arXiv:2305.01890v1 [cs.NI] 3 May 2023

亚毫秒级延迟服务级别目标下的网络功能链调度

王建峰[1], 悉达 nt Gupta[1], Marcos A. M. Vieira[2],Barath Raghavan[1],和 Ramesh Govindan [1]

[1]南加州大学[2]米纳斯吉拉斯联邦大学

摘要

网络功能虚拟化（NFV）旨在用基于软件的网络功能（NFs）取代硬件中间盒。网络功能虚拟化系统在云和边缘的部署日益广泛。然而，尤其是在边缘，网络功能虚拟化对吞吐量的传统关注与满足极低延迟 SLO 的需求之间存在不匹配，因为边缘服务本身就要求低延迟。此外，基于云的网络功能虚拟化系统需要在尽可能减少 CPU 核心使用率的同时实现如此低的延迟。

我们发现，真实世界流量具有突发性，会导致现有网络功能虚拟化系统中的延迟峰值高达数十毫秒。为此，我们构建了 NetBlaze，采用一种新颖的多尺度核心扩缩容策略，即使面对对抗性流量，也能实现亚毫秒级的 p99 延迟服务等级目标。NetBlaze 在机架、服务器和核心这三个空间尺度上，以逐渐精细的时间尺度做出流量到核心的分配决策，从而应对多时间尺度的突发流量。与最先进的方法相比，NetBlaze 是唯一能够在使用相当数量核心的同时，实现亚毫秒级 p99 延迟服务等级目标的方法。

1 引言

网络功能虚拟化（NFV）支持在软件中处理数据包，通常采用由网络功能（NFs）组成的链。网络功能可以过滤流量、对其进行加密、修改传输中的标头或有效载荷等。大量研究表明，与硬件中间盒相比，网络功能虚拟化提高了网络的可管理性 [20, 28, 29, 35, 37–39]。近期研究进一步证明了网络功能虚拟化具备良好的性能——在商用硬件上，网络功能虚拟化系统能够以线速处理复杂的网络功能链 [3, 18, 36, 39, 41]。

受这些发展的推动，云服务提供商已开始向客户提供网络功能虚拟化能力。这在当今分布式云环境的边缘最为明显 [1, 2, 12],：云机架向客户开放了处理数据包的能力。在这种场景下，数据包处理的延迟必须尽可能短。网络功能链的执行纯属额外开销，而在网络延迟较低的边缘（在近 50 个国家/地区，云端往返时延（RTT）低于 20 毫秒 [26]），即使几毫秒的延迟也可能造成显著的额外开销，尤其是因为延迟增加会对收入产生不利影响。

随着处理器速度不断提高，许多网络功能链能够在几十微秒内处理单个数据包。然而，如今的网络功能虚拟化系统的第99百分位（p99）延迟仍在数毫秒量级（§4）。如下文所述，延迟增加的很大一部分源于排队。本文探讨的问题是：能否设计一种网络功能虚拟化系统，实现微秒级的 p99 延迟服务等级目标（SLO）？

我们面临的一个主要挑战是网络功能虚拟化对流亲和性的要求（§2.1）。网络功能虚拟化系统会将流分配给核心，使同一流中的所有数据包都由同一个核心处理。对于每个数据包都会更新共享状态的有状态网络功能而言，这样可以避免昂贵的跨核心同步。为在保持流亲和性的同时实现低延迟，网络功能虚拟化系统会自动扩缩容核心分配 [18, 36, 39] ——随着流量增加或减少，分别增加或减少分配给流量处理的核心数量。这既能确保及时处理流量，又能最大限度地降低核心使用率。在云环境中，这一点尤为重要，因为未用于数据包处理的核心可以用于处理能够带来收入的应用工作负载。

如今网络功能虚拟化系统中的自动扩缩容算法无法及时应对真实流量中出现的两类突发流量，而这些突发流量会显著增加排队时延（§2.2）。第一类是流中瞬时数据包速率超过单个核心的处理能力。第二类是大量流到达形成的突发，同样会超过核心容量，因为每条新流都需要为有状态网络功能额外支付设置开销。这些突发流量可能持续时间很短（10 毫秒或更短），却会导致数毫秒的排队时延（§2.2）。

现有网络功能虚拟化自动扩缩容方法在真实流量下无法实现低延迟，原因有三（§2.3）：它们仅根据观测到的数据包速率做出核心分配决策，因此无法考虑流到达突发；它们使用时间平均估算，无法捕捉短时间尺度的突发；它们以更长的时间尺度（100 ms）做出核心分配决策

比真实流量中突发流量的持续时间长 1 秒。

贡献。本文介绍了 NetBlaze 的设计与实现。NetBlaze 是一种机架规模 NFV 系统，能够在确保高 CPU 效率的同时实现微秒级延迟服务级别目标（§3）。其贡献包括：

- 一种新颖的分层多尺度流到核心映射策略，将核心分配分布在不同的空间尺度（入口、服务器、核心）和不同的时间尺度（从入口处的秒级到核心处的微秒级（§3.1）。这确保了 NetBlaze 能够适应不同时间尺度上的流量变化，同时通过在入口处（流量最多的位置）进行粗粒度决策，并在核心处（流量相对较少的位置）进行细粒度决策，确保核心分配决策具有可扩展性。
- 一种更准确的网络功能核心容量估算技术，同时考虑瞬时流数和数据包速率。
- 一种轻量级机制，用于招募辅助 核心，以处理可能违反延迟 SLO 的突发流量。
- 一种快速算法，用于将服务器上的 RSS 桶重新映射到核心，以确保没有任何核心持续处于过载状态。

通过骨干网和互联网服务提供商内部的流量轨迹开展实验（§4）表明，NetBlaze 能够支持约为 100–500 μs 的 p99 延迟服务等级目标，而最先进的网络功能虚拟化系统 [8,18,39] 表现出 10× 更高的 p99 延迟服务等级目标。此外，NetBlaze 的 CPU 效率与所有这些系统相当，甚至更高。消融研究表明，其每项设计决策都对 NetBlaze 的性能有显著贡献。

2 背景、动机与方法

为了说明 NetBlaze 所要解决的问题及其采用的方法，我们先介绍一些背景知识。

2.1 背景

部署场景。许多云服务提供商采用分布式云设计，在网络边缘部署机架 [1,2,12]。这些机架部署要么位于企业场所内，要么位于运营商的入口处（例如，电信运营商）。每个机架都包含一台机架顶部交换机和多台多核服务器。每台服务器的核心都可以运行边缘应用，例如多接入边缘计算（MEC）所设想的应用。核心也可以运行软件网络功能：事实上，这些部署中的云服务提供商软件明确支持软件网络功能。本文针对这些部署中的网络功能处理。

网络功能和网络功能链。网络功能可以通过多种方式处理数据包：转换网络地址（例如，网络地址转换）、过滤流量（例如，使用访问控制列表）、加密流量等。通常，流由网络功能链处理。单个边缘机架可以执行多个网络功能链，以服务不同类别的流量。对于每条网络功能链，运维人员都会定义该链应处理哪种流量类别（或流量聚合）。

延迟 延迟 ***SLO***。在边缘机架环境中，最好能够 e to 限定使用网络功能链处理数据包所产生的延迟。这有助于云服务提供商满足应用的端到端延迟目标。这个界限称为延迟 SLO，以数据包穿过网络功能链时经历的延迟的高百分位数表示（例如，第95百分位或第99百分位，分别记作 p95 或 p99 ）。这种延迟主要由两部分构成：网络功能链处理数据包所需的时间，以及数据包经历的排队延迟。

流亲和性。许多网络功能是有状态的：它们在处理数据包时会创建并更新内部状态。大多数有状态网络功能会维护逐流状态 [16]：网络功能遇到新流时会创建状态，而该流的每个数据包都可能更新逐流状态。如果同一流的数据包由不同核心处理，网络功能可能需要付出大量同步和缓存一致性开销，以确保各核心对逐流状态的更新保持一致。因此，网络功能系统通常会保留流亲和性：在可能的情况下，同一流的数据包由单个核心处理。

自动扩缩容。Metron [18], Quadrant [39], 和 Dyssect [8] 等网络功能虚拟化系统会将网络功能或网络功能链分配给单个核心。在这些论文所考虑的场景中，与我们的场景一样，网络功能链处理的流量通常超过单个核心的处理能力，因此会为每个网络功能或网络功能链分配多个核心。

