# DPU驱动的软件与可编程硬件协同UPF架构设计与优化

刘松[1,2]，张琳[1,2]，高铠炜[1,2]，雷玮琛[1,2]，师建新[2,3]，苑新婧[1,2]，蒲凌君[1,2]，张建忠[2,3]
（1.南开大学计算机学院，天津 300350；2.数据与智能系统安全教育部重点实验室，天津 300350；
3.南开大学密码与网络空间安全学院，天津 300350）

**摘　要：**随着移动网络的演进，高带宽、低时延及海量连接的服务对用户平面功能（UPF）提出更高要求。然而，移动流量分布偏斜性与业务异构性导致规则数量激增且粒度细化，引发的规则依赖问题严重制约现有UPF转发性能，且现有依赖消除算法的计算时延过高，难以满足新型的低时延服务要求。为此，提出一种数据处理单元（DPU）驱动的软件与可编程硬件协同UPF架构，该架构基于流量特征识别和软硬件内在服务特点，将大流分离并卸载至可编程硬件加速，CPU侧则执行小流处理，快速生成独立规则以消除规则依赖，同时提出一种规则存储结构与多管线段协同更新机制。在高并发测试环境下，系统吞吐量达97 Gbit/s，端到端转发时延低于500 μs，规则依赖解析计算开销降低65%以上，规则存储与更新效率提升50%。

**关键词：**移动网络；可编程网络；数据处理单元；网内计算；用户面功能
**中图分类号：**TP393.0
**文献标志码：**A
**DOI：**10.11959/j.issn.1000–436x.2026026

# Design and optimization of a DPU-driven software and programmable hardware collaborative architecture for UPF

Liu Song[1,2], Zhang Lin[1,2], Gao Kaiwei[1,2], Lei Weichen[1,2], Shi Jianxin[2,3], Yuan Xinjing[1,2], Pu Lingjun[1,2], Zhang Jianzhong[2,3]
1. College of Computer Science, Nankai University, Tianjin 300350, China
2. Ministry of Education Key Laboratory of Data and Intelligent System Security, Tianjin 300350, China
3. College of Cryptology and Cyber Science, Nankai University, Tianjin 300350, China

**Abstract:** As mobile networks evolve, the user plane function (UPF) faces demands for high bandwidth, low latency, and massive connectivity. However, skewed traffic distribution and diverse services lead to a surge in fine-grained rules. The resulting rule dependencies have become a critical bottleneck for UPF forwarding performance. Moreover, existing algorithms for resolving dependencies suffer from high computational latency and fail to meet strict timing requirements. To address this, a data processing unit (DPU) driven software-hardware hyper UPF architecture was proposed. This architecture separated elephant and mice flows based on traffic features. Elephant flows were offloaded to programmable hardware for acceleration. In contrast, the CPU handled mice flows and employed a fast independent rule generation algorithm to eliminate rule dependencies. Additionally, a rule storage structure and multi-pipe cooperative update mechanism were proposed. Experimental results show that under a high concurrency test environment, the system achieved a throughput of 97 Gbit/s with an end-to-end latency below 500 μs. The computational overhead for rule dependency resolution was reduced by over 65%, with rule storage and update efficiency improved by approximately 50%.

**Keywords:** mobile network, programmable network, data processing unit, in network computing, user plane function

收稿日期：2025–12–17；修回日期：2026–01–27
通信作者：师建新，jxshi@nankai.edu.cn
基金项目：国家自然科学基金资助项目（No.62502239, No.62402247）
**Foundation Items：**The National Natural Science Foundation of China (No.62502239, No.62402247)

## 0　引言

用户面功能（user plane function，UPF）是移动网络中的关键组件，负责承载用户设备（user equipment，UE）与数据网络（data network，DN）之间的数据包转发与流服务质量保证。随着移动网络从 5G 向 6G 演进，UPF 需承载更高带宽、更低时延及更大连接规模的用户流量[1-4]。在这一演进过程中，移动网络的数据流量普遍呈现显著的偏斜性与重尾分布特征[5]，该特征随着云游戏、增强现实、虚拟现实、视频会议和车联网等交互式业务的大规模部署而更加突出。这类应用在带宽需求上高度异构，例如云游戏同时包含大带宽的视频帧流与小带宽的控制信号，增强现实或虚拟现实则由大规模图像数据与小规模环境感知数据共同构成。此外，规则数量也随业务复杂度的提升而持续增长[6-7]。这促使 UPF 需引入更加细粒度的规则集以提供差异化服务质量保障。

传统基于通用服务器的 UPF 具备较强的灵活性，能够支持大量复杂规则，但其转发性能受限于算力与系统调度，在高并发流量场景下难以同时满足高带宽与低时延的性能需求。为突破软件方案的性能瓶颈，学者尝试将 UPF 卸载至可编程网络硬件，实现高吞吐、低时延的数据转发。然而，规则数量增加和粒度细化，导致规则容易出现覆盖或包含关系，例如通配符规则与五元组规则会在匹配空间上发生重叠，迫使可编程硬件需按照优先级顺序逐条匹配。在该匹配机制下，低优先级规则的生效依赖高优先级规则的不命中，进而在规则之间形成优先级依赖[8]。该依赖对不同类型硬件带来不同性能瓶颈，可编程交换机依赖三元内容可寻址存储器进行优先级匹配，当某规则存在依赖关系时，该规则和它的依赖规则必须共同驻留于同一硬件表，使得大量低频小流规则被迫卸载并迅速耗尽有限的片上存储资源；而以英伟达 BlueField 为代表的数据处理单元[9]（data processing unit，DPU）虽然能够利用片上系统中的内存容纳更大规模规则集，但其可编程硬件中的转发管线段支持的优先级深度有限，当依赖过长时，可编程硬件需串联更多管线段以维持正确的匹配语义，导致管线段变长并引起吞吐量下降和转发时延增加[10]。这些规则依赖导致的一系列问题严重制约了可编程硬件加速型 UPF 的性能。

为应对上述性能挑战，现有研究主要从以下几个方面进行了探索。1)基于 CPU 的 UPF 优化：该类方案在保留通用服务器规则处理灵活性的前提下，通过优化 CPU 侧转发路径提升 UPF 性能，例如采用数据平面开发套件（data plane development kit，DPDK）、扩展伯克利数据包过滤器（extended Berkeley packet filter，eBPF）/快速数据路径（express data path，XDP）等旁路内核技术减少协议栈开销[11-15]，或利用元组空间搜索和决策树分类器加速规则匹配[16-17]。然而，在高负载场景下仍需消耗大量 CPU 资源以维持吞吐量，处理时延随负载明显上升，难以满足低时延业务需求。2)基于可编程硬件的 UPF 优化：早期研究将 UPF 的主要功能卸载到可编程交换机和智能网卡中[18-19]；后续研究则进一步优化 UPF 在可编程网络硬件上的规则查询和存储效率[20-21]；一些研究考虑混合架构，通过将 UPF 功能解耦到 CPU 和可编程硬件上，可以兼顾前两类架构的优势[22-23]；还有部分研究尝试将基站与 UPF 协同设计，以进一步降低 5G 网络整体转发时延[24-25]。但上述工作仅关注功能拆分或优化，未从根本上解决规则依赖带来的问题。3)规则依赖解除：该类研究从规则层面缓解依赖问题。早期方法通过规则拆分消除依赖，但导致规则数量膨胀和表项开销增加[26-27]；后续方法主要通过依赖链缩短或规则重写构造无依赖规则集[8,28-31]，但分别面临一致性维护或全局计算开销高的问题，在大规模规则场景下难以满足 UPF 的实时性要求。综上，亟须一种能够以低计算开销生成无依赖规则，并在软件与可编程硬件之间实现高效协同的 UPF 系统设计。

为充分利用可编程硬件的加速能力，软件与可编程硬件高效协同的 UPF 系统面临两方面的需求：1)需要一种软硬件协同高效调度机制，使软件端负责复杂规则管理与依赖处理，硬件端专注于高速转发；2)需要一种能够在大规模规则集下快速生成无依赖规则的轻量化算法，以确保混合架构在实时环境下的可用性。在此背景下，DPU 作为一种集成通用 CPU、大容量内存与可编程硬件的异构平台[32-33]，为上述两项需求的统一实现提供了可能，其板载 ARM CPU 可承担依赖消除与规则生成任务，而可编程硬件用于执行无依赖的高吞吐规则匹配。然而，现有研究尚未系统地探讨如何利用

DPU构建面向UPF的混合架构，更缺乏与其软硬件结构相匹配的快速规则独立化算法。

受文献[31]的启发，本文结合UPF规则特性与实际硬件，设计了一种快速规则依赖消除算法，通过CPU位操作指令对关键计算步骤进行加速，从而在保证规则质量的同时降低计算开销，与现有工作形成互补。为此，本文提出一种DPU驱动的软硬协同分层UPF架构，名为协同架构UPF（collaborative architecture UPF，CA-UPF），旨在面向具有显著流量偏斜特性的移动网络场景，实现UPF在复杂规则依赖条件下的高性能转发。该架构通过合理划分软硬件规则处理职责，充分发挥DPU异构资源的协同优势。首先，软件端基于DPDK在DPU的CPU实现高性能数据包处理流水线设计，并开发了快速独立规则生成算法，以降低硬件优先级深度需求。在可编程硬件端，结合BlueField-2 DPU的管线特性，设计优化了UPF转发管线和规则存储格式，以提升规则容纳能力与整体转发性能。本文的主要贡献如下。

1)提出CA-UPF。该架构充分利用DPU板载ARM CPU与可编程硬件的协同能力，将CPU在规则存储与灵活控制方面的优势与可编程硬件在高吞吐、低时延转发方面的能力相结合。CA-UPF已在具有代表性的商用DPU平台NVIDIA BlueField-2上完成实现与实验验证。

2)设计了一种快速独立规则生成算法，用于消除规则优先级依赖，降低可编程硬件对优先级深度的需求并提升转发效率。在此基础上，结合BlueField-2 DPU的特性，设计并优化了UPF转发管线、规则存储格式和规则更新策略，提高了硬件规则容纳能力并降低表项更新开销。

3)实验结果表明，在包含40万个流、具有流量偏斜与复杂规则依赖的实验场景下，CA-UPF可实现最高97 Gbit/s的系统吞吐量，大流端到端转发时延稳定低于500 μs；同时，所提算法将规则依赖解析的计算开销降低了65%，规则存储和更新效率提高约50%，有效提升了系统在复杂规则条件下的扩展性与动态更新能力。

