 和编程马拉松）以推广 P4 并支持 P4 的开发社区。

开发环境

由 P4.org 发布的开源 P4 开发工具包括参考编译器、软件交换机（称为行为模型）、参考程序以及测试和调试框架 [5]。在掌握了这些工具以后，开发者就可以编写、编译自己的 P4 程序，使用行为模型来运行程序，并用测试框架对其进行测试。测试框架既支持单一设备，也支持使用 Mininet[2] 进行网络级测试 [19]。

在基于 P4 的数据平面编程里，一个微妙而重要的方面是生成运行时 API 的方式。对于传统的固定功能包转发设备来说，每个设备都有其自己的运行时 API（常被误称为软件开发工具包 (SDK)）。这些 API 本质上是一些非常复杂的库，是设备供应商手工编写的，并且只适用于该供应商的设备。这些 API 由于来自不同设备供应商而缺乏一致性和互相兼容性，妨碍了我们在网络里部署异构（不同厂商）的数据包转发设备。

对于 P4 来说，API 是由编译器在编译时根据一个给定的 P4 程序自动生成的。因此，P4 程序的开发者也完全定义了运行时 API。这种做法有一个重要的好处：跨平台可移植性。只要网络拥有者为其

[1] 有关支持该参考程序的最新协议和功能的列表，请访问https://github.com/p4lang/switch/tree/master/p4src。

[2] Mininet是一个轻量级软件定义网络和测试平台。

设备使用相同的 P4 程序，这些设备产生的 API 就是相同的。所以 P4 的应用程序可以横跨各类型的 P4 可编程设备，实现可移植性。

最近已经推出多个 P4 可编程的目标设备，还有更多即将出台。一些厂商已经展示了其基于 FPGA 或基于 GPU 的 P4 可编程网卡 [20]。P4 也可以作为一个领域专用语言来扩展基于软件的可编程的转发设备。OVS、eBPF 和 DPDK 都是很好的例子 [21,22]。如果程序员想要修改这些目标以增添新的数据平面协议，他们需要具备丰富的软件开发经验，熟悉大型的、复杂的代码库。P4 能大幅度地降低这类任务的复杂度，因为 P4 可以让程序员只关注网络协议设计，而不用去理解、扩展和测试一个大型而复杂的代码库，比如内核网络栈。这种方法也使得基于软件的包处理器的部署和维护更加方便。例如，在 OVS 中实现一个新的数据平面协议可能需要修改 Linux 内核中的代码，而使用 P4 可编程的 OVS，可以将特定协议的实现和 OVS 的底层实现完全脱离。

语言演化路线

随着 P4 被广泛应用于编程各种类型的可编程数据包处理器，我们确定了两个额外的目标：兼容异质性和代码重用。

兼容异质性包含功能异质性和架构异质性两个方面。功能异质性源于不同类型的数据包处理器的不同功能，如校验、验证和更新、先进的模式匹配、加密 / 解密、状态操作等。架构异质性源于不同类型的数据包处理器的不同设备架构。

语言设计者所面临的挑战是：以前后一致和统一的方式解决这一异质性挑战，同时保持语言核心部分的简洁。语言设计工作组一直在探索一些新的语言结构（如 extern type 和架构模型）来解决这个问题。最新预发布的 P4 规范的附录（第 17.7 节）详细解释了这些概念 [15]。

代码重用对于建设一个繁荣的语言生态系统，以及提升该语言的整体开发效率至关重要。我们确定了提高 P4 重用性的三个具体子目标：**可移植性、架构和语言的分离、组合性**。

研究问题

随着 P4 带来了可编程的数据平面，许多令人兴奋的研究课题也随之产生：

- P4 应该如何演变？对于“语言演化路线”一节提出的问题，如何找到正确的方向来解决，这将是一个重要的研究贡献。
- 如何测试两个 P4 程序在编译前后的等效性？
- 给定一个 P4 程序，不论表目是否被填充，能否自动地产生一组完备的测试用例，来确认它具有某些性质？
- 如何证明 P4 程序的正确性？
- 能否直接从更高级别的网络策略产生 P4 程序？
- 利用可编程数据平面，能实现什么样新颖的网络应用程序？在近期的论文和讲座中已经出现了一些出色的和具有广阔前景的应用，包括先进的网络监控和分析 [23~25]、把中间层功能嵌入到交换机 [26]、通过在网络里的处理来协助分布式应用 [27,28]、协同设计网络数据平面和在其上运行的分布式应用程序 [29]。我们相信，学术界和产业界已开始触及由可编程数据平面技术带来的无穷可能。

结论

我们都知道如何在 CPU 上编程，使之易于创建新的程序，试验新的想法，构建新的应用，并与他人分享。但今天，常见的不只是可编程的 CPU，还有 GPU、智能手机的应用处理器，甚至是数字信号处理器 (DSP)。程序员用高级语言描述所期望的行为，然后编译到领域专用的处理器上运行。我们已经多次在计算机历史上看到这种事例，而在网络领域里这样的故事才刚刚开始。

在 P4 和新兴的可编程数据包处理硬件（如 PISA）的基础上，P4 语言联盟提出要开发更灵活的交换机，其功能在生产环境中可被指定或改变。程序员可以决定转发平面如何处理数据包，而无须担心实施细节。编译器将命令式的程序转化为表依赖关系图 (table dependence graph)，而表依赖关系图可

以映射到许多具体的目标交换机，包括优化的硬件实现。

我们预测网络在未来几年内将成为一个可编程平台，重现已经发生在计算和存储行业的演化模式。这种趋势将为网络拥有者、经营者和研究人员提供前所未有的灵活性和可视性，为网络控制、管理和服务的创新提供重要的研发机会。