这些系统会自动扩缩容 CPU 核心分配：动态调整分配给某个网络功能或网络功能链的核心数量。具体而言，自动扩缩容会（a）动态跟踪流量的增减，并（b）相应地增加或减少分配给各网络功能链的核心数量，目标是以尽可能少的核心处理该网络功能链的流量。这样可以提高 CPU 利用率，因为未用于数据包处理的核心可以用于应用程序或后台任务。[1]

这些系统基于不同目标自动扩缩容核心使用率。Metron [18] 会自动扩缩容，以确保有足够的核心处理当前流量，同时尽量减少因核心过载造成的丢包。Quadrant [39] 会自动扩缩容，以确保达到目标 p99 延迟 SLO，而 Dyssect [8] 会自动扩缩容，以确保达到目标 p50 中位延迟。我们的工作重点是通过自动扩缩容来确保 p99 延迟服务等级目标处于 200–300μs。[2]

2.2 挑战

有两个因素使自动扩缩容面临挑战。首先是延迟 SLO。如果网络功能虚拟化系统分配的核心太少，数据包可能会排队，进而导致严重的服务级别目标违规。如果分配的核心太多，则可能降低 CPU 效率。其次是流亲和性：自动扩缩容解决方案不得违反流亲和性，否则会使网络功能设计变得复杂

[1]近期工作 [11,27,32,43] 在云环境中 RPC 处理的核心分配方面也观察到了类似现象。[2]Quadrant 在合成的非突发流量上实现了类似的服务等级目标。在 §4中，我们展示了在真实流量轨迹上，其第99百分位时延可能高达数毫秒。

网络功能链	流量输入	
	骨干网	AS
轻负载（5μs）	831.3	53.8
中等负载（10μs）	2,690	578
重型（20 μs）	6,648	3,584

表1： 大流可能会抬高 p99 延迟。每个单元格表示相应网络功能链和流量轨迹的 p99 延迟，单位为μs。

网络功能链	流量输入	
	骨干网	AS
轻量（5μs）	4,029	3,062
中等（10μ秒）	3,830	14,241
重型（20 μs）	4,671	14,558

表2： 小流会抬高 p99 延迟。每个单元格表示相应网络功能链和流量轨迹的 p99 延迟，单位为μs。

并造成显著的性能损失。[3] 流亲和性约束使网络功能链的核心分配从根本上有所不同，因此，像最短队列优先（JSQ）调度这样的通用任务调度方案并不适用于我们的场景。

除此之外，突发流量也给高效自动扩缩容带来了挑战。当流量突发到来时，如果网络功能链没有足够数量的核心，突发流量中的数据包就可能产生排队延迟。如果突发流量较小，或者延迟 SLO 较宽松，则可能不会发生服务级别目标违规。然而，我们发现，这些极端流量情况确实存在于真实世界流量中，并会导致严重的服务级别目标违规。

超级突发。在真实的数据包跟踪中，可能出现两种类型的大规模突发流量（超级突发），从而导致服务级别目标违规。据我们所知，我们是首个将超级突发识别为 NF 系统核心分配挑战的研究团队。

大流。 大流是指其数据包处理需求超过单个核心处理能力的<笔误>单个</笔误>流（或其一部分）。例如，假设某个网络功能链处理一个数据包需要 10 μs；如果在 10 ms 内，单个流上到达的数据包超过 10^3 个，就会超出该时间窗口内核心的处理能力。大流的出现源于 NF 系统中的流亲和性约束。

在实际环境中采集的数据包跟踪中包含大流。为说明这一点，我们使用了两份数据包跟踪：一份是城市骨干网轨迹 [7]。，另一份采集自大型自治系统（AS） [24]。对于这两份跟踪数据，我们计算了以下情况下产生的 p99 延迟：将每个流分别放在一个核心上处理，并使用 3 种不同类型的网络功能链：一种轻型链，处理每个数据包需要 5 μs，以及中型和重型链，分别需要 10 和 20 μs。为此，我们使用了数据包级离散仿真框架。该框架模拟一个机架规模 NFV 系统，其中包含一台机架顶部交换机和多台多核服务器。框架以真实流量跟踪数据为输入，重放流量轨迹，并将流量导向运行网络功能链的仿真集群。

在这种设置下，由于每个流都由单个核心处理，因此预计 p99 延迟会接近网络功能链的处理时间。然而，我们发现，在某些情况下，整体 p99 延迟高出两个数量级以上（表1）：对于重型链，骨干网流量轨迹为 6.6 毫秒，AS 跟踪为 3.6 毫秒。这表明，在这两种流量轨迹中，都有一个或多个流的数据包到达速率在短时间内超过核心容量，导致严重排队，进而影响整体延迟。

[3]近期大量研究探讨了低开销微秒级远程过程调用的核心分配 [11,27,32,43]。

小流。 小流表示单个核心上的大量活动流，这些流的总体处理需求超过了单个核心的处理能力。在自动扩缩容期间，即使没有大流，小流也可能导致尾延迟 SLO 违规。这是因为：(a) 如果对应的网络功能链是有状态的，那么在短时间内到达数据包的流数量（其中可能包含突发流）可能远高于中位数；(b) 为每个流创建状态所带来的开销可能会耗尽核心容量。

真实数据包跟踪中也包含小流。为展示这一点，我们采用与上文所述相同的两条流量轨迹，但：(a) 使用流哈希（类似于基于 RSS 的系统所用的方法）将流分配到核心；(b) 分配足够多的核心，使整个流量轨迹的 p50 延迟低于网络功能处理时间的 10 倍；(c) 对每条流进行整形，使突发性仅由流到达造成，而非数据包到达造成。

理想情况下，由于流量经过整形，我们会预期 p99 延迟接近 p50 延迟。然而，如表 2所示，p99 延迟要高得多（在 AS 跟踪中，重型网络功能链的延迟接近 14.6 ms）。

2.3 方法

现有方法的不足。 面对超级突发，现有系统无一能够 [8, 18, 39] 维持较低的 p99 延迟服务等级目标。我们在§4 中通过在集群测试平台上使用真实流量轨迹[4]进行实验来证明这一点；直观而言，这是因为：

- 它们仅根据包数做出核心重新分配决策。这没有考虑小流超级突发。
- 其中一些方案在流量入口处，利用从各个服务器收集的平滑流量统计信息做出核心重新分配决策。平滑流量统计信息不足以捕捉短时间尺度上的流量需求变化。这还会在测量—响应循环中引入新的延迟来源，导致入口无法快速应对各个 CPU 核心上的瞬时突发。
- 它们都以相对粗粒度的时间尺度（100 毫秒至 1 秒）进行核心重新分配决策。如果 p99 延迟 SLO 是 500 μs，这显然远远不够。要满足这一服务等级目标，系统需要检测并作出反应[5] to

[4]其中，Quadrant [39] 展示了 50-300 μs 范围内的 p99 延迟 SLO。可是，他们的实验回放的是合成非突发流量；我们已向作者核实这一点。[5]我们无法预测小流或大流何时出现。主动式方法

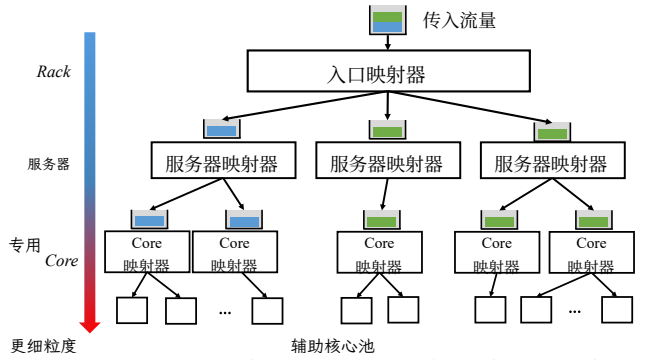


图 1： NetBlaze 的分层多尺度分配在三个不同的空间尺度和时间尺度上将流量映射到核心。

潜在的延迟违规情况可能发生在短于 500 μs 的更细时间尺度上；或者保守地在更长时间间隔内过度分配核心，牺牲 CPU 效率。

NetBlaze 的方法。 对于包含超级突发的流量，NetBlaze 可支持约 100 μs 量级的 p99 延迟服务等级目标，同时确保较高的 CPU 效率。它通过两个关键思路实现这一点：在不同的空间尺度和时间尺度上进行分层多尺度分配，以及在使用数据包数或速率之外，结合流数量来估算核心容量。下一节将对此进行介绍。