## 1 背景与动机

### 1.1 UPF和规则依赖

UPF是核心网络中承担用户面流量处理的关键网元，它负责在接入网与DN之间转发UE的数据包，执行数据包分类、服务质量策略的实施，并报告流量使用情况。UPF由控制面配置的规则来实现这些功能，例如，数据包检测规则（packet detection rule，PDR）用于识别数据包，由于不同PDR之间可能在匹配字段上存在重叠，第三代合作伙伴计划（3rd Generation Partnership Project，3GPP）标准通过引入规则优先级来消除匹配歧义，确保每个数据包仅命中唯一的PDR；转发动作规则（forwarding action rule，FAR）用于确定数据包的路由；服务质量保证规则（quality of service enforcement rule，QER）用于服务质量的控制；使用报告规则（usage reporting rule，URR）用于监控流量，并且当UE离线时，通过缓冲动作规则（buffering action rule，BAR）来对数据包进行缓冲。上述规则决定了UPF的转发行为，规则的执行效率直接影响核心网络的吞吐性能、时延水平和服务质量保障能力，因此，对UPF的性能进行优化具有重要意义。

基于CPU的UPF依托充足的内存资源，可以维护带优先级的大规模规则集合；而在基于可编程硬件的UPF中，规则优先级的语义可能会导致硬件性能下降。可编程硬件为保证与标准定义在逻辑语义上的一致性，必须严格遵循高优先级规则优先匹配的原则，但该原则容易导致规则依赖问题。如图1所示，隧道端点标识符（tunnel endpoint identifier，TEID）为GPRS隧道协议用户面（GPRS tunnelling protocol user plane，GTP-U）用于区分不同承载/会话的标识字段，本文示例中PDR表项基于TEID与目的IP前缀等字段进行匹配。当低优先级规则PDR 6在匹配字段上被若干高优先级通配符规则PDR 1~5所覆盖时，可编程硬件无法仅部署PDR 6而保证匹配结果的正确性。这是因为PDR 6与PDR 1~5的匹配空间存在重叠，若硬件中缺失这些高优先级规则，原本应命中PDR 1~5的数据包将无法被正确捕获，进而被错误地匹配至低优先级规则PDR 6。为避免上述错误匹配，所有与之存在覆盖关系的高优先级规则必须同时加载至硬件，并保持其原有的优先级顺序。规则依赖在不同硬件平台上表现出不同后果。在可编程交换机中，相关规则必须作为依赖集合同时部署，导致部分低命中率的高优先级规则长期占用有限的硬件表项，降

低了规则存储效率；在DPU中，为维持优先级匹配语义，规则往往需要跨多个转发阶段协同处理，从而拉长转发路径并限制规则管理的灵活性。上述问题共同削弱了可编程硬件对UPF转发处理的加速效果。

待匹配数据包
TEID:3
源IP:10.53.0.1
目的IP:192.168.1.100

Hit

| PDR ID | 优先级 | TEID | 目的IP地址 | FAR ID | QER ID |
|---|---|---|---|---|---|
| 1 | 4 | 3 | 192.168.3.0/255.255.255.32 | 12 | 5 |
| 2 | 4 | 3 | 192.168.2.16/255.255.255.16 | 11 | 4 |
| 3 | 4 | 3 | 192.168.4.100/255.255.255.255 | 18 | 3 |
| 4 | 13 | 3 | 192.168.2.64/255.255.255.192 | 13 | 7 |
| 5 | 13 | 3 | 192.168.2.80/255.255.255.240 | 15 | 4 |
| 6 | 127 | 3 | 192.168.0.0/255.255.0.0 | 1 | 1 |

图1　UPF中规则依赖的例子

### 1.2　规则依赖对DPU的影响分析

以NVIDIA BlueField为代表的DPU为软硬协同UPF提供了可编程的数据平面执行能力。然而，在UPF规则数量膨胀和优先级语义并存的场景下，规则依赖问题在DPU的可编程硬件中被进一步放大，并具体体现为硬件层面的性能约束。DPU通过数据中心芯片架构（data center-on-a-chip architecture，DOCA）对可编程硬件进行管线构建[9]，其管线由多个管线段串联构成。在规则依赖场景下，DPU的可编程硬件面临多项结构限制。

1) 规则依赖会导致管线被迫加深。单个管线段能支持的优先级层级有限，而UPF的PDR可包含多达256个优先级。当规则之间存在较长的依赖链时，系统无法在单个管线段内完成匹配，只能将多个管线段串联使用。随着管线段数量的增加，数据包需要经过更多处理阶段，处理时延明显上升，吞吐量随之下降。本文对此进行了实验评估，其结果如图2(a)所示，当管线段数量达到60条时，系统吞吐量下降超过50%，平均时延超过300 μs。

2) 规则依赖会放大定宽规则存储结构带来的空间浪费。在DPU的可编程硬件中，每条规则均以定宽表项的形式存储，表项中定义了常见的匹配字段和动作参数。这种设计有利于保证流水线处理的通用性与访问效率，但也意味着单条规则实际使用的字段越少，未被利用的空间越多。当规则依赖迫使多条相关规则必须同时驻留于硬件中时，规则总量的增加会进一步放大定宽分配所带来的空间浪费。

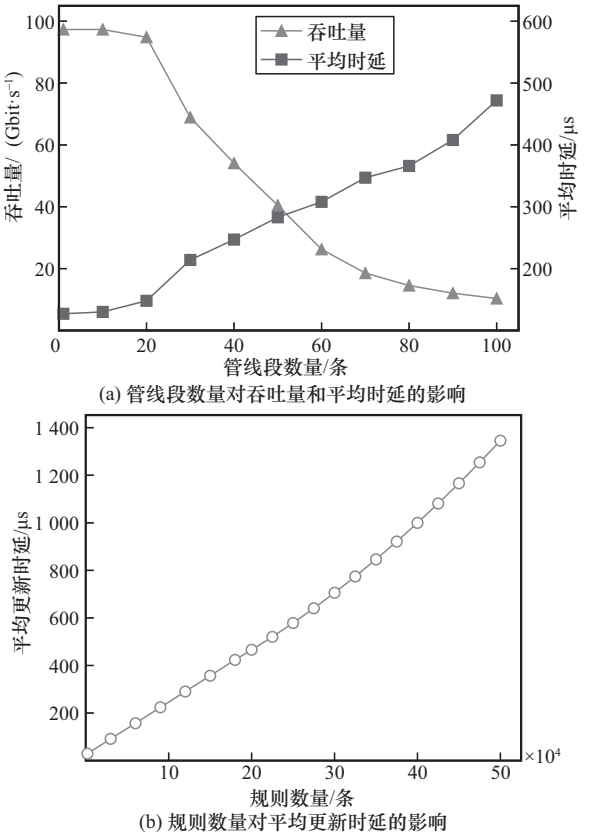



图2　管线段数量和管线段中流规则数量对系统性能的影响

3)规则依赖会加剧规则更新过程开销。UPF规则具有高度动态性，频繁的会话建立与释放、服务质量策略调整等都会通过DOCA向可编程硬件下发更新指令。对于支持掩码和优先级的管线段，DOCA为了保证流水线的一致性，在执行规则插入、删除或修改时会进行同步操作。实验结果如图2(b)所示，当规则数量较少时，更新时延基本保持在较低水平；但随着规则数量的增加，插入或删除的平均时延急剧上升，并伴随着瞬时吞吐量下降。当规则依赖导致硬件中驻留规则数量增加后，单次更新所需的同步开销随之放大，并更容易在更新过程中引发瞬时吞吐量下降。

综上所述，优先级容量有限、存储结构固定和更新同步成本高，使得规则依赖链在DPU上被放大为优先级、存储与更新3个方面的核心瓶颈，对构建软硬协同UPF架构设计提出巨大挑战。

## 2　系统设计与实现

### 2.1　系统结构

为实现CA-UPF，本文充分利用DPU的体系结构特性以及移动业务流量的内在规律，系统结构如

图 3 所示。移动网络流量普遍呈现偏斜分布特征，少量大流占据主要带宽资源，而大量小流带宽需求低但数量庞大。基于该特性，CA-UPF 采用大流与小流分离处理的体系结构：将少量但带宽占用高的大流卸载至 DPU 的可编程硬件所在的数据平面进行加速转发，以获得高吞吐与低时延；将数量庞大但带宽占用低的小流留在 CPU 所在的控制平面处理，以利用其在复杂规则管理、规则重构以及规则动态更新方面的优势。通过上述设计，CA-UPF 在满足性能需求的同时，降低了优先级依赖规则对可编程硬件的性能影响，提高了可编程硬件的有效规则容量与转发效率。由此形成以控制平面为决策核心、数据平面为高速执行单元的协同架构，下文将分别从控制平面与数据平面的角度，介绍 CA-UPF 的关键模块设计与实现。

1) 控制平面设计。图 3 右侧展示的控制平面运行在 DPU 板载 ARM CPU 中，负责小流转发、流识别、规则管理以及与核心网的交互。其中，高速外设互连（peripheral component interconnect express，PCIe）总线交换机用于连接 ARM CPU 和可编程硬件，支撑控制平面与数据平面之间的数据交换。控制平面由分组转发控制协议（packet forwarding control protocol，PFCP）服务器、数据包解析器、规则查询器、流分类器、独立规则生成器以及数据包重组器等模块组成。PFCP 服务器与核心网交互，接收、解析并维护来自会话管理功能（session management function，SMF）的控制指令，完成规则下发与状态同步，这是 UPF 的核心功能之一。为了高效处理来自数据平面的数据包，控制平面集成了数据包解析器、规则查询器和数据包重组器。数据包解析器基于 DPDK 在用户态实现，可快速提取数据包头部信息，以降低时延和 CPU 开销。规则查询器采用优化的哈希结构实现高速查找，并根据匹配结果执行相应动作，如封装、解封装、转发、缓存或丢弃等操作。数据包重组器根据匹配结果更新相关头字段，并使用 DPDK 将处理后的数据包发送回数据平面的出口域。为确定某条流是否应被卸载至可编程硬件，CA-UPF 引入了流分类器。流分类器通过计数最小草图进行实时流量监测，以极低的内存开销高效地将流划分为大流与小流。被识别为大流的流将被卸载到数据平面，以利用硬件加速能力。为了避免数据平面中的规则优先级导致性能下降，独立规则生成器会首先检查待卸载规则与现有规则之间是否存在优先级依赖。如果存在此类依赖，则采用快速独立规则生成算法生成一个语义等价且不含优先级依赖的规则，然后通过规则管理器将其卸载到可编程硬件；若未检测到依赖关系，则直接卸载原始规则。