我们大力鼓励研究界为 P4.org 贡献全新的和改进的技术、验证和软件。P4 将很好地帮助研究人员参与数据包处理逻辑的相关研究，而他们无须将工作和一个特定的专用芯片捆绑起来，也无须花费大量精力去学习错综复杂的 API。■

致谢：

感谢白巍、李宇亮、李小舟、陈力、成建晖、李少波在本文中文版的翻译中给予的专业意见。

作　者：

尼克・麦克欧文(Nick McKeown)

斯坦福大学教授。主要研究方向为互联网架构、软件定义网络、可编程转发平面、网络中立性、拥塞控制、计算机科学教育。nickm@stanford.edu

金昶勳(Changhoon Kim)

Barefoot Networks公司系统架构总监。主要研究方向为可编程网络数据平面、网络监控和诊断、网络验证、自编程/配置网络和大规模分布式系统的调试与诊断。chang@barefootnetworks.com

译　者：

高荣新

从事网络芯片开发30余年。目前在Barefoot Networks公司负责亚太事务。ron@barefootnetworks.com

参考文献

[1]P4 Language Consortium. http://p4.org.

[2]P. Bosshart, G. Gibb, H.-S. Kim, and et al.. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN, in ACM SIGCOMM, 2013.

[3]P. Bosshart, D. Daly, G. Gibb, et al.. P4: Programming protocol-independent packet processors, ACM SIGCOMM Computer Communication Review, vol. 44, no. 3, 87~95, 2014.

[4]The P4 Language Specification. http://p4.org/wp-content/uploads/2015/04/p4-latest.pdf, Mar. 2015. Version 1.0.2.

[5]P4 Open Source Repositories. https://github.com/p4lang.

[6]R.Ozdag, White paper- intel ethernet switch fm6000 series - software defined networking, 2012. Intel Corporation.

[7]XPliant packet architecture press release. http://cavium.com/newsevents-Cavium-and-XPliant- Introduce-a-Fully-Programmable-Switch-Silicon- Family.html, 2014.

[8]G. Gibb, G. Varghese, M. Horowitz, and N. McKeown, Design principles for packet parsers, in ANCS, 13~24, 2013.

[9]A. Sivaraman, M. Budiu, A. Cheung, C. Kim, S. Licking, G. Varghese, H. Balakrishnan, M. Alizadeh, and N. McKeown, Packet transactions: High-level programming for line-rate switches. http://arxiv.org/abs/1207.0016, Jan. 2016.

[10]L. Jose, L. Yan, G. Varghese, and N. McKeown, Compiling packet programs to reconfigurable switches, in *12th USENIX Symposium on Networked Systems Design and Implementation* (NSDI 15), 103~115, 2015.

[11]A. Sivaraman, C. Kim, R. Krishnamoorthy, A. Dixit, and M. Budiu, Dc. p4: programming the forwarding plane of a data-center switch, in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, p. 2, ACM, 2015.

[12]C. Kim, Programming the Network Dataplane using P4. http://schd.ws/hosted_files/p4workshop2015/4b/ChangK-P4-Workshop-June-04-2015.pdf, July 2015.

[13]C. Kim, P4 Status Update: Where are we now and what's next? http://schd.ws/hosted_files/2ndp4workshop2015/33/ Chang%2C%20P4%20Workshop%20Nov%2018%202015.pdf, Nov. 2015.

[14]Apache License version 2.0. http://www.apache.org/licenses/LICENSE-2.0.

[15]The P4 Language Specification – pre-release. http://p4.org/wp- content/uploads/2016/03/p4_v1.1.pdf, Jan. 2016. Version 1.1.0.

[16]P4 Tutorial at SIGCOMM'15.http://conferences.

sigcomm.org/sigcomm/2015/tutorial- p4.php.

[17]1st P4 Workshop. http://p4.org/p4/1st-p4-workshop-on-june-4-2015/.

[18]2nd P4 Workshop. http://p4.org/p4/2nd-p4-workshop-on-november-18-2015/.

[19]Mininet – An Instant Virtual Network on Your Laptop. http://mininet.org/.

[20]OpenFlow's Possible Successor P4, Gets Into Hardware. https://www.sdxcentral.com/articles/news/p4- openflows-possible-successor-gets-into-hardware/2015/11/.

[21]P4 and OpenvSwitch. http://p4.org/p4/p4-and-open-vswitch/.

[22]D. Daly, P4 Applied to a vSwitch Data Plane. http://schd.ws/hosted_files/p4workshop2015/67/DanD-P4-Workshop-June-04-2015.pdf.

[23]Improving Network Monitoring and Management with Programmable Data Planes. http://p4.org/p4/inband-network-telemetry/.

[24]C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, In-band network telemetry via programmable dataplanes, in ACM SIGCOMM Industrial Demo, 2015.

[25]Y. Li, R. Miao, C. Kim, and M. Yu, Flowradar: A better netflow for data centers, in 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), (Santa Clara, CA), 311~324, USENIX Association, 2016.

[26]Adding L4 Load-balancing to Every Switch. http://p4.org/wp-content/uploads/2015/12/USC- Barefoot-Demo-Poster.pdf.

[27]D. R. K. Ports, J. Li, V. Liu, N. K. Sharma, and Krishnamurthy, Designing distributed systems using approximate synchrony in data center networks, in 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), (Oakland, CA), 43~57, USENIX Association, 2015.

[28]H. T. Dang, M. Canini, F. Pedone, and et al.. Paxos made switch-y, CoRR, vol. abs/1511.04985, 2015.

[29]X. Li, R. Sethi, M. Kaminsky, D. G. Andersen, and M. J. Freedman, Be fast, cheap and in control with switchkv, in 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), (Santa Clara, CA), 31~44, USENIX Association, 2016.