3 NetBlaze 设计

本节首先概述 NetBlaze，随后详细介绍其算法。

3.1 概述

目标。 NetBlaze 是一种机架级 NFV 编排系统，旨在同时实现高 CPU 效率（以使用的核时衡量），并确保微秒级的 p99 延迟 SLO。

假设。 （1）NetBlaze 的用户（例如，网络运维人员）指定一组要在机架上运行的网络功能链。每条链处理一个由运维人员定义的流量聚合（即，一组流），运维人员还会为该链指定 p99 延迟 SLO。NetBlaze 力求满足网络功能链所处理流的这一 SLO。（2）流量聚合规模足够大，因此 NetBlaze 可以专门分配一台服务器来处理一条链；NetBlaze 面向企业边缘或互联网服务提供商入口，这些位置的流量可能十分庞大。（3）在大多数情况下，网络功能链会在一个核心上运行至完成 [39]；事实证明，运行至完成是网络功能链的一种低开销执行策略。当核心不运行网络功能链时，可以执行边缘应用。

主要挑战。 鉴于网络功能处理的流亲和性约束，NetBlaze 设计中的主要挑战是确定一种策略（或一组策略），并结合相应机制，将流映射到核心，使 NetBlaze 能够实现上述目标。如 §2.3所述，在 100 毫秒量级的时间尺度上，单纯依据入口并利用数据包统计信息进行分配，无法满足 NetBlaze 的需求。这就引出了一个问题：应在何处做出分配决策？又应采用何种时间尺度？

分层多尺度分配。 Net-Blaze 的核心思想是将流映射到三个不同的空间尺度（入口、服务器和核心），以及三个不同的时间尺度（秒、100 毫秒和数百个 μs，分别对应），每个尺度具有不同的目标（图 1）：

- 在入口映射器处，NetBlaze 旨在最大限度减少分配给网络功能链的服务器数量。
- NetBlaze 的服务器映射器，运行在每台运行网络功能链的 NetBlaze 服务器上。它尝试使用最少的高利用率核心，使 NetBlaze 能够处理或适应持续数百毫秒时间尺度的突发流量。它使用专用 核心进行网络功能处理。
- 在每个核心上，NetBlaze 的核心映射器会从资源池中调配辅助核心，以应对微秒级突发流量。辅助核心通常用于其他应用，但也可临时调配用于网络功能链处理。

处理小流和大流。 核心映射器检测到小流可能导致服务级别目标违规时，会迅速调配一个或多个辅助核心（位于同一服务器上），并将迁移 流到这些核心上。核心映射器检测到大流时，会拆分 网络功能链中的网络功能，在辅助核心上执行一个或多个网络功能。辅助核心完成分配的流的处理后，会恢复应用程序处理。

处理流数量依赖。为了将流映射到核心，网络功能虚拟化系统需要估计或预测，对于网络功能链，在给定延迟 SLO 的情况下，一条链能够处理多少数据包或流。一些系统会动态地、被动地确定这一点 [18],另一些系统则使用通过离线性能分析得出的核心容量预测器 [39]。鉴于 NetBlaze 严格的时间要求，它采用后者。不过，与以往基于包数的核心容量预测器不同，NetBlaze 以流数量和包数来预测核心容量，因为较高的流数量可能违反延迟 SLO（§2.2）。

在接下来的小节中，我们将更详细地介绍分层多尺度分配。为简化描述，我们聚焦于机架上承载的一条网络功能链。通过简单的扩展，我们的 NetBlaze 实现也支持在机架上运行多条网络功能链。

3.2 吸收突发流量：核心映射器

目标与挑战。 服务器映射器将映射到某个网卡队列的一组流分配给一个专用核心。核心映射器在每个专用核心上各运行一个实例，旨在防止其所负责的流违反服务等级目标。为此，它必须在小于目标延迟 SLO（100 微秒）的时间尺度上做出流重新映射决策

例如，将流拆分到多个核心上（针对大流）或保守地分配核心，都可能导致 CPU 使用效率低下和/或处理开销增加。因此，我们着重研究反应式策略：尽可能快速地检测大流和小流，并迅速做出核心重新分配决策。

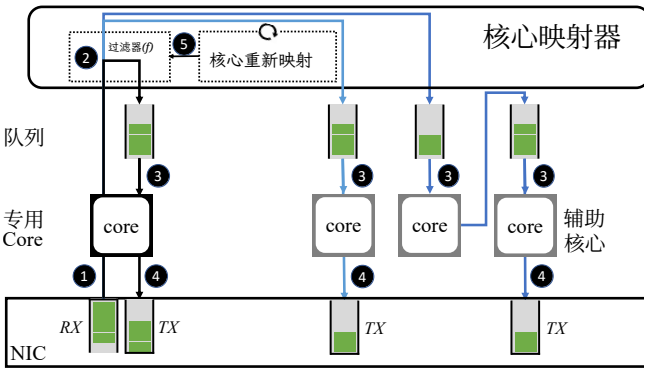


图2： 在每个周期内，NFV 运行时都会维护数据包和流的到达数量。周期结束时，核心映射器会标记可能发生的 SLO 违规。如果积压超过核心容量，核心映射器就会将队列拆分成多个较小的队列，并调配一些辅助核心来处理这些队列。

μs）。原则上，核心映射器类似于针对远程过程调用的微秒级核心分配 [11,27],，但流亲和性约束导致了截然不同的设计。

方法。核心映射器以微秒级的时间尺度持续执行以下操作：(a) 判断是否可能发生 SLO 违规；(b) 预测避免这些违规所需的额外核心容量；(c) 将部分流重新映射到一个或多个辅助核心。

判断是否可能发生 SLO 违规。给定 p99 延迟 SLO L，NetBlaze 必须以短于 L 的时间尺度监控流量（并重新映射流），才能检测到违规。时间尺度有多种选择，其中一种足够有效的选择是 $\frac{L}{2}$。这个时间间隔称为一个周期。例如，如果 L 为 200 μs，核心映射器就会采用 100 μs 的周期。在每个周期内，核心映射器力求维持以下不变条件：分配足够的 CPU 核心，以便在该周期内处理所有在周期开始前到达的流。满足这一条件后，它就能确保没有数据包在核心中停留超过两个周期，也就是不超过延迟 SLO。此类周期的时长比其他网络功能虚拟化系统的周期短 3 到 4 个数量级 [3,8,39]。

为了确定潜在的服务级别目标违规，在每个周期结束时，NetBlaze 会：(a) 统计该周期内的数据包数量和活动流数量；(b) 预测专用核心能否处理这些数据包和流，而不会导致服务级别目标违规。

确定队列占用情况。在每个专用核心上，NFV 运行时[6] 会持续从网卡队列中提取数据包，并跟踪数据包和流的到达情况（图 2）。它还会跟踪在一个周期内已处理的数据包和流。根据这些信息，它会估算当前周期内队列中的数据包数量和新流到达数量（即积压）。

预测核心使用率。利用这两个量，NetBlaze 判断专用核心是否能够处理积压。如果不能，则必须调度辅助核心。NetBlaze 训练

[6]运行时将待处理的数据包交付给网络功能链 ain.

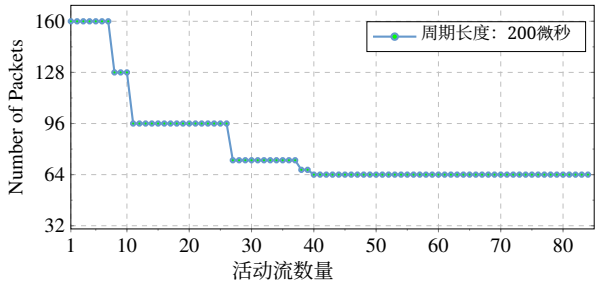


图 3： 测试网络功能链在一个周期内能够处理的活动流数量（至少有一个数据包到达）（f）与数据包数量（p）之间帕累托前沿的示例。随着f增加，为避免积压，网络功能链必须处理更少的数据包。

一个基于真实流量轨迹的数据驱动预测器。训练过程如下：NetBlaze 使用完整系统（包括入口映射器和服务器映射器）处理真实流量轨迹，但禁用核心映射器的流迁移组件（下文将对此进行讨论）。这会产生出现服务等级目标违规的周期。在每个此类周期中，它记录核心实际能够处理的流数量（f）和数据包数量（p）。