2) 数据平面设计。如图 3 左侧所示，数据平面运行在 DPU 的可编程硬件上，负责高效的规则匹

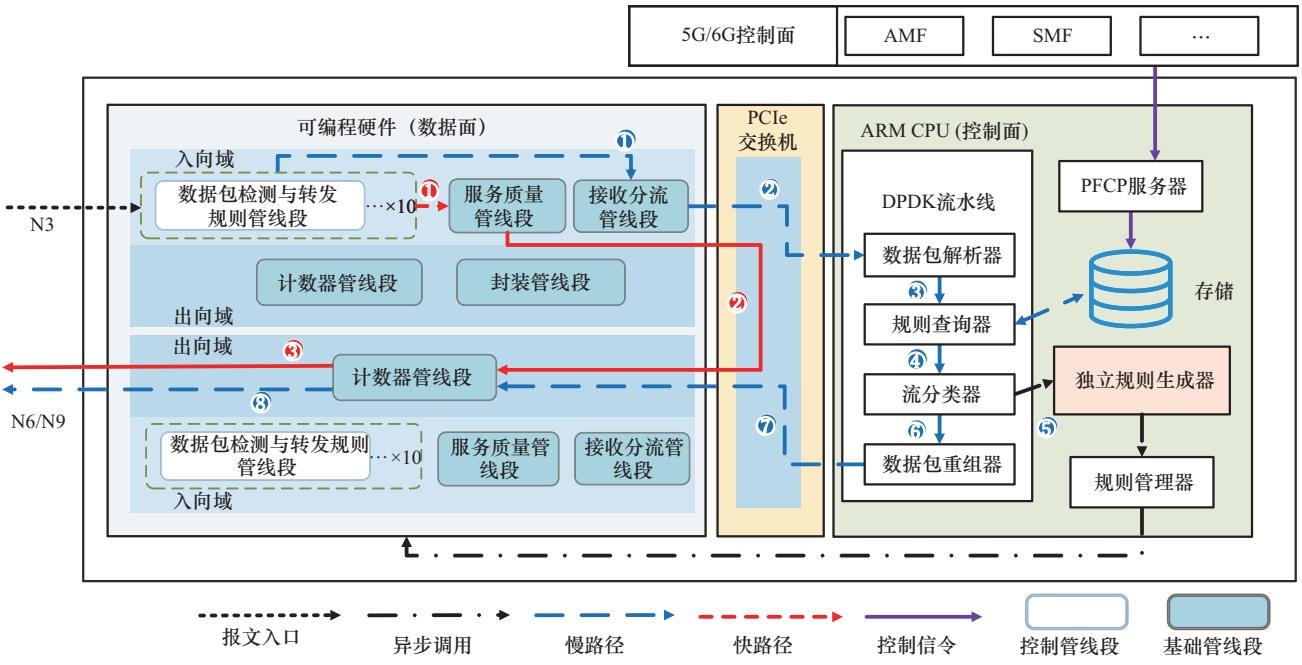



图3　系统结构

配与动作执行，加速 PDR、FAR、QER 和 URR 的处理。数据平面基于 NVIDIA 的 DOCA Flow 框架实现，该框架支持灵活构建数据包处理流水线，并通过级联组合的方式定义匹配条件、监测方式与执行动作，从而实现高性能转发。根据 DOCA Flow 框架要求，CA-UPF 的数据面被划分为入口与出口两个域，每个域由若干面向特定任务的专用流水线组成。图中展示的是从 UE 到 DN 的上行转发路径；下行路径具有相同的结构。入向域集成了数据包检测与转发规则（PDFR）管线段、服务质量管线段以及接收分流管线段；出向域则包括计数器管线段和封装管线段。其中，数据包检测与转发规则管线段是核心模块，用于联合执行 PDR 与 FAR 的匹配与动作。PDR 规则负责检查数据包头部字段，而 FAR 规则定义转发或数据包修改操作。

在同一管线段内合并 PDR 与 FAR 的匹配查找能减少冗余存储并提升硬件利用率，因为 DOCA Flow 中的规则采用定宽表项格式。由于每条 PDR 都对应一条 FAR，将二者合并能够避免规则重复，提高硬件存储效率。数据包检测与转发规则管线段还包含一个支持通配符规则卸载的控制管线段，以满足灵活的匹配需求。服务质量管线段依据 QER 执行限速与颜色标记，以实现差异化的服务保障。未命中任何 PDR 和 FAR 的数据包将通过接收端分流管线段重定向至控制平面，以便进一步分析并触发规则更新。在出口域中，计数器管线段维护细粒度的流量统计信息，用于监测与计费；封装管线段将下行数据包重新封装为 GTP-U 格式，确保协议合规并将数据包正确送达用户设备。

### 2.2　协同工作流程设计

如图 3 所示，CA-UPF 的工作流程由两条协同处理路径组成：短虚线标注表示快速路径，点线标注表示慢速路径。快速路径完全在可编程硬件上执行，用于处理那些规则已经卸载的大流；慢速路径则运行在 DPU 板载 ARM CPU 中，用于处理小流或未命中规则的流。下面以上行方向为例，对该过程进行详细说明。

1) 快速路径。①当来自 UE 的上行数据包抵达 N3 接口后，首先进入可编程硬件数据平面，由数据包检测与转发规则管线段进行处理，如果命中相应的 PDR 和 FAR，硬件将在同一流水线上完成所有相关动作，包括头部检查、GTP-U 解封装以及转发决策；②数据包进入服务质量管线段，根据预设的服务质量策略执行限速与流量标记；③数据包依次通过计数器管线段进行流量统计，并通过封装管线重新封装为 GTP-U 格式，然后经由 N6/N9 接口转发至 DN。整个过程完全在可编程硬件中完成，不需要经过 CPU，从而为已建立的流提供确定性的低时延和线速转发能力。

2) 慢速路径。①当数据包在数据包检测与转发规则管线段中未命中任何已有规则时，它将通过接收分流管线段被重定向至控制平面；②在控制平面中，由 DPDK 实现的数据包解析器在用户态提取关键头部字段，以便后续处理；③规则查找器在本地规则表中进行匹配，若找到对应规则，则按照指定动作处理该数据包，否则进入下一阶段；④流分类器基于计数最小草图的估计方法分析该流的流量特征，以判断其属于大流还是小流；⑤对于需要硬件加速的大流，规则管理器会检查其与现有规则之间的依赖关系，并采用快速独立规则生成算法构造语义等价且无冲突的规则，该模块的所有操作以异步方式执行，以尽可能降低额外时延，生成的新规则随后通过 PCIe 接口卸载至可编程硬件；⑥数据包重组器对数据包进行重构，使其重新进入数据平面；⑦通过计数器管线段完成流量测量；⑧经 N6/N9 接口发送至 DN。

通过上述分层协同设计，CA-UPF 实现了紧密的软硬件协同：快速路径为已建立的会话提供确定性的高速转发，而慢速路径则为新流或未匹配的流提供动态规则生成与自适应卸载能力，从而在 6G 边缘网络环境下同时兼顾高性能与高灵活性。

## 3　关键问题和方法

为了使 CA-UPF 高效运行，还需解决若干实际挑战，包括 DPU 硬件对优先级的支持受限、内存利用效率低以及动态更新带来的额外开销等问题。为此，本文提出了 3 项关键优化设计：快速独立规则生成算法、规则存储优化以及规则更新优化机制。

### 3.1　快速独立规则生成算法

在 UPF 的规则卸载与加速过程中，如何在存在复杂规则依赖关系的情况下生成可直接下发至可编程硬件的独立规则，是影响系统性能与实时性的关键问题之一。本文围绕这一需求，研究独立规则

生成问题。设规则集合为rules，其中目标流对应的最优匹配规则为best_match，由其生成的独立规则记为best_match′。算法的目标是在保持转发语义完全一致的前提下，构造一条与best_match等价的独立规则best_match′，使其在不需要任何依赖规则的情况下正确匹配目标流。best_match′生成过程可进一步抽象为最小冲突位覆盖问题，其中，冲突位指的是在规则匹配字段中导致不同规则产生匹配歧义的比特位置，而覆盖则表示通过对这些比特位进行具体化，使生成规则在匹配空间上能够与所有依赖规则形成有效区分，从而彻底消除对高优先级规则的依赖。该问题可被归约为经典的集合覆盖问题，在计算复杂性上属于NP难问题[34]。这表明，在规则规模较大或依赖关系复杂的情况下，求解最优解在计算开销上难以满足UPF对实时性的要求，因此，独立规则生成算法的设计必须在计算复杂度与生成质量之间进行权衡。具体而言，算法需满足3个条件：首先是语义等价性，best_match′必须与best_match在转发行为上保持一致；其次是低计算开销，算法需要具备较低的求解时间，以满足用户会话操作的时延要求；最后是匹配范围最大化，best_match′应尽可能保留原有规则best_match的匹配范围，避免因过度具体化导致匹配空间收缩，从而引发规则更新频次的增加。

#### 3.1.1 核心思想

围绕上述约束，本文提出了一种快速独立规则生成算法，在规则依赖消除过程中实现计算开销与质量之间的平衡。在处理此类问题时，通常将规则间的潜在冲突关系建模为具体化矩阵，其中每一行对应一条依赖规则，每一列则对应匹配字段中的一个比特维度。现有方法[31]采用以列为中心的贪婪搜索策略。该策略通过对具体化矩阵进行纵向遍历，在每一步迭代中计算各列的覆盖权重，并选择能够区分最多冲突规则的比特位作为具体化目标。尽管该策略在理论上能够获得较高质量的冲突位选择结果，但其计算过程依赖于对全量矩阵的频繁扫描与状态更新，导致较高的时间开销，难以适应UPF场景下的实时性要求。

为降低计算成本，本文算法将搜索视角重构为以行为中心的策略，即以单条规则为基本单位分析其潜在冲突关系。由于PDR中支持通配符匹配的字段数量有限，主要集中于IP地址和端口号，规则冲突仅涉及有限的匹配维度，因此可以用有限个比特刻画其冲突关系。在UPF场景下，规则冲突由一个IP地址及源端口和目的端口共同决定，上行PDR关注目的IP，下行PDR关注源IP，其余端口字段保持一致，总宽度为64 bit，可直接映射至单个64 bit CPU寄存器。基于该表示，冲突位的定位可直接调用CPU的位操作指令完成。具体而言，使用计数尾随零指令可在常数时间内定位差异掩码中最低有效的置位比特，即最低位的差异比特，从而以硬件级位操作替代高开销的串行比对，降低基础运算的指令周期开销。

然而，朴素的以行为中心的策略缺乏对全局冲突位分布的感知，容易选择区分能力较弱的边缘比特，进而导致规则过度具体化。以网络切片隔离场景为例，假设存在一组高优先级依赖规则，其IP地址集中在10.0.0.0/9网段，而目标流属于另一业务切片，IP地址为10.128.0.100。在二进制表示下，干扰规则与目标流在第8位比特上存在全局性的固定差异，即干扰规则该位恒为0，而目标流该位恒为1。这一比特位能够同时区分目标流与所有干扰规则，是一个可以一举消除全部依赖关系的全局关键特征位。然而，由于IP地址低位主机号的随机性，依赖规则在若干低位比特上也与目标流存在零星差异。朴素行中心策略遵循最低位优先的选择原则，在逐行扫描时极易优先选中这些低位差异。尽管该过程在形式上消除了规则依赖，但生成的独立规则被迫固定大量无关的主机号比特，最终退化为仅能匹配单一主机的高度具体化规则，削弱了对目标业务切片网段的通配能力。

基于这一观察，本文在行中心处理框架内进一步引入轻量级统计采样机制，利用网络流规则冲突分布普遍存在的长尾特性，对比特位的全局区分能力进行估计，从而优先选择关键特征位。在此基础上，结合自适应早停与搜索空间剪枝策略，该算法在保持线性计算复杂度的同时，在规则生成速度与规则压缩效果之间取得了更优的折中。

#### 3.1.2 算法设计

本文设计了一种快速独立规则生成算法，该算法依次经历3个阶段：依赖规则识别、具体化矩阵构建和冲突位选择，最后生成独立规则。

快速独立规则生成算法首先需要识别所有与best_match.mask存在掩码重叠关系的依赖规则。对

于任意规则$r \in$ rules，若其满足以下两个条件

$$r.\text{priority} > \text{best\_match.priority} \tag{1}$$

$$(r.\text{ip} \wedge \text{best\_match.mask}) = (\text{best\_match.ip} \wedge \text{best\_match.mask}) \tag{2}$$

则认为规则$r$与最优匹配规则best_match存在优先级依赖关系。其中，$r$.priority表示规则$r$的优先级；best_match.priority表示最优匹配规则的优先级；$r$.ip和best_match.ip分别表示规则$r$和最优匹配规则的IP地址；best_match.mask表示最优匹配规则的掩码；∧表示按位与操作。式(1)说明规则$r$的优先级更高，因此可能覆盖best_match。式(2)表示两条规则在best_match的掩码范围内匹配到相同的流量，存在匹配域重叠。两个条件同时满足时，$r$会影响best_match的生效，从而构成优先级依赖。该步骤对应算法1，在给定掩码范围内，收集所有与best_match存在匹配重叠且优先级更高的规则，构建依赖规则集合。

**算法1**　依赖规则识别算法

**输入**　rules、best_match // 所有规则、最佳匹配规则

**输出**　$D$ // 依赖规则集合

1) dep_count ← 0// 初始化依赖计数器，用于索引依赖规则列表
2) $D$ ←[ ]//初始化空列表，用于存储与best_match冲突的高优先级规则
3) for $r \in$ rules do
4)　　if $r \neq$ best_match and $r$.priority > best_match.priority then
5)　　// 判断规则$r$是否与best_match存在冲突
6)　　conflict ← (best_match.ip ∧ best_match.mask)== ($r$.ip ∧ best_match.mask)
7)　　if conflict then
8)　　// 将冲突且优先级更高的规则加入依赖规则列表
9)　　$D$[ dep_count ] ← $r$
10)　　end if
11)　end if
12) end for
13) return $D$

在识别出所有依赖规则后，算法构建具体化矩阵，用于表示规则之间潜在的比特级冲突。构建该矩阵的核心意义在于：它能够精确定位冲突比特，从而有效实现最优匹配规则与更高优先级依赖规则的分离。具体而言，设依赖规则集合为$D = \{\text{dep}_0, \text{dep}_1, \cdots, \text{dep}_{m-1}\}$，其中$m = |D|$为依赖规则总数，对于每条依赖规则dep，矩阵中的对应元素按式(3)计算。

$$\text{diff} = (\text{target} \oplus \text{dep.value}) \wedge \text{dep.mask} \wedge (\sim \text{best\_match.mask}) \tag{3}$$

其中，⊕表示按位异或，∧表示按位与，~表示按位取反，target代表目标流的所有匹配项所拼接的比特串，dep.value代表依赖规则dep的规则比特串，dep.mask代表依赖规则dep的掩码。式(3)用于计算diff，其指出了必须对best_match的哪些比特位进行具体化，才能将其与依赖规则dep区分开来。具体来说，首先，使用异或操作找出两者的原始差异；其次，结合dep.mask排除掉依赖规则本身不关心的位，保留有效匹配位；最后，将结果与最优匹配规则掩码的反码相与。这一步确保了只筛选出那些在最优匹配规则中原本为通配，但与依赖规则发生冲突的比特位。上述步骤对应算法2。

**算法2**　具体化矩阵构建算法

**输入**　best_match、target、$D$ // 最佳匹配规则、目标流的所有匹配项、依赖规则集合

**输出**　$\boldsymbol{C}$ // 具体化矩阵

1) dont_care ← ~best_match.mask // 计算无关位：即best_match的掩码中为0的位
2) Initialize concrete_matrix // 初始化具体化矩阵，该矩阵用于存储每个依赖规则的差异位
3) for $i \in |D|$ do
4)　　// 计算目标流量target与当前依赖规则的差异
5)　　diff ← (target ⊕ $\text{dep}_i$.value) ∧ $\text{dep}_i$.mask ∧ dont_care
6)　　$C$[ $i$ ] ← diff
7) end for
8) return $\boldsymbol{C}$

在具体化矩阵构建完成后，算法需要进一步执行冲突位选择，以构造能够将最优匹配规则与所有依赖规则区分开的最小比特集合。定义具体化矩阵

记为 $\boldsymbol{C} \in \{0,1\}^{m \times 64}$，其第 $i$ 行 $C[i]$ 表示规则 $\mathrm{dep}_i$ 与 best_match 之间的冲突位掩码。所选比特应同时满足两个基本要求，一方面，能够完全消除与依赖规则之间的匹配歧义；另一方面，在保证区分正确性的前提下，尽量保持原始规则的通配范围，避免因过度具体化导致匹配空间缩小。

为此，本文在以行为中心的思想下提出一种轻量级的全局感知机制。其核心思想在于，不同冲突比特在规则集合中的分布具有明显差异，具有较强区分能力的比特通常在多条依赖规则中重复出现，而较弱区分能力的比特则仅在少数规则中出现。基于这一特性，算法通过采样方式对冲突比特的整体分布进行刻画，并据此指导后续的比特选择过程。具体而言，算法从具体化矩阵 $\boldsymbol{C}$ 中选取前 $l$ 条依赖规则进行采样，在采样过程中，统计每个比特位 $b$ 的出现频次，记为

$$\mathrm{freq}[b] = \left|\left\{ i \in [0,l) \,\middle|\, \{(C[i] \gg b) \wedge 1\} = 1 \right\}\right| \tag{4}$$

并以此构建 64 bit 的全局掩码。

$$\mathrm{global\_mask} = \vee_{i=0}^{l-1} C[i] \tag{5}$$

其中，$\vee$ 表示按位或运算，该掩码用于约束后续搜索空间，使算法优先关注在多条规则中具有较强区分能力的候选比特，从而降低对较弱区分能力比特的影响。

完成采样后，算法进入执行阶段。维护一个已选比特集合 $S$，对于每条尚未被区分的依赖规则 $\mathrm{dep}_i$，即满足 $C[i] \cap S$ 不为空的依赖规则 $\mathrm{dep}_i$，算法在其候选集 $C[i] \cap \mathrm{global\_mask}$ 中选择新的具体化比特位。若 $C[i] \cap S$ 为空，则候选集退化为 $C[i]$。在候选集中，优先选取出现频次最高的候选比特位，也就是 $\mathrm{freq}[b]$ 最大的比特位 $b$，若存在多个候选比特，则选择编号较小者以保证结果的确定性。

在实现层面，冲突比特的定位依赖 CPU 提供的位操作指令完成。通过对差异掩码执行计数尾随零操作，算法能够在常数时间内定位最低有效的比特位，即

$$b = \_\_\mathrm{builtin\_ctz}(C[i]) \tag{6}$$

从而以硬件指令替代基于循环的位操作，降低运算的时间开销。

综上所述，该冲突位选择策略在保持时间开销的同时，通过轻量级采样实现了对全局冲突模式的有效感知，避免了朴素行中心策略因盲目选择低位噪声而导致的过度具体化问题，冲突位选择的整体流程如算法 3 所示。

**算法 3** 冲突位选择算法

**输入** $\boldsymbol{C}$、$m$、$W$、$l$ // 具体化矩阵、依赖规则条数（$\boldsymbol{C}$ 的行数）、冲突位向量位宽（在此为 64）、采样上限

**输出** $S$ // 选中的冲突位集合

1) // 阶段 1：自适应采样，识别高价值冲突位
2) for $b$ = 0 to $w$ − 1 do
3) freq[$b$] ← 0 // 初始化各比特位的冲突频率
4) end for
5) global_mask ← 0 // 记录采样中出现过的所有冲突位
6) sample_$n$ ← min($l$,$m$) // 限制采样规模以平衡开销与代表性
7) for $i$ = 1 to sample_n do
8) $x$ ← $C[i]$ // 获取第 $i$ 条依赖规则的冲突特征
9) global_mask ← global_mask ∨ $x$ // 聚合采样规则的冲突位，形成全局有效候选空间
10) while $x \neq 0$ do
11) $b$ ← __builtin_ctz($x$) // 返回 $x$ 最低位 1 的索引
12) freq[$b$] ← freq[$b$] + 1 // 统计该位在采样规则中的冲突频次
13) $x \leftarrow x \wedge (x-1)$ // 遍历所有置位
14) end while
15) end for
16) // 阶段 2：贪心覆盖，最小化冲突位数量
17) $S$ ← 0// 初始化选中冲突位集合；目标是用最少位覆盖所有依赖规则
18) for $i$ = 0 to $m$ do// 遍历全部依赖规则
19) $\mathrm{dep}_i \leftarrow C[i]$ // 当前规则的冲突特征
20) if ($\mathrm{dep}_i \wedge S$) == 0 then
21) candidates ← $\mathrm{dep}_i \wedge$ global_mask //优先从高价值候选位中选择
22) if candidates == 0 then
23) candidates ← $\mathrm{dep}_i$ // 若无高价值候选，则使用本规则自