每个这样的元组 $\langle f, p\rangle$ 表示核心的处理能力，而元组集合构成一个帕累托前沿。任何位于该前沿之外的 $\langle f, p\rangle$ 都无法由该核心处理。例如，如果在训练过程中获得了元组 $\langle 55,64\rangle$，则该核心无法处理 $\langle 55,220\rangle$的积压；它必须调用一个或多个辅助核心来处理剩余积压。这一直觉构成了核心映射器的处理能力预测器的基础，该预测器用于确定专用核心能够处理多少积压。它还用于确定处理剩余积压所需的辅助核心数量。具体而言，它会使用预测器，反复将部分剩余积压分配给一个辅助核心，直到所有积压都已分配。图 3 展示了第 §4中使用的有状态网络功能链在周期长度为 200μs 时的帕累托前沿。[7]

因此，在此步骤结束时，核心映射器会生成一组 $\langle f'_i, p'_i\rangle$ 元组，将其分配给专用核心和每个辅助核心（下标i 表示对应的核心），使f' 和p' 的总和与积压相对应。在上面的示例中，原始积压可以拆分为三组流：$\langle 26,95\rangle$、$\langle 22,80\rangle$ 和 $\langle 7,45\rangle$，使得每组流都位于帕累托前沿内，并可由一个核心在一个周期内处理。

核心重新映射。现在，NetBlaze 必须将f'_i条流（包含p'_i 个数据包）迁移到第i个辅助核心。为此，它会从资源池中调配一个或多个辅助核心。对于每个辅助核心，核心映射器会迁移其积压中的一部分流，将其移至与该辅助核心关联的软件队列。要迁移一条流，核心映射器会创建一个与辅助核心关联的软件队列，并将数据包从自身队列移至该软件队列。随后，每个辅助

[7]帕累托前沿以 32 个数据包为间隔呈现离散跳跃，因为数据包是按批次处理的（最大批次大小为 32）

核心都会运行一个核心映射器，以避免后续周期中因流量突发而导致服务级别目标违规。

核心重新映射中棘手的一步是处理大流。大流是指其数据包积压如此之高，以至于单个核心无法在一个周期内将其处理完的流。由于 NetBlaze 必须维持流亲和性，因此只能拆分 网络功能：如果一条链包含 3 个网络功能，我们可以将最后 1 个或 2 个网络功能迁移到辅助核心。这样可以处理更多数据包，让 NetBlaze 清空积压。为此，针对大流的特定数据包积压p，NetBlaze 需要确定采用哪种拆分策略最合适。它通过对不同包数和不同拆分策略进行性能分析，开发另一个预测器来实现这一点。如果任何经过性能分析的拆分策略都无法满足数据包数量要求，NetBlaze 就会尽力处理该流（这可能导致服务级别目标违规），同时通知运维人员。

3.3 核心效率：服务器映射器

目标与挑战。 核心映射器每隔 100s 的 μs 做出决策，并能够处理这一时间尺度上的突发流量，但流速率和流数量可能在更长的时间尺度上发生变化。服务器映射器旨在通过持续重新分配流，将其调度到处理提供负载所需的最少数量的专用核心上，从而确保较高的核心效率。

如果我们能够在 μs 时间尺度内实现这一点，就不需要核心映射器了。然而，在这样的时间尺度内重新映射大型流量聚合并非易事：基于软件的转发（例如通过虚拟交换机）成本高昂且难以扩展，尤其是在网卡的线路速率达到 100 Gbps 时。基于硬件的方法依赖于对网卡或交换机中的流表进行重新编程，而这无法在 μs 时间尺度内完成。因此，我们选择采用分层设计：服务器映射器负责配置专用核心，而核心映射器则调配辅助核心，以清除短期积压。

服务器映射器面临的关键挑战是：(a) 快速估算服务提供负载所需的专用核心数量，以及 (b) 快速重新计算流到核心的映射关系。完成这些操作的时间尺度越短，NetBlaze 的 CPU 效率就越高。在我们当前的设计中，服务器映射器每 1 秒运行一次。更短的时间间隔并不能进一步提高系统的核心效率，反而可能导致 NetBlaze 不稳定，因为在较短的时间尺度下，数据包速率和流速率的测量可能变得不准确。

专用核心数量预测器。 为了预测专用核心的数量，NetBlaze 需要一个核心容量预测器。核心映射器也使用这种预测器，但它预测的是核心在较短时间间隔内能够清空的积压。与核心映射器不同，服务器映射器需要一种预测器，能够估算核心在较长时间尺度内、不过载的情况下可以处理的提供负载（活跃流数量 f 与聚合数据包速率 r的组合）。为此，对于各种 f 值，NetBlaze 会针对每条网络功能链及其关联的延迟 SLO，通过经验测定核心在不违反延迟 SLO 的情况下能够支持的数据包速率 r。

利用这些数据，给定聚合流数量 f 和聚合数据包数量 p，NetBlaze 会尽可能将这一提供负载分配到最少数量的专用核心上；下文将对此进行说明。

对于该预测器，NetBlaze 使用合成流量，而这种流量无法捕捉真实流量中的突发性。因此，服务器映射器的估计较为保守：它并不旨在保证尾延迟目标，而是依靠核心映射器按需适应突发流量。

快速流到核心的重新映射。这是服务器映射器的主要功能。服务器映射器知道每个活动专用核心上的流量聚合流数量 f' 和数据包速率r'。其核心容量预测器能够确定，对于给定的核心，该核心可以支持的流数量f和数据包速率r。随后，它会尝试将传入流量重新分配到尽可能少的专用核心上。

利用接收端扩展实现快速映射。 我们借鉴并改进了先前工作中的一种思路 [3, 8]。许多网卡支持接收端扩展（RSS）[15]，可将传入流量哈希到大量（例如，128 或 512 个）RSS桶中。每个桶又可以绑定到一个网卡队列，该队列分配给一个专用核心；来自该桶的流量将被导向该核心。通常，由于桶的数量多于核心数量，多个桶可以绑定到同一个核心。NetBlaze 利用这一观察结果，但采用了不同于 [3]的方法。

用于流量重新分配的快速启发式算法。对于每个 RSS 桶，NetBlaze 都会跟踪每个决策间隔内的平均活跃流数量和数据包速率。然后，对于每个核心，它都需要找到一组满足以下约束的桶：

- 这组桶仅分配给该核心。
- 每个桶的 $\langle f, r\rangle$ 元组之和（记为 $\langle f', r'\rangle$）不得超过预测处理能力 $\langle \hat{f}, \hat{r}\rangle$。

除此之外，服务器映射器还必须确保所有桶都分配给核心。可以将这个问题建模为一种优化形式。其目标是尽量减少专用核心的数量。遗憾的是，这种形式（为简洁起见，此处省略）是一个带约束的混合整数线性问题（MILP），使用商业求解器求解可能需要数百毫秒甚至更长时间。

相反，我们采用一种快速的贪心启发式算法，力求使用尽可能少的专用核心，同时对一个难以通过数学公式表达的关键约束进行建模：尽量减少桶到核心的映射变更次数。这样可以避免流频繁在核心之间迁移，因为这会影响缓存局部性，从而降低网络功能处理的性能。

该启发式算法会处理负载增加的核心（由于流到达或流量增加），也会处理负载降低的核心（由于流离开）：

- 它首先确定哪些专用核心过载（即，其 $\langle f', r'\rangle$ 超过核心容量）。对于每个过载的核心，它会确定需要迁移出去的最少桶数

以便迁移到其他专用核心，确保剩余流量不超过核心容量。它采用首次适配策略，将这些桶移至其他负载较低的核心。如果不存在这样的核心，则会分配一个新的专用核心。

• 然后，它从负载最轻的核心开始，尝试将其现有的桶迁移到其他核心，若成功则回收该核心。它会重复此过程，直到无法再回收任何核心。

NetBlaze 会先写入新的桶到核心映射表，然后再删除旧表，以确保不会丢弃任何数据包。由于网卡的限制，这一步可能需要长达几毫秒的时间 [3, 17], 。它保守地选择 1 秒的决策间隔，以尽可能减少开销，并避免桶到核心分配发生振荡。附录中的图 A.1展示了该算法。

专用核心的增强模式。在服务器映射器的周期性调用之间，专用核心可能会出现持续超过几个周期的大量积压。NetBlaze 必须处理此类过载，以尽量减少服务级别目标违规，因为专用核心在一个周期内只能分发有限数量的数据包。

为了解决这一问题，我们尝试按需调用服务器映射器（在两次周期性调用之间）。然而，正如上文所述，由于更新网卡硬件中的接收端缩放表需要毫秒级延迟，这仍然导致了较高的尾延迟。为避免这种情况，在此类过载情形下，NetBlaze 会在软件中为专用核心启用 *增强模式* 。在此模式下，专用核心会跳过网络功能处理，专注于从网卡队列拉取数据包并将其分发给辅助核心。为此，它会招募另一个核心代其执行网络功能处理。队列清空后，它便可恢复网络功能处理。评估结果表明，启用增强模式对于满足尾延迟服务等级目标是必要的（§4.5）。