身冲突位
24)　　end if
25)　　$b_{star} \leftarrow -1$ // 初始化最优冲突位
26)　　$best_{freq} \leftarrow -1$ // 对应最高频率
27)　　$x \leftarrow$ candidates
28)　　while $x \neq 0$ do
29)　　$b \leftarrow$ __builtin_ctz($x$)
30)　　if $b_{star} == -1$ or freq[$b$]> $best_{freq}$ or (freq[$b$]==$best_{freq}$ and $b < b_{star}$)then
31)// 优先选择高频冲突位，频率相同时选择索引更小的位
32)　　$b_{star} \leftarrow b$
33)　　$best_{freq} \leftarrow$ freq[$b$]
34)　　end if
35)　　$x \leftarrow x \wedge (x - 1)$
36)　　end while
37)　　$S \leftarrow S \vee (1 << b_{star})$ // 将选中的冲突位加入集合
38)　　end if
39) end for
40) return $S$ // 返回最小冲突位集合

在全部冲突比特位确定之后，以最优匹配规则 best_match 为基础，对冲突比特集合 $S$ 对应的比特位进行具体化，得到独立规则 best_match′。该独立规则在转发语义上保持一致，其匹配空间已与所有依赖规则完全区分，因此在卸载可编程硬件时，不需要再卸载任何依赖规则即可正常工作。

### 3.2　规则存储和更新优化

1) 规则存储优化。为了进一步提升硬件空间利用率，CA-UPF 在 DOCA Flow 框架下重构了 UPF 的规则组织方式。在朴素实现中，PDR 和 FAR 被映射到不同的管线中，每条规则在硬件中占据一个定宽结构体。PDR 和 FAR 共享多个常用匹配字段，这种分离式结构会导致字段存储冗余和流水线级联增加，从而带来较大的空间效率问题。为解决这一问题，CA-UPF 将每条 PDR 与其对应的 FAR 合并为一条统一的规则项，称为数据包检测与转发规则。在统一的结构体中，匹配字段与动作参数被共同存储，使匹配与转发能够在同一流水线上完成。在下行方向中，每条 PDR 必须关联一个唯一的 FAR，且 FAR 包含与每个 UE 或每个应用相关的 TEID 字段。通过合并 PDR 与 FAR，CA-UPF 可以在单一流水线中完成匹配与封装处理，使规则项数量减少约一半。相比之下，在上行方向中，由于 FAR 不依赖 TEID 且其转发动作较为简单，空间节省效果不如下行。需要指出的是，CA-UPF 并未合并其他类型的规则，例如 QER 和 URR，因为它们与 PDR 并不存在一一对应关系，一条 PDR 可能关联多个 QER 或 URR，因此不适合进行合并。总体来看，PDR-FAR 合并设计在不改变 UPF 处理语义的前提下，有效提升了硬件空间利用效率。

2) 规则更新优化。根据动机实验，DPU 上单条规则的更新时延几乎随单个管线段中表项数量线性增长。当表规模较小时，一次更新可在数十微秒内完成；而当表项达到约 20 万条时，平均更新时延已上升至约 0.5 ms，进一步导致全局刷新变慢，甚至引发控制面与数据面的短暂不一致。为解决这一问题，CA-UPF 采用多管线段协同机制，将 PDFR 规则均衡分布在多个管线段上，并将每个管线段的表项数量限制在设定阈值以下，在本文的部署中，该阈值为 100 000。在这一运行点下，平均更新时延约为 0.25 ms（约降低至一半）。当管线段级联数量不超过十个时，总体吞吐量仍能接近单管线段基线，同时实现低时延与高并发更新能力。

在多管线段布局上，CA-UPF 进一步引入随机选择策略，以常数级开销将新规则分配到负载较低的管线段。对于一个以五元组和 TEID 为关键字的规则 $k$，控制面使用两个独立哈希函数 $H_1$ 和 $H_2$ 生成两个独立哈希值以获得两个候选管线段。

$$p_1 = H_1(k) \bmod N, \quad p_2 = H_2(k) \bmod N \tag{7}$$

其中，$p_1$ 和 $p_2$ 是两个由哈希函数得出的初始候选管线段索引，$N$ 为并行管线段的数量。由于负载均衡选择需要两个不同的候选管线段，为避免出现 $p_2 = p_1$ 的哈希冲突，CA-UPF 通过以下规则构造最终的第二候选管线段 $p_0$。

$$p_0 \leftarrow \begin{cases} (p_1 + 1) \bmod N, & p_2 = p_1 \\ p_2, & \text{其他} \end{cases} \tag{8}$$

当 $p_2 = p_1$ 的哈希冲突出现时，将第二候选设为相邻管线段，其索引为 $(p_1 + 1) \bmod N$。若无冲突，则直接采用 $p_2$。

随后，控制器读取两个候选管线段的瞬时负载$L(p)$，即第$p$号管线段当前的规则数量。规则最终被安装到负载较轻的管线段。一旦规则安装完成，该管线段将继续处理该流后续的所有数据包，不需要控制面再次介入。该策略优点在于，仅探测两个候选点即可降低热点风险，并在负载均衡效果上接近最优，而不需要全局扫描或集中式协调，从而可快速响应流量突发和偏斜情况。

## 4 实验评估

### 4.1 实验环境设置

为了验证CA-UPF架构在真实高并发网络环境下的性能表现，本文搭建了基于NVIDIA BlueField-2DPU的硬件测试平台。实验拓扑由两台高性能服务器通过100 Gbit/s光纤直连构成。其中一台作为流量生成器，配备双路Intel Xeon Gold 5332RCPU，运行Pktgen[35]发包工具以模拟高并发、线速的用户流量；另一台作为被测设备，搭载Intel Xeon Gold 5318YCPU及BlueField-2 DPU 100 GB，部署CA-UPF原型系统。软件环境均采用Ubuntu 22.04操作系统及NVIDIA DOCA 2.10开发框架。

针对移动网络流量显著的重尾分布特征与日益复杂的QoS策略需求，本文构建了一个包含40万条并发流的合成数据集，旨在模拟1万个UE的高并发接入场景。每个UE分配唯一的IP地址，并扩展出40个IPv4五元组连接以模拟多业务并行会话。具体的流量与规则参数设置如下。

1) 流量偏斜度设置。为了模拟流量偏斜特性，流量被划分为两类。大流占比约10%，用于模拟虚拟现实/增强现实、高清视频等高带宽业务，占据总流量负载的90%。在CA-UPF中，此类流量是硬件卸载的主要对象。小流占比约90%，主要用于模拟物联网设备状态上报、网络语音通话及即时通信等低带宽业务。同时，大小流的数据包大小遵循广泛使用的5G UPF测试模型[36]，范围为64~1 518 B，平均为690 B。

2) 规则集规模与依赖性配置。为了复现规则依赖导致的规则遮蔽现象，本文使用规则构造工具ClassBench-ng[37]构造了复杂的依赖规则链。对于大流和小流来说，规则间的平均依赖深度为10，即一个数据包的匹配过程平均受制于10条高优先级规则的覆盖关系。同时，为了测试极端情况下的算法鲁棒性，在规则集中引入了长尾依赖特征，使得最大依赖深度达到50。

为了评估CA-UPF的性能，本文选择了以下两个开源的UPF实现作为比较基准。

1) Free5GC[38]：基于x86架构的开源解决方案。它提供了符合3GPP Release 15标准的完整UPF实现，并广泛用于工业和学术研究[39]。

2) UPF-Accel[40]：NVIDIA专门为BlueField DPU开发的UPF。UPF-Accel的原始版本不支持PDR规则的基于优先级的匹配。为了确保与CA-UPF进行公平和全面的性能比较，本文对UPF-Accel进行了必要的扩展和修改，通过多管线段串联的方式实现了基于优先级的规则匹配。

需要说明的是，在现有的开源UPF实现中，UPF-Accel是唯一能够在真实DPU硬件上运行的开源加速型UPF，可作为评估DPU能力的基准方案。其他硬件加速类UPF原型[18-21,24]均面向可编程交换机设计，其体系结构与编程接口与BlueField所采用的DOCA框架存在本质差异。因此，本文主要选取UPF-Accel作为对比对象，以全面评估基于DPU的UPF性能表现。

为了保证实验结果的可比性，除非另有说明，后续实验均基于上述所示的默认参数配置进行。

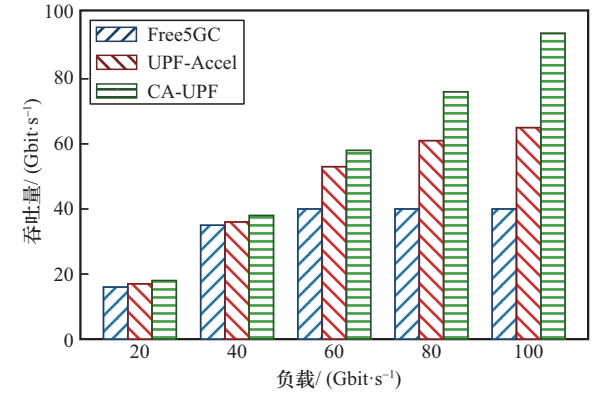
### 4.2 系统整体性能评估

1) 吞吐量性能。为了综合评估CA-UPF在不同流量负载下的转发性能，本文将其与纯软件方案Free5GC及硬件加速方案UPF-Accel进行了对比测试。实验在20~100 Gbit/s的输入速率范围内，对40万条并发流进行了吞吐量测量。图4(a)展示了不同方案的平均吞吐量测试结果。①Free5GC：作为纯软件实现，其性能受限于CPU的包处理瓶颈，在低负载（如20 Gbit/s）下，其转发能力与其他UPF相当，但随着输入速率提升，CPU资源迅速耗尽，吞吐量增长停滞，最终饱和在约40 Gbit/s。②UPF-Accel：得益于DPU硬件加速，其在中高负载（40~100 Gbit/s）表现大幅优于软件方案，然而，缺乏对规则依赖的优化，复杂的规则集迫使硬件执行多级管线段串联，导致转发效率衰减，在100 Gbit/s满载输入下，其吞吐量仅能维持在65 Gbit/s左右。③CA-UPF：在相同软硬件架构条件下，CA-UPF通过快速独立规则生成算法在规则卸载阶

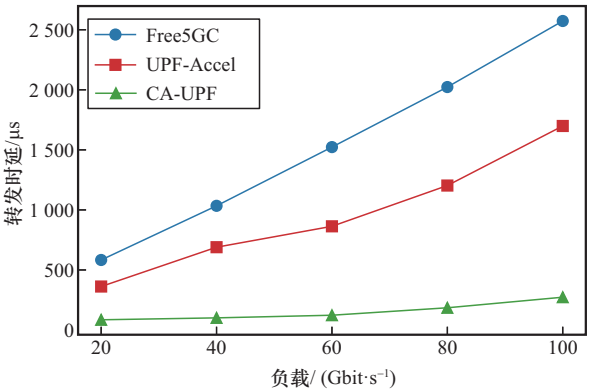
段消除了大流规则的优先级依赖，从而使硬件端仅需执行有限数量的管线段匹配即可完成转发处理。在此基础上，CA-UPF 将小流留驻 CPU 处理，仅将消除了依赖的大流规则卸载至硬件。实验结果表明，该算法有效规避了因规则依赖导致管线段数量过多而引发的吞吐量降低，使 CA-UPF 在所有测试场景中均保持了最高的吞吐性能。特别是在 100 Gbit/s 高负载条件下，CA-UPF 实现了接近线速的转发，吞吐量高达 97 Gbit/s，在所选对比基线中取得最高值。

2) 转发时延性能。在相同的实验配置下，本文进一步评估了 3 种架构在不同吞吐负载下的数据包平均转发时延，测试结果如图 4(b)所示。虽然各方案的转发时延均随吞吐负载的提升呈现增长趋势，但其内在机制与具体表现差异较大。Free5GC 作为纯软件实现，在大流量场景下面临严峻的计算瓶颈，随着负载升高，CPU 资源争用加剧导致排队与处理时延呈非线性急剧增长，峰值时延高达 2 500 μs。相比之下，UPF-Accel 虽然利用了可编程硬件加速，但受制于复杂的规则依赖，硬件必须执行多级管线段串联以维持匹配语义，这种深层流水线处理引入了额外的硬件转发时延。与之不同的是，CA-UPF 通过快速独立规则生成算法在规则卸载前消除了大流规则的依赖，使硬件处理路径能够被简化为有限数量的管线段匹配。在此基础上，软硬协同的分层架构得以有效发挥作用。实验结果表明，CA-UPF 在全负载范围内均保持了最低的转发时延（低于 500 μs），有效验证了其在保障时延敏感型业务服务质量方面的优势。

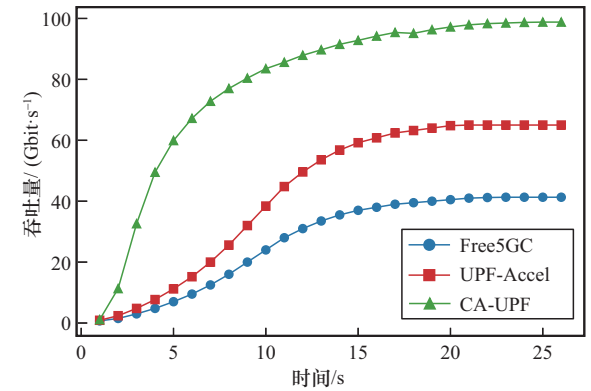
3) 随时间变化的吞吐量性能。图 4(c)进一步展示了在 100 Gbit/s 满载输入下，3 种 UPF 的瞬时吞吐量随时间变化的动态轨迹。由图 4(c)可知，各方案的吞吐量增长特性差异明显。CA-UPF 展现出最优的动态响应能力，其吞吐量在测试初期迅速攀升并收敛至接近 100 Gbit/s 的饱和状态，证明了其能够快速建立转发表项并充分利用链路带宽。相比之下，Free5GC 受限于 CPU 的处理瓶颈，吞吐量增长缓慢且最终仅稳定在约 40 Gbit/s，无法应对线速流量冲击。UPF-Accel 的表现介于两者之间，虽然利用了硬件加速，但其必须在硬件流水线中维护复杂的规则优先级，导致需要卸载的规则增多，管线段处理深度增加，限制了其峰值吞吐量停滞在 65 Gbit/s 左右。实验结果表明，CA-UPF 通过快速独立规则生成算法缩短了规则生成与卸载准备阶段的耗时，使得硬件管线段能够更早进入运行状态，从而在短时间内快速提升至近 100 Gbit/s，并长期维持最高的持续吞吐量。



(a) 各负载下的平均吞吐量比较



(b) 各负载下的转发时延比较



(c) 时间维度下的吞吐量变化

图 4　Free5GC、UPF-Accel 和 CA-UPF 的整体性能评估

### 4.3　影响吞吐量和时延的因素分析

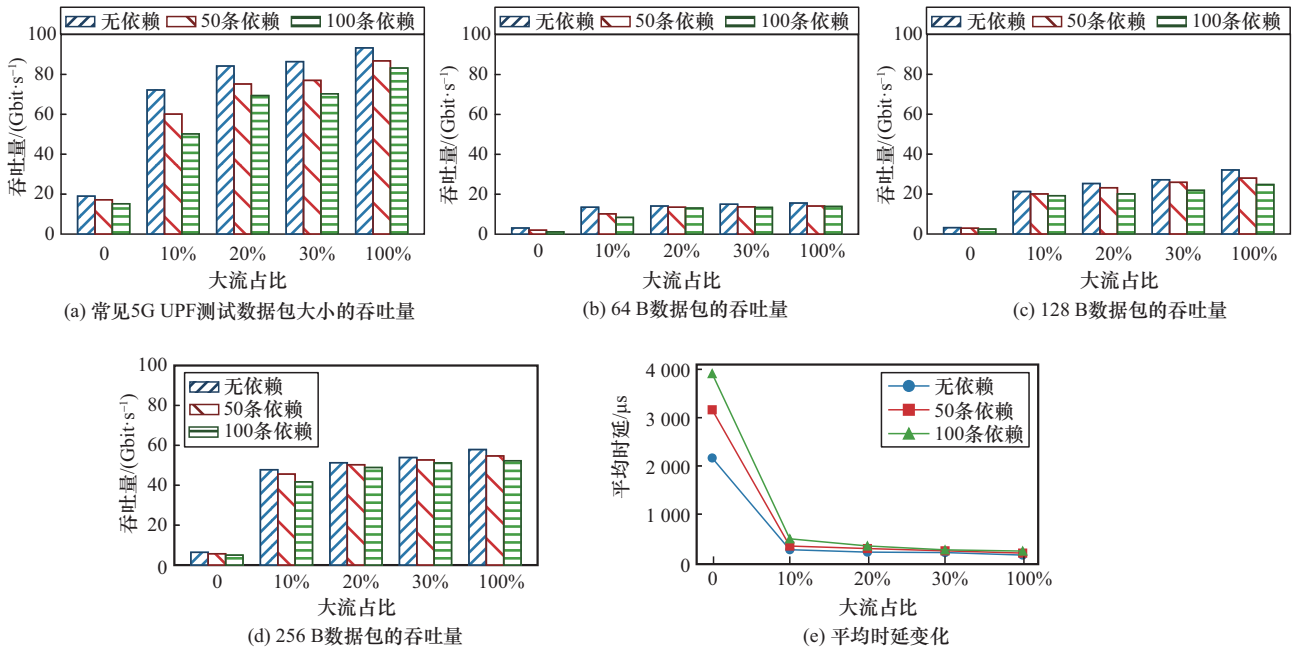
为了全面评估 CA-UPF 架构的性能，本文选取大流占比、依赖规则数量及数据包大小 3 个关键维

度，探究其对系统吞吐量的影响。实验以 1 000 个用户设备作为基准，每个用户生成 10 条随机流及对应的 PDR。这些流被划分为包含 1 万个数据包的大流与包含 100 个数据包的小流。实验设置了从 0~100% 不等的 5 种大流占比场景，以全面评估 CA-UPF 在典型及极端条件下的吞吐性能。同时，为考察规则依赖性的影响，实验配置了无依赖、50 条依赖及 100 条依赖 3 种场景，对应生成的 PDR 规则总量分别约为 1 万条、50 万条及 100 万条。此外，除标准 UPF 测试模型外，实验还引入了 64 B、128 B 及 256 B 的小包测试，以精确量化包长对系统吞吐量的影响。

图 5(a)~图 5(d)展示了在固定数据包与规则数量下，系统吞吐量随大流占比提升而增长的趋势。这主要归因于 CA-UPF 的卸载机制，仅大流规则被卸载至可编程硬件进行加速处理，因此随着大流占比提高，由硬件处理的数据包份额相应增加，从而大幅提升了整体吞吐量。同时，从图 5(a)~图 5(d)中也可以看出，在固定包长与大流占比的条件下，增加依赖规则会导致吞吐量下降。其核心原因在于规则依赖解析算法的计算开销，依赖规则越多，解析过程越复杂，计算耗时越长，进而制约了系统整体吞吐量。最后，考察数据包大小的影响。受限于硬件固有的转发性能瓶颈，较小的数据包通常导致较低的吞吐量。在吞吐量需求恒定的前提下，小包意味着更高的数据包转发速率，这容易接近甚至超出硬件转发能力的上限，导致有效吞吐量性能降低。

在与吞吐率评估相同的实验设置下，本文基于典型 5GUPF 分组长度分布对系统时延进行了测量。CA-UPF 的数据包处理时延主要源于两类路径：由 CPU 处理对应未卸载规则的数据包，以及由 DPU 处理对应已卸载规则的数据包。对于 CPU 处理路径，其端到端时延由数据包解析、PDR 规则匹配与依赖消解计算、数据包封装或解封装处理以及转发时延 4 个部分构成。相比之下，DPU 处理路径的时延构成更为精简，仅包含规则匹配、数据包处理及转发 3 个环节。

图 5(e)展示了不同场景下 CA-UPF 的平均时延变化。在大流占比为 0 的纯小流场景中，所有数据包均由 CPU 处理，受限于 CPU 有限的处理能力，平均时延偏高，甚至超过 3 000 μs，明显劣于含有大流的其他场景。在引入大流的场景中，大流触发规则卸载机制，使数据包直接由 DPU 硬件处理，大幅降低了单包处理开销，促使系统总时延迅速降至 500 μs 以下。关于规则依赖性的影响，随着依赖规则数量增加，平均时延呈现一定幅度的增长。这主要源于规则依赖消解算法复杂度的提升，增加了规则卸载前 CPU 端的预处理耗时。




(a) 常见5G UPF测试数据包大小的吞吐量　(b) 64 B数据包的吞吐量　(c) 128 B数据包的吞吐量

(d) 256 B数据包的吞吐量　(e) 平均时延变化

图 5　不同数据包大小和不同大流量占比下的 CA-UPF 吞吐量与平均时延

### 4.4　算法性能分析与规则重构更新策略评估

为验证本文所提出的快速独立规则生成算法在计算效率与规则生成质量两方面的综合性能优势，本文选取当前最先进的规则依赖消除工作 Awesome-Cache 中的贪婪启发式算法（以下简称贪婪算法）作为主要对比基线，并从规则生成时延和冲突位选择质量两个维度对算法进行评估。