3.4 服务器最小化：入口映射器

入口映射器将流分配给服务器，目标是尽量减少分配给网络功能链的服务器数量。一个机架可能会看到大量流，因此，如果入口映射器需要逐流做出服务器分配决策，就无法扩展。相反，入口映射器基于目的地址前缀做出分配决策。具体而言，它会将具有相同长度为 k 的目的地址前缀的流分配给同一台服务器。对于 IPv4，这需要 2^{32-k} 路由表项。k 可根据机架顶部交换机的表容量进行选择。

当流量到达一个先前未被占用 （即 尚未观察到任何数据包的）前缀时，系统会将该前缀分配给专用核心数量最多的服务器，前提是该数量低于阈值 τ。如果不存在符合条件的服务器，入口映射器便会从机架中选取一台新的服务器，并将流量分配给它。因此，只有当所有其他服务器的负载都接近目标负载时，才会启用新的服务器。参数 τ 确保每台服务器都有足够的辅助核心来处理突发流量。

每当专用核心的数量发生变化时，每个服务器映射器都会将该数量报告给入口映射器，后者据此引导前缀。NetBlaze 不会尝试整合专用核心数量较少的服务器。这需要进行跨服务器网络功能状态迁移，但也没有必要，因为我们假设任何服务器上的非专用核心都可用于运行边缘应用。

4 评估

在本节中，我们评估：NetBlaze 实现微秒级尾延迟服务级目标的能力（§4.2）、NetBlaze 相较于其他网络功能虚拟化系统的 CPU 核心效率（§4.3），以及其设计选择如何提升性能（§4.4-§4.5）。

实现。 NetBlaze 使用 BESS [5],这一款基于 DPDK 的高性能交换机 [10]。其设计由两部分组成：机架级入口和一个工作节点，共需 2.9k 行 C 代码++。前者负责从工作节点收集统计信息并运行入口映射器（§3.4）。后者包含服务器映射器（§3.3）、核心映射器（§3.2），以及其他功能，使专用核心能够将流量从网卡队列拉取到软件队列中，并招募辅助核心来处理突发流量。此外，NetBlaze 用于训练核心容量预测器的框架还增加了 1.2k 行 Python 代码。

4.1 实验设置与方法

测试平台。我们使用 Cloudlab [9]在单台机架顶端交换机下，由 5 台服务器组成的集群上运行实验。每台服务器配备两颗 AMD EPYC Milan CPU，每颗 CPU 有 32 个核心，主频为 2.8 GHz，并配有 256 GB ECC 内存（DDR4 3200 MHz）。为减少抖动，我们禁用了同时多线程和 CPU 频率调节。每台服务器配有一块用于数据流量的 100 GbE Mellanox ConnectX-6 网卡，以及一块用于控制和管理流量的独立网卡。

方法。在该集群上，我们运行 NetBlaze 和其他网络功能虚拟化系统，为处理真实世界流量的网络功能链提供服务，并测量端到端数据包处理延迟和 CPU 核心使用指标。我们采用两个来自已记录用例的典型有状态网络功能链 [21, 23]。

链 ***1*** 是一条流监控和负载均衡流水线：访问控制列表→负载均衡器→计费。链 ***2***是一条计算密集度更高的链，包含深度数据包检测和加密网络功能：访问控制列表→UrlFilter→Encrypt。这些网络功能和网络功能链经常用于评估网络功能虚拟化系统的性能 [18,36, 39]。访问控制列表强制执行 1.5k 条访问控制规则。负载均衡器是一个四层负载均衡器，可将流分配到一组后端服务器。计费是一种监控网络功能，用于跟踪每条流的数据使用量。UrlFilter 对流进行 TCP 重组，并应用复杂的字符串匹配规则（即Snort [45] 规则）来阻止提及禁用 URL 的连接。Encrypt 使用 128 位 ChaCha [4] 对每个数据包进行加密。网络地址转换维护可用四层端口列表，并为连接执行地址转换，分配空闲端口，并在连接的整个生命周期内维护该端口映射。

流量轨迹。我们的实验使用了两种真实流量轨迹[8]

[8]Wh虽然我们将 NetBlaze 用于边缘场景，但在评估中 我们使用

平台	延迟 SLO [μs]	p50 [μs]	p99 [μs]	Avg. 核心	Loss Rate
NetBlaze	*100*	28.8	89.4	34.6	0.006
	200	32.1	174.4	30.7	0.006
	300	30.7	238.8	30.1	0.005
	400	34.5	254.4	30.5	0.010
	500	31.1	298.2	29.7	0.005
Metron	-	57.7	2541	21.8	0.024
Quadrant	*100*	30.8	3051	45.3	0.062
	200	39.1	3021	43.3	0.057
	300	38.8	3015	43.3	0.056
	400	39.9	2988	43.1	0.052
	500	53.7	2994	41.7	0.044
Dyssect	*100*	63.0	78798	34.7	0.224
	200	68.0	68356	31.9	0.106
	300	78.1	70392	32.7	0.249
	400	76.2	64885	30.1	0.218
	500	68.7	71586	31.3	0.194

表3: 针对链 1，在 NetBlaze 和其他系统下，随系统延迟目标变化的端到端指标（p50 和 p99 延迟、时间平均 CPU 核心使用率及丢包率）对比。

平台	延迟 SLO [μs]	p50 [μs]	p99 [μs]	Avg. 核心	Loss Rate
NetBlaze	*100*	18.0	101.4	56.0	0.004
	200	22.8	180.3	41.9	0.010
	300	24.8	197.2	42.8	0.002
	400	24.9	210.3	40.3	0.007
	500	26.8	270.5	40.6	0.003
Metron	-	254.5	5580	32.1	0.044
Quadrant	*100*	34.0	5459	76.7	0.126
	200	34.7	5465	78.3	0.133
	300	34.0	5452	75.5	0.132
	400	34.5	5473	77.1	0.137
	500	34.2	5455	77.2	0.135
Dyssect	*100*	62.6	78955	50.6	0.294
	200	61.0	62528	45.6	0.246
	300	60.2	77793	47.4	0.279
	400	50.2	87130	48.2	0.208
	500	61.4	78030	44.8	0.214

<escape>**表4:** </escape>NetBlaze 与其他方法在骨干网流量下针对 **链 2** 的端到端指标（p50 和 p99 延迟、CPU 核心使用率的时间平均值以及丢包率）随系统延迟目标变化的对比。

(§2)。骨干网流量轨迹包含 490 万条流，流大小从 54 字节到 2.06 亿字节不等，最长持续时间为 52 秒。流到达率平均为每秒 3.12 万条，最高为每秒 5.12 万条；数据包大小平均为 988 字节，最大为 1514 字节。***AS*** 跟踪数据包含 160 万条流，大小从 54 字节到 5.14 亿字节不等，最长持续时间为 205 秒。流到达率平均为每秒 1.01 万条，最高为每秒 3.90 万条。数据包大小平均为 976 字节，最大为 1514 字节。因此，与骨干网流量轨迹相比，AS 跟踪数据的流大小和持续时间范围更广，但流速率略低。我们使用骨干网流量轨迹中 15 秒的流量训练核心容量预测器（§3.2，§3.3），并在其余部分进行测试。

我们在一台专用服务器上运行基于 BESS 的流量生成器。每次实验运行单一的网络功能链与流量轨迹组合。对于我们选择的网络功能链和流量轨迹，5 台服务器足以处理工作负载，因此我们据此选择集群规模。

指标。我们使用以下指标量化 NetBlaze 的性能：(a) 端到端延迟，采用两种度量方式：中位数（p50）和 p99 尾延迟；以及 (b) ***CPU*** 核心使用率，以总 CPU 小时或处理测试流量时的时间平均 CPU 核心数衡量。理想情况下，NetBlaze 应在确保达到目标 p99 延迟 SLO 的同时，保持较低的 CPU 核心使用率。

对比。我们将 NetBlaze 与三个先前提出的网络功能虚拟化系统进行比较，这些系统会根据流量动态变化自动扩缩容网络功能链。

Metron [18] 支持网络功能链的扩缩容。它支持在网卡上进行二层交换，并在机架顶部交换机上修改二层报头，以执行路由决策。它通过定期检查交换机上的丢包情况（借助数据包计数器）来检测过载核心，并将一个核心的流量聚合拆分为两个子集，当检测到核心过载时，将其中一个子集迁移到新的核心。Metron 不显式支持延迟服务级别目标。

Quadrant [39] 明确以第99百分位延迟服务等级目标为目标，通过集群级入口收集每个核心上的最坏情况数据包延迟，以此扩展网络功能链，并采用基于滞后的方法来控制运行网络功能链的每个核心的端到端延迟。研究表明，它能够满足合成流量的第99百分位延迟服务等级目标。真实流量轨迹中包含大量突发流量，正如我们将在下文展示的那样，这些突发流量会影响 Quadrant 的性能。

Dyssect [8] 旨在最小化服务网络功能链的第50百分位延迟。它使用接收端扩展在工作节点上扩展网络功能链，并定期使用混合整数线性规划公式将一部分流迁移到新的 CPU 核心。然而，Dyssect 并非为机架级系统设计；为了支持 Dyssect 的机架级实验，我们应用 NetBlaze 的机架级入口，将流量分发到 Dyssect 工作节点。

4.2 延迟性能比较

我们首先将 NetBlaze 的延迟与采用不同自动扩缩机制的其他网络功能虚拟化系统进行比较。

结果。 表格3、4、5和6展示了不同网络功能虚拟化平台在处理骨干网流量和自治系统流量时的第50百分位和第99百分位延迟。在所有平台中，NetBlaze 是唯一能够实现微秒级（100-500 μs）第99百分位延迟 SLO 的平台。所有平台都表现出亚毫秒级 p50 延迟，表明其扩缩容技术能够很好地限制中位延迟。NetBlaze 在实现这一点的同时，在所有系统中还具有最低的数据包丢失率（参见**丢包率** 列）。

Metron 不显式支持服务等级目标（因此表中只有一项），在处理链 1 和链 2 时，第99百分位延迟分别为 2541 μs 和 5580μs（骨干网流量），在处理 AS 流量时则分别为 1926 μs 和 5310 μs。此外，它在所有平台中的 p50 延迟最高，因为

骨干网和自治系统流量轨迹，因为几年后边缘流量可能会增加到这些水平。

平台	延迟 SLO [μs]	p50 [μs]	p99 [μs]	Avg. 核心	Loss Rate
NetBlaze	*100*	22.7	51.9	17.5	0.010
	200	22.9	52.1	17.4	0.011
	300	23.1	53.4	15.0	0.006
	400	23.4	53.2	16.2	0.013
	500	23.5	53.5	17.5	0.008
Metron	-	48.6	1926	11.9	0.014
Quadrant	*100*	37.2	1026.5	17.9	0.014
	200	30.7	930.0	17.3	0.010
	300	29.5	765.2	16.9	0.013
	400	29.3	556.5	17.3	0.011
	500	28.2	810.2	16.4	0.011
Dyssect	*100*	17.7	13586	17.9	0.044
	200	17.6	13031	17.2	0.054
	300	17.8	12672	17.9	0.050
	400	18.2	12480	15.9	0.052
	500	19.1	22776	16.5	0.049

表5： NetBlaze 和其他方案在不同系统延迟目标下运行**链 1** 并承载 **AS 流量**时的端到端指标比较（p50 和 p99 延迟、CPU 核心使用率的时间平均值及丢包率）。

平台	延迟 SLO [μs]	p50 [μs]	p99 [μs]	Avg. 核心	Loss Rate
NetBlaze	*100*	33.1	87.0	25.2	0.006
	200	41.6	103.3	18.1	0.008
	300	48.8	105.6	16.3	0.008
	400	50.0	108.0	18.4	0.007
	500	60.8	103.3	16.6	0.007
Metron	-	78.7	5310	14.8	0.029
Quadrant	*100*	70.4	5352	19.2	0.073
	200	69.9	5363	18.6	0.072
	300	70.5	5351	19.3	0.074
	400	70.4	5369	18.2	0.073
	500	69.9	5373	18.2	0.072
Dyssect	*100*	24.2	48041	18.4	0.056
	200	24.1	57273	18.1	0.049
	300	23.2	34573	16.9	0.041
	400	24.4	50812	17.0	0.040
	500	23.8	47229	16.7	0.044

表6： NetBlaze 与其他系统在 **链 2** 上运行 **AS 流量** 时的端到端指标（第50百分位和第99百分位延迟、时间平均 CPU 核心使用率以及丢包率）对比，随系统延迟目标变化。

它依靠硬件实现动态扩缩容。通过定期发送控制平面查询，在机架顶端交换机上收集数据包计数器统计信息来检测丢包，无法避免短时突发造成的高延迟。此外，重新编程交换机表（以拆分流量聚合）可能需要数百毫秒。

Quadrant 支持 p99 延迟 SLO 目标。然而，在我们的实验中，承载骨干网流量时，链 1 的 p99 延迟为 2988–3051 μs，链 2 的 p99 延迟为 5452–5473 μs。尽管 Quadrant 会主动监测每个核心最坏情况下的数据包处理延迟，但在高度突发的真实世界流量轨迹上，它仍无法实现良好的尾延迟表现，原因有三。1)扩缩容信号不当：对于每个核心，Quadrant 都会维护最坏情况下数据包处理延迟的移动平均值，并每隔数百毫秒将其更新至机架级入口。该信号不足以快速检测瞬时突发。2) 扩缩容延迟：Quadrant 通过一个机架级控制器（而不是由每个工作节点分别进行本地决策）做出扩缩容决策；该控制器每 50 毫秒将流从过载核心重新分配，并通过远程过程调用执行扩缩容，由此增加毫秒级延迟。3) 扩缩容不准确：Quadrant 在一个扩缩容周期内从一个核心迁移的流最多为一半。由于在其扩缩容周期内可能会有大量流到达，这种做法可能效果不佳。

Dyssect 支持 p50 延迟服务等级目标。以优化 p50 延迟为目标，它将流从核心迁移，以控制每个核心的负载，该负载定义为执行实际网络功能处理所占 CPU 时间的百分比。Dyssect 取得了良好的 p50 延迟结果：35 μs 和 60 μs（骨干网流量），但在所有方案中具有最高的 p99 延迟：链 1 和链 2 分别为 78 ms 和 87 ms（骨干网流量）。核心负载不足以作为维持低尾延迟的扩缩容信号。Dyssect 还采用 100-ms 的扩缩容周期，以及计算密集型的混合整数线性规划用于确定流迁移的公式化方法，以及其数据平面实现中的锁，这些都会增加延迟。

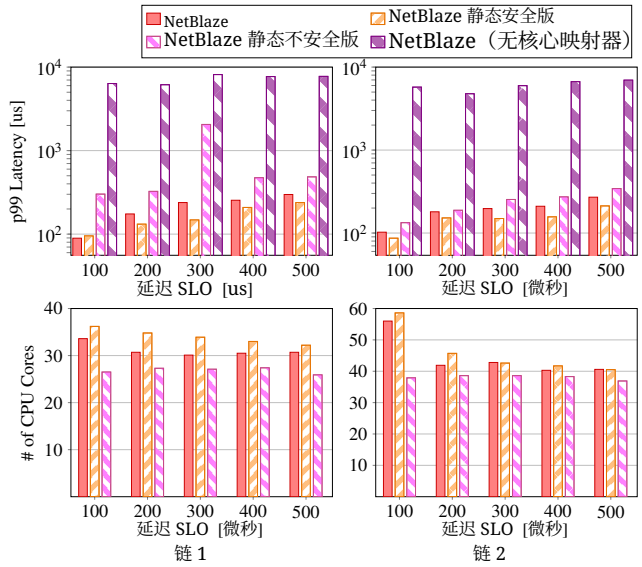
总体而言，对于所有系统，链 2 的 P99时延均高于链 1，因为前者的计算密集程度更高。唯一的例外是 Dyssect：对于骨干网流量轨迹，其两条链的 P99 时延相当，而该流量轨迹的流速率要高得多。

总之，我们发现，如果满足以下任一条件，现有系统就无法确保亚毫秒级尾延迟服务等级目标：（1）仅使用基于硬件的扩展；（2）没有快速检测和响应流量突发的机制。NetBlaze 采用混合扩缩容设计，并通过核心映射器快速处理瞬时突发。

4.3 CPU 核心使用率对比

为了比较不同网络功能虚拟化系统的 CPU 核心使用率，在每个 100-μs 周期内，我们记录网络功能虚拟化系统中的活跃 CPU 核心数量，并计算整个实验期间的时间平均核心使用率。