1) 规则生成时延分析。选定目标数据流 IP 地址为 192.168.1.100，并设定其最佳匹配规则为 192.168.0.0/255.255.0.0。在此基础上，随机生成了包含 10 000 条与最佳匹配规则存在依赖关系的规则集。为了全面考察算法在不同规模下的表现，本文构建了从 200~10 000 条不等的多个测试规则子集，并分别评估了两种算法在这些场景下的执行效率。图 6 为不同算法在不同的依赖规则数量下的性能对比。在规则数量较少（低于 1 000 条）的场景中，两者性能表现相当，生成时延的差异微乎其微。然而，随着规则集规模扩展至 2 000 条以上，性能差距逐渐突出。特别是在模拟未来移动网络大规模规则场景（5 000 条及 10 000 条规则）的极端测试中，差异尤为突出：当规则数达到 5 000 条时，贪婪算法的执行时延急剧攀升至约 1 500 μs，而本文算法仅为 500 μs 左右，约为前者的三分之一；在 10 000 条规则的极端情况下，贪婪算法的时延已突破 2 800 μs，远超时延敏感型业务的可接受阈值，而本文算法成功将生成时延控制在约 1 000 μs，仅为对比算法的一半。这充分表明本文算法在大规模规则集下具有快速生成优势，能够满足实时性要求。

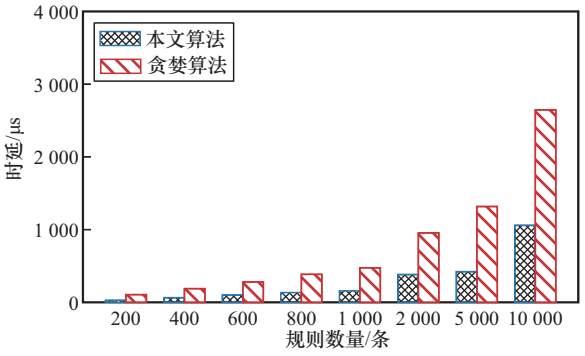



图 6　不同算法在不同的依赖规则数量下的性能对比

2) 规则卸载时延分析。在实际工作流程中，规则生成仅是规则卸载链路的一部分。当流量被判定为大流后，控制平面需将规则下发至可编程硬件并完成表项更新。因此，本文进一步评估在不同规则依赖规模下的规则卸载时延，以验证本文算法对整体更新时间的影响。具体而言，本文以目标规则及其依赖规则数量为自变量，构造不同规模的依赖集合，并对比两种策略的卸载时延。①无算法的规则卸载：目标规则与其依赖规则全部卸载至硬件。②有算法的规则卸载：先执行快速独立规则生成算法构造独立规则，而后仅卸载该独立规则至硬件。两种策略均统计从控制平面触发规则卸载开始，到硬件表项更新完成为止的总时延。实验结果如图 7 所示。当目标规则的依赖规模较小时，无算法的规则卸载策略由于不需要额外计算，规则卸载时延低于有算法的规则卸载策略，例如，在依赖规则数为 10 条的情况下，有算法的规则卸载时延约为 190 μs，无算法的规则卸载时延约为 110 μs；但随着依赖规则数量增大，无算法的规则卸载策略需要下发的规则条目增多，导致表项更新开销快速累积，当依赖规则数量增至 100 条和 1 000 条时，其规则卸载时延分别上升至 1 013 μs 和 10 016 μs。相比之下，有算法的规则卸载策略虽然引入了独立规则生成的计算开销，但由于始终仅需卸载单条独立规则，其规则卸载时延始终保持在 200~320 μs，即使在依赖规则数量达到 2 000 条时也仅约为 320 μs。因此，快速独立规则生成算法不仅能够降低依赖解析的计算开销，还能够在依赖较重的场景中减少规则卸载所需的总更新时间，从而更适合大流量下频繁触发卸载的实际运行环境。

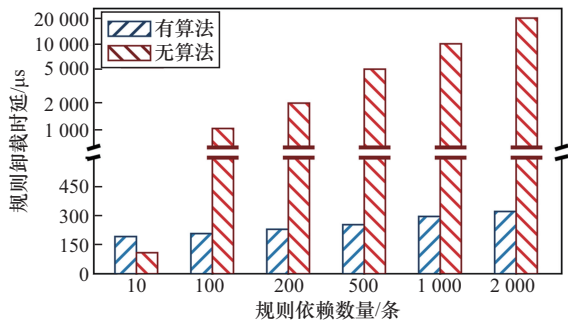



图 7　是否使用算法在不同规则依赖数量下的规则卸载时延

3) 冲突位选择的质量分析。除计算和规则卸载效率外，快速独立规则生成算法还需避免因过度具体化导致的规则匹配空间缩减。为此，本文进一步评估不同算法在冲突位选择质量方面的表现。冲突位数量反映了生成独立规则所需具体化

的比特数，其值越小，表示规则对原始匹配空间的保留程度越高。在该实验中，本文采用 ClassBench-ng 提供的 10 类规则参数模板生成依赖规则集合，以保证测试场景的代表性与可复现性。针对每一类模板，分别构造包含 10、100、500、1 000 及 2 000 条依赖规则的测试实例，并统计不同算法在生成独立规则时所需具体化的冲突位数量。最终结果对不同模板取平均值，以反映算法在典型规则依赖场景下的总体表现。图 8 为不同算法在不同依赖规则规模下的平均冲突位数量对比。为全面刻画规则质量的上下界，在贪婪算法的基础上，引入了额外两个对比算法：①低位优先算法，代表缺乏全局感知能力的快速独立规则生成算法；②最优解算法，用于提供理论上的质量上界。实验结果表明，随着依赖规则数量的增加，各算法生成规则所需的冲突位数量整体呈上升趋势。低位优先算法在所有测试规模下均需要具体化最多的冲突位，规则质量较差。相比之下，本文算法在所有测试场景下生成的平均冲突位数量均优于低位优先算法，并与贪婪算法及最优解算法保持较小差距。在依赖规则数量不超过 2 000 条的场景中，本文算法生成规则的平均冲突位数量仅比最优解算法高约 0.6 bit，表明其在规则质量上接近最优。与贪婪启发式算法不同，本文算法在结构上将规则依赖消解过程重构为对固定宽度差异比特掩码的局部操作，使冲突位定位问题能够直接映射为位级计算。在此基础上，计数尾随零内联函数可被自然用于快速定位有效冲突位，并由编译器直接生成 ARM 架构下的对应机器指令，从而降低规则生成阶段的计算开销。实验结果表明，与贪婪算法相比，本文算法有效降低了计算开销，在整体系统性能上展现出较大优势。

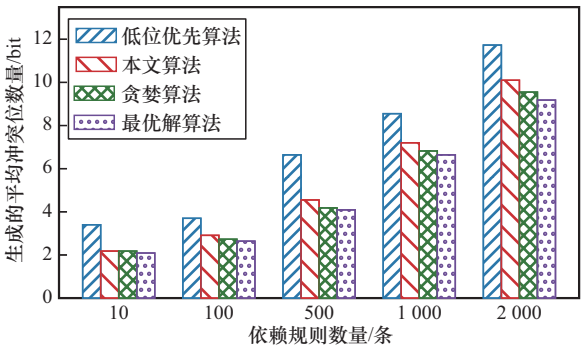



图 8　不同算法在不同依赖规则数量下的平均冲突位数量对比

为量化规则重构方法对系统内存占用的优化效益，本文在不同规则集规模下进行了对比测试。图 9 为是否使用规则重构在不同规则数量条件下的内存占用对比。实验选取了两组具有代表性的数据集：分别包含 10 000 条和 100 000 条下行 PDR 与 FAR。选取下行规则作为测试基准的主要原因在于，该场景下每条 PDR 与 FAR 存在严格的一一映射关系，最能体现规则重构策略在消除冗余存储方面的聚合优势。图 9 的实验数据显示，在规则数量较少时，规则重构带来的内存节省相对有限；然而，随着规则集规模的指数级增长，其优化效果愈发明显，在包含 10 000 条规则的场景下，内存占用降低约 50%；当规则规模进一步扩展至 100 000 条时，内存占用降幅提升至约 70%。这是因为大规模规则集为规则重构提供了更大的聚合空间，使得 PDR 与 FAR 合并存储带来的边际效益提升，从而提高了硬件资源的利用效率。

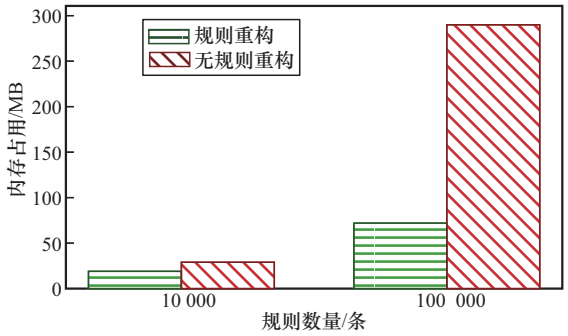



图 9　是否使用规则重构在不同规则数量下的内存占用对比

针对大规模流表场景下的更新瓶颈，本文进一步评估了规则更新优化机制对降低流表项更新时延的有效性。图 10 为是否使用规则更新优化在不同的已存储规则数量下的规则更新时延。实验结果表明，未采用优化机制时，更新 1 000 条规则的时延随已存储的规则数量的增加呈现出线性增长趋势，暴露出单管线处理能力的局限性。相比之下，引入优化方案后，更新时延平均降低约 50%。这得益于本文提出的多管线负载均衡策略，通过将更新任务合理分配至多个并行管线段中处理，有效缓解了单条管线段的同步压力与阻塞，从而大幅缩短了批量规则更新的完成时间。该结果证明，规则更新优化机制在应对海量并发连接建立与释放的动态场景时，对保障系统的高可扩展性与性能稳定性具有重要意义。

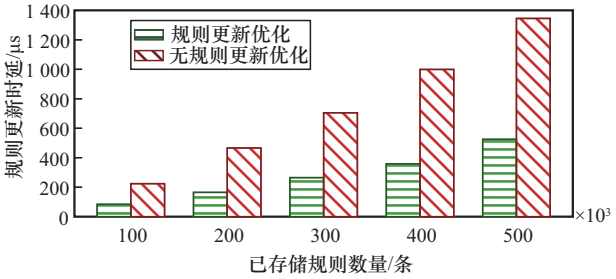



图 10　是否使用规则更新优化在不同的已存储规则数量下的规则更新时延

## 5　结束语

随着 6G 移动网络的发展，UPF 面临着复杂规则管理和高吞吐量、低时延的性能需求。传统的 CPU 架构和硬件加速方案各有优势，但在处理大规模复杂规则时，仍存在性能瓶颈。为解决这些挑战，本文提出了 CA-UPF 架构，结合 NVIDIA BlueField-2 DPU 的硬件加速和软件的灵活性，提供了软硬协同的高效 UPF 解决方案。CA-UPF 的主要创新在于提出了一种快速独立规则生成算法，能够实时消除规则间的优先级依赖，降低了计算开销并提高了硬件转发效率。同时，针对 DPU 硬件特性，优化了规则存储结构和更新策略，提升了内存利用率并减少了规则更新时延。实验结果表明，CA-UPF 在高并发流量和复杂规则场景下，能够实现 97 Gbit/s 的吞吐量，并在 40 万个流的条件下，将大流的转发时延控制在 500 μs 以下。相比现有独立规则生成算法，CA-UPF 减少了 65% 以上的计算开销，展现了优异的性能和可扩展性。未来的工作将聚焦于在更广泛的硬件平台上验证 CA-UPF 架构，特别是在可编程交换机等硬件上的应用，同时进一步提升规则处理效率和扩展性，并结合人工智能技术优化流量分类和规则调度策略。

## 参考文献：

[1] Wang C X, You X H, Gao X Q, et al. On the road to 6G: visions, requirements, key technologies, and testbeds[J]. IEEE Communications Surveys & Tutorials, 2023, 25(2): 905-974.