结果。 表 3、4、5、6 展示了 Net-Blaze 和其他网络功能虚拟化系统的 CPU 核心使用率。与其他网络功能虚拟化系统相比，Net-Blaze 的尾延迟低了 1 到 2 个数量级；此外，与 Quadrant 相比，在骨干网流量下，Net-Blaze 的 CPU 使用量最多可减少 34%（链 1）/48%（链 2），在 AS 流量下最多可减少 17%（链 1）/15%（链 2）；与 Dyssect 相比，在骨干网流量下最多可减少 14%（链 1）和 20%（链 2），在 AS 流量下最多可减少 16%（链 1）/11%（链 2）。Quadrant 监控每个核心的最坏情况数据包延迟，以做出流迁移决策。瞬时突发可能会增加最坏情况时延，并触发将流重新分配到新核心，从而增加核心使用率。Dyssect 的 CPU 核心使用率略优于 Quadrant，且与 Net-Blaze 相当。这是因为 Dyssect 使用 CPU 核心负载作为



图表 4： NetBlaze 与其他网络功能虚拟化系统所使用的时间平均 CPU 核心数随所达到的 p99 延迟变化的比较。

	平台	Avg. 核心	p50 [μs]	p99 [μs]
链 1	**Metron**	21.8	47.7	2540
	NetBlaze	23.9	51.3	2126
链 2	**Metron**	32.1	254.5	5580
	NetBlaze	30.4	374.2	4567

表7： NetBlaze 和 Metron 的端到端第50百分位和第99百分位延迟，以及按时间平均的 CPU 核心使用率对比。在这些实验中，我们分别将 NetBlaze 的延迟目标设为 2500 μs 和 5000 μs。NetBlaze 的 p99 延迟仍然更低，且 CPU 核心使用率相近。

其扩缩容信号，因此对突发流量不那么敏感。

与 Metron 相比，NetBlaze 使用的 CPU 核心略多（10–26%），但 p99 延迟低了一个数量级；这是因为 Metron 采用了高效的基于硬件的扩缩容。虽然 Metron 致力于在扩缩容期间最大限度地减少数据包丢失，但它并未积极优化端到端延迟（因此 p50 和 p99 延迟较高）。相比之下，NetBlaze 的 p99 延迟低了一个数量级；它通过使用核心容量预测器将流量分配给核心来避免数据包丢失，并且丢包率更低。

为了更好地比较 Metron 和 NetBlaze 的核心使用率，我们将 NetBlaze 的目标 p99 延迟 SLO 设为 Metron 达到的 p99 延迟。表 7 显示，即使延迟 SLO 较宽松，NetBlaze 仍能实现更低的 p99 延迟，并且与 Metron 相比，使用数量相当的 CPU 核心。在该实验中，NetBlaze 将流迁移到辅助核心的积极性较低，因此使用的核心更少。

4.4 消融研究：核心映射器

NetBlaze 的核心映射器会响应工作节点上每个专用核心的突发流量，并使用流和数据包计数来做出流到核心的映射决策。为了理解.....的重要性

针对这些设计选择，我们将 NetBlaze 与 3 种不同的变体进行比较：1) 静态安全：无论流到达数量如何，始终使用最小的数据包数量；2) 静态不安全：无论流到达数量如何，始终使用最大的数据包数量；3) 不使用核心映射器：禁用核心映射器的 NetBlaze 变体。

结果。 图4 表明，只有 NetBlaze 使用其核心映射器或采用 静态安全核心映射器时，才能提供亚毫秒级延迟 SLO。若不使用核心映射器，NetBlaze 就无法实现亚毫秒级延迟。使用 静态不安全核心映射器时，NetBlaze 可以实现亚毫秒级 p99 延迟，但在某些情况下会违反 SLO。 静态安全 变体在大多数情况下可实现更低的 p99 延迟，但与 NetBlaze 的设计相比，最多可能多使用 13% 的 CPU 核心，因此我们采用后者（§3.2）。

4.5 消融研究：服务器映射器

NetBlaze 的服务器映射器会针对持续负载变化，在 NetBlaze 工作节点上找出所需专用核心数最少的配置。与核心映射器一样，我们探索了两种变体：静态安全，根据最低数据包计数分配核心；以及静态不安全，根据最高数据包计数分配核心。

结果。 图 5 显示，三种设计都能够以相当的 CPU 核心使用率满足链 1 的亚毫秒级第99百分位延迟服务等级目标。对于链 2，NetBlaze 的设计在延迟和 CPU 核心使用率方面都占据优势，这表明，在某些设置下，服务器映射器必须考虑流和数据包速率，才能高效地满足服务等级目标。

增强模式的必要性。 NetBlaze 的增强模式不会在持续过载时重新配置 RSS 表，而是使用专用核心清空数据包队列，并将网络功能处理迁移到其他核心（§3.3）。为展示启用增强模式的重要性，我们将 NetBlaze 与两个不同的变体进行比较：1）不启用增强模式：禁用增强模式的 NetBlaze 变体；2）按需调用：该变体不启用增强模式，而是在出现持续过载时立即调用服务器映射器的基于 RSS 的流到核心重新映射。

结果。图6 显示，启用增强模式对于实现亚毫秒级尾延迟服务等级目标至关重要。其他两种 Net- Blaze 变体只能达到毫秒级尾延迟；前者是因为无法处理秒级时间尺度上的负载变化，后者则是因为重新编程接收端缩放表会产生开销。

4.6 分析调度开销

入队和出队开销。 在 NetBlaze 中，专用核心从网卡队列中取出数据包，并在处理之前将其入队到软件队列（由该核心或辅助核心拥有）。稍后会将数据包出队并进行处理。现在我们来量化这项入队/出队开销（NetBlaze 使用无锁队列）。

结果。通过计算 1 万次入队操作的平均周期开销

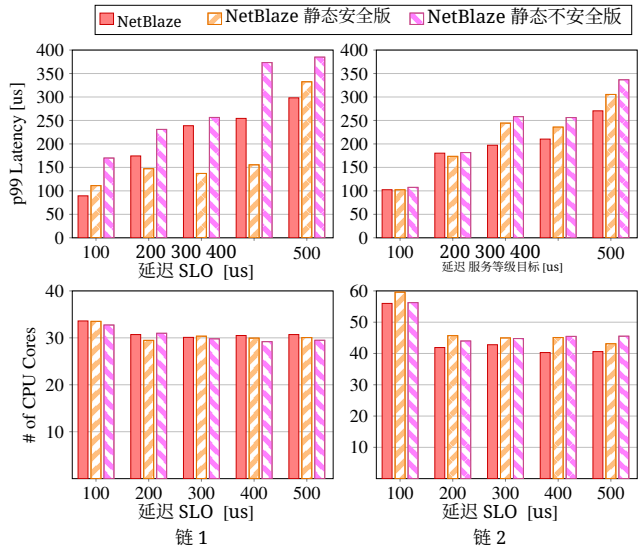


图 5： NetBlaze 与其他网络功能虚拟化系统所使用的时间平均 CPU 核心数比较，横轴为实现的第99百分位延迟。

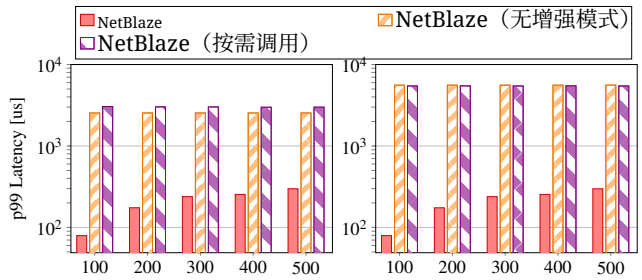


图6： NetBlaze 与其他网络功能虚拟化系统所用 CPU 核心数的时间平均值比较，随实现的 p99 延迟变化。

和出队操作，我们发现：**入队** 每批次增加 40 个周期；**出队** 每批次增加 32 个周期。与网络功能链的数据包处理时间相比，这项开销很小。

核心重映射开销。 在每个短周期结束时，NetBlaze 的核心映射器会触发核心重新映射，以平衡软件队列上的工作负载，并在必要时招募辅助核心。该核心重新映射包括决策和执行过程，二者会扫描软件队列（由专用核心负责）的所有数据包。