[2] Hossain E, Vera-Rivera A. 6G cellular networks: mapping the landscape for the IMT-2030 framework[J]. IEEE Transactions on Technology and Society, 2025, 6(4): 377-392.

[3] Cai Y L, Swindlehurst A L, Yener A, et al. Guest editorial: special issue on next generation advanced transceiver technologies: part I[J]. IEEE Journal on Selected Areas in Communications, 2025, 43(3): 577-581.

[4] Pennanen H, Hänninen T, Tervo O, et al. 6G: the intelligent network of everything[J]. IEEE Access, 2025, 13: 1319-1421.

[5] Kim J, Lee Y, Lim H, et al. OutRAN: co-optimizing for flow completion time in radio access network[C]//Proceedings of the 18th International Conference on Emerging Networking EXperiments and Technologies. New York: ACM Press, 2022: 369-385.

[6] Jain V, Panda S, Qi S X, et al. Evolving to 6G: Improving the Cellular Core to lower control and data plane latency[C]//Proceedings of the 2022 1st International Conference on 6G Networking (6GNet). Piscataway: IEEE Press, 2022: 1-8.

[7] 王晓云, 陆璐, 刘超, 等. 面向 6G 的网络架构建模、评估及优化[J]. 通信学报, 2024, 45(7): 235-249.
Wang X Y, Lu L, Liu C, et al. Modeling, evaluation, and optimization for 6G network architecture[J]. Journal on Communications, 2024, 45(7): 235-249.

[8] Katta N, Alipourfard O, Rexford J, et al. CacheFlow: dependency-aware rule-caching for software-defined networks[C]//Proceedings of the Symposium on SDN Research. New York: ACM Press, 2016: 1-12.

[9] Burstein I. Nvidia data center processing unit (DPU) architecture[C]//Proceedings of the 2021 IEEE Hot Chips 33 Symposium (HCS). Piscataway: IEEE Press, 2021: 1-20.

[10] Li F L, Chen Q, Shen J X, et al. Performance characteristics and guidelines of offloading middleboxes onto BlueField-2 DPU[J]. IEEE Transactions on Computers, 2025, 74(2): 609-622.

[11] Bhattacharyya A, Ramanathan S, Fumagalli A, et al. An end-to-end DPDK-integrated open-source 5G standalone radio access network: a proof of concept[J]. Computer Networks, 2024, 250: 110533.

[12] Scheich C, Corici M, Buhr H, et al. eXpress data path extensions for high-capacity 5G user plane functions[C]//Proceedings of the 1st Workshop on eBPF and Kernel Extensions. New York: ACM Press, 2023: 86-88.

[13] Zhou J E, Ma Z X, Tu W J, et al. Cable: a framework for accelerating 5G UPF based on eBPF[J]. Computer Networks, 2023, 222: 109535.

[14] Jia H N, Wang M, Li B Y, et al. 5GC$^2$ache: improving 5G UPF performance via cache optimization[J]. arXiv Preprint, arXiv: 2404.13991, 2024.

[15] Li Y, Ma J W, Ying X, et al. High-performance 5G CUPF based on X86 hardware platform[J]. ZTE Technology Journal, 2025, 31(1): 58-62.

[16] Jain V, Chu H T, Qi S X, et al. L25GC: a low latency 5G core network based on high-performance NFV platforms[C]//Proceedings of the ACM SIGCOMM 2022 Conference. New York: ACM Press, 2022: 143-157.

[17] Qi S X, Ramakrishnan K K, Chen J C. L26GC: evolving the low-

latency core for future cellular networks[J]. IEEE Internet Computing, 2024, 28(2): 29-36.

[18] MacDavid R, Cascone C, Lin P P, et al. A P4-based 5G user plane function[C]//Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR). New York: ACM Press, 2021: 162-168.

[19] Panda S, Ramakrishnan K K, Bhuyan L N. Synergy: a SmartNIC accelerated 5G dataplane and monitor for mobility prediction[C]//Proceedings of the 2022 IEEE 30th International Conference on Network Protocols (ICNP). Piscataway: IEEE Press, 2022: 1-12.

[20] Liu Y Z, Nie H, Cai H, et al. X-plane: a high-throughput large-capacity 5G UPF[C]//Proceedings of the 29th Annual International Conference on Mobile Computing and Networking. New York: ACM Press, 2023: 1-14.

[21] Wen Z, Yan G. HiP4-UPF: towards {high-performance} comprehensive 5G user plane function on P4 programmable switches[C]//Proceedings of the 2024 USENIX Annual Technical Conference (USENIX ATC 24). Berkeley: USENIX Association, 2024: 303-320.

[22] Moon Y, Han Y, Kim S, et al. Data plane acceleration using heterogeneous programmable network devices towards 6G[C]//Proceedings of the ICC 2024 - IEEE International Conference on Communications. Piscataway: IEEE Press, 2024: 421-426.

[23] Singh S K, Rothenberg C E, Langlet J, et al. Hybrid P4 programmable pipelines for 5G gNodeB and user plane functions[J]. IEEE Transactions on Mobile Computing, 2023, 22(12): 6921-6937.

[24] Chen C C, Chang C Y, Nikaein N. IUP: integrated and programmable user plane for next-generation mobile networks[J]. IEEE Network, 2025, 39(3): 91-98.

[25] Harkous H, Kak A, Urie A, et al. Flat UP: toward RAN-core convergence for the 6G user plane[J]. IEEE Communications Magazine, 2025, 63(5): 62-68.

[26] Yan B, Xu Y, Xing H Y, et al. CAB: a reactive wildcard rule caching system for software-defined networks[C]//Proceedings of the Third Workshop on Hot Topics in Software Defined Networking. New York: ACM Press, 2014: 163-168.

[27] Li X F, Xie W C. CRAFT: a cache reduction architecture for flow tables in software-defined networks[C]//Proceedings of the 2017 IEEE Symposium on Computers and Communications (ISCC). Piscataway: IEEE Press, 2017: 967-972.

[28] Sheu J P, Chuo Y C. Wildcard rules caching and cache replacement algorithms in software-defined networking[J]. IEEE Transactions on Network and Service Management, 2016, 13(1): 19-29.

[29] Li R, Zhao B H, Chen R X, et al. Taming the wildcards: towards dependency-free rule caching with FreeCache[C]//Proceedings of the 2020 IEEE/ACM 28th International Symposium on Quality of Service (IWQoS). Piscataway: IEEE Press, 2020: 1-10.

[30] Wan Y, Song H Y, Xu Y, et al. T-cache: dependency-free ternary rule cache for policy-based forwarding[C]//Proceedings of the IEEE INFOCOM 2020 - IEEE Conference on Computer Communications. Piscataway: IEEE Press, 2020: 536-545.

[31] Luan Z Y, Li Q, Zhang Z T, et al. AWEsome-cache: dependency-free rule-caching for arbitrary wildcard patterns in TCAM[C]//Proceedings of the 2023 IEEE 31st International Conference on Network Protocols (ICNP). Piscataway: IEEE Press, 2023: 1-12.

[32] Xing J R, Qiu Y M, Hsu K F, et al. Unleashing SmartNIC packet processing performance in P4[C]//Proceedings of the ACM SIGCOMM 2023 Conference. New York: ACM Press, 2023: 1028-1042.

[33] Chen X, Sun X, Zhang W B, et al. Accelerating sketch-based end-host traffic measurement with automatic DPU offloading[C]//Proceedings of the IEEE INFOCOM 2024 - IEEE Conference on Computer Communications. Piscataway: IEEE Press, 2024: 171-180.

[34] Caprara A, Toth P, Fischetti M. Algorithms for the set covering problem[J]. Annals of Operations Research, 2000, 98(1/2/3/4): 353-371.

[35] Turull D, Sjödin P, Olsson R. Pktgen: Measuring performance on high speed networks[J]. Computer Communications, 2016, 82: 39-48.

[36] Wu Q, Zhai X B, Liu X, et al. Performance tuning via lean measurements for acceleration of network functions virtualization[J]. IEEE/ACM Transactions on Networking, 2023, 31(1): 366-379.

[37] Matousek J, Lucansky A, Janecek D, et al. ClassBench-ng: benchmarking packet classification algorithms in the OpenFlow era[J]. ACM Transactions on Networking, 2022, 30(5): 1912-1925.

[38] Chen J C. Free5GC[R]. 2025.

[39] Mukute T, Mamushiane L, Lysko A A, et al. Control plane performance benchmarking and feature analysis of popular open-source 5G core networks: OpenAirInterface, Open5GS, and free5GC[J]. IEEE Access, 2024, 12: 113336-113360.

[40] NVIDIA. UPF-Accel [R]. 2025.

[作者简介]

**刘松**（1998– ），男，河北邢台人，南开大学博士生，主要研究方向为可编程网络、网内计算和5G网络架构优化。

**张琳**（2002– ），女，河北沧州人，南开大学硕士生，主要研究方向为可编程网络、5G网络架构优化和DPU加速。

**高铠炜**（2003– ），男，江苏淮安人，南开大学硕士生，主要研究方向为可编程网络。

**苑新婧**（1995– ），女，天津人，博士，南开大学副教授，主要研究方向为网络安全、可编程网络和传输资源调度。

**雷玮琛**（1999– ），男，山西朔州人，南开大学硕士生，主要研究方向为可编程网络和5G网络架构优化。

**蒲凌君**（1988– ），男，天津人，博士，南开大学副教授，主要研究方向为边缘智能、具身智能系统设计和资源调度。

**师建新**（1997– ），男，河北张家口人，博士，南开大学讲师，主要研究方向为可编程网络、网络智能与资源管理、新型流媒体传输与处理、低轨卫星网络等。

**张建忠**（1964– ），男，河北石家庄人，博士，南开大学教授、博士生导师，主要研究方向为计算机网络与网络安全。