结果。 表 8 展示了不同数据包队列长度下核心重新映射处理的执行时间基准测试结果。如图所示，即使数据包超过 1k，核心重新映射过程也能在 5 μs 内完成。实际上，NetBlaze 可以通过其服务器映射器设计控制软件队列长度，因此预计软件队列中的数据包会少得多。对于 100s-μs 延迟 SLO，这一开销相对较小。若要将 NetBlaze 扩展至更严格的 SLO，可能需要重新设计；我们将此留待未来工作。

5 相关工作

NetBlaze 的独特之处在于，它能够在集群环境中以较高的 CPU 核心效率，为网络功能链实现微秒级服务等级目标，但

数据包队列长度	核心重新映射进程 [周期]	核心重新映射处理 [μs]
128	4.2k	1.5
256	7.1k	2.6
512	7.7k	2.8
1024	12.2k	4.4

表8： NetBlaze 核心映射器的核心重新映射过程的平均执行时间随数据包队列长度的变化。该过程在每个短周期结束时运行，且开销很低（即 $< 5\mu s$ 即可重新映射超过 1k 个数据包）。

借鉴了文献中多个研究领域的相关成果。

CPU 调度。 近期有一系列研究探索了面向数据中心应用的微秒级远程过程调用。ZygOS 和 RackSched [32,43] 研究了在服务器/机架环境中处理 RPC 请求的调度策略；Arachne [33] 允许应用根据负载管理核心；Shenango [27]将延迟敏感型与批处理应用共置，以提高 CPU 效率；Caladan [11] 则减轻共置任务之间的干扰。近期的一项后续研究 [25] 探索了面向微秒级 RPC 的负载均衡与扩缩容的最优组合。NetBlaze 借鉴了这类关于延迟与效率权衡的研究，但针对的是不同的环境：网络功能虚拟化，其中有状态网络功能的流亲和性要求会带来性质截然不同的解决方案。

数据包调度。 Fastpass [30] 采用集中式数据包级调度，实现“零队列”网络。Net-Blaze 针对每个工作节点的网卡队列进行优化，而非交换机队列，并且面向数据包速率和流速率更高的流量，因此需要采用分层设计，而非集中式设计。

网络功能虚拟化。 先前的研究可分为四类。网络功能虚拟化框架 [13,14,28,29,31,34,39] 致力于解决网络功能的开发、链式连接和隔离问题。尽管 NetBlaze 的研究重点有所不同，但它可以利用这些机制来连接和隔离网络功能。在性能优化方面，一系列论文提出了相关机制，以高效管理有状态网络功能的网络功能状态 [6,16,40],，并优化网络功能在共享资源池上运行时的资源分配 [38],，以及最大限度地减少跨核心连接网络功能时的丢包 [20]。研究人员一直在探索将网络功能处理卸载到硬件上的方法，包括智能网卡、OpenFlow、PISA 交换机、图形处理器、现场可编程门阵列等 [18,19,22,41,42]。最后，为了实现实际的网络功能虚拟化部署，学术界一直在探索如何在集群中部署网络功能虚拟化，以服务动态流量，这涉及负载均衡和扩缩容。这些方案 [3,8,18,36,39] 具有一定的参考价值，但不足以实现有状态网络功能链的微秒级服务等级目标（如 §4所示）。

6 结论

本文指出网络功能虚拟化中的一个关键问题：如何在保持较高 CPU 核心效率的同时，实现微秒级尾延迟服务级目标。我们的流量轨迹分析揭示了真实流量中的两种突发类型，并表明网络功能虚拟化系统必须妥善处理

这些突发，才能实现这一目标。我们认为，网络功能虚拟化系统需要快速适应瞬时流量突发。为此，我们提出了一套扩展机制：核心映射器能够吸收每个 CPU 核心上的瞬时突发，而服务器映射器则通过将流分配并迁移至尽可能少的 CPU 核心，确保较高的 CPU 效率。我们的网络功能虚拟化系统 NetBlaze 能够满足亚毫秒级尾延迟 SLO，而当今其他网络功能虚拟化系统则无法做到这一点。

参考文献

[1] 亚马逊云科技（AWS）。AWS Marketplace：F5 分布式云服务。可访问：https://aws.amazon.com/marketplace/pp/prodview-cn54su2icbkkc，2022年。

[2] Microsoft Azure。Microsoft Azure：混合云和多云解决方案。可在以下网址查阅：https://azure.microsoft.com/en-us/solutions/hybrid-cloud-app/#overview，2022年。

[3] Tom Barbette、Georgios P. Katsikas、Gerald Q. Maguire 和 Dejan Kosti´c。Rss++：负载和状态感知的接收端扩展。载于第*15*届新兴网络实验与技术国际会议论文集，CoNEXT '19，第 318–333 页，纽约，纽约州，美国，2019年。计算机协会。

[4] Daniel J Bernstein 等人。ChaCha，Salsa20 的一种变体。载于*SASC*研讨会论文集，第 8 卷，第 3–5 页。瑞士洛桑，2008年。

[5] BESS（伯克利可扩展软件交换机）。BESS 的官方 GitHub 仓库：https://github.com/NetSys/bess，2022年。

[6] Anat Bremler-Barr、Yotam Harchol 和 David Hay。Openbox：用于开发、部署和管理网络功能的软件定义框架。载于*2016*年*ACM SIGCOMM*会议论文集，SIGCOMM '16，第511–524页，纽约，美国纽约州，2016年。计算机协会。

[7] CAIDA UCSD匿名互联网流量跟踪数据。可通过以下网址获取 http://www.caida.org/data/passive/passive_数据集。xml，2019年。

[8] Fabricio B. Carvalho、Ronaldo A. Ferreira、Italo Cunha、Marcos A. M. Vieira 和 Murali K. Ramanathan。《Dys-sect：有状态网络功能的动态扩展》。载于*IEEE*国际计算机通信会议论文集，INFOCOM '22，2022年。

[9] Dmitry Duplyakin、Robert Ricci、Aleksander Mar- icq、Gary Wong、Jonathon Duerig、Eric Eide、Leigh Stoller、Mike Hibler、David Johnson、Kirk Webb、Aditya Akella、Kuangching Wang、Glenn Ricart、Larry Landwe- ber、Chip Elliott、Michael Zink、Emmanuel Cecchet、Snigdhaswin Kar 和 Prabodh Mishra。《云实验室的设计与运行》。载于*2019*年 *USENIX* 年度技术大会（*USENIX ATC 19*），第 1–14 页，华盛顿州伦顿，2019年7月。USENIX 协会。

[10] Linux基金会。数据平面开发套件（DPDK）。DPDK 官方网站：http://www.dpdk.org，2015年。

[11] Joshua Fried、Zhenyuan Ruan、Amy Ousterhout 和 Adam Belay。《 Caladan：在微秒级时间尺度缓解干扰》。载于第 *14* 届 *USENIX* 研讨会 *O perat ing S* 系统 *Desi gn an d Impl ementat ion*（*OSDI 20*），第281–297 页。USENIX 协会，2020年11月。[12] Google Cloud Platform（GCP）。推出 Google Distributed Cloud。可访问：https://cloud.google.com/blog/topics/hybrid-cloud/announcing-google-distributed-cloud-edge-and-hosted，2022年。

[13]Aaron Gember-Jacobson、Raajay Viswanathan、Chaithan Prakash、Robert Grandl、Junaid Khalid、Sourav Das 和 Aditya Akella。OpenNF：推动网络功能控制创新。载于*2014* 年 *ACM SIGCOMM* 会议论文集，SIGCOMM '14，第 163–174 页，纽约，美国，2014 年。计算机协会。

[14] Jinho Hwang、K. K. Ramakrishnan 和 Timothy Wood。NetVM：在商用平台上利用虚拟化实现高性能和灵活网络。发表于 第 *11* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 14*），第 445–458 页，西雅图，华盛顿州，2014 年 4 月。USENIX 协会。

[15] Intel 公司接收端扩展（RSS）。提升多核系统中的网络性能。可访问：http://www.intel.com/content/dam/support/us/en/documents/network/sb/318483001us2.pdf，2016 年。

[16] Murad Kablan、Azzam Alsudais、Eric Keller 和 Franck Le。无状态网络功能：打破状态与处理之间的紧密耦合。载于第 *14* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 17*），第 97–112 页，美国马萨诸塞州波士顿，2017 年。USENIX 协会。

[17] Georgios P. Katsikas、Tom Barbette、Marco Chiesa、De- jan Kosti´c 和 Gerald Q. Maguire。关于（智能）网络接口卡，你需要了解的内容。载于 Oliver Hohlfeld、Andra Lutu 和 Dave Levin （编辑），被动与主动测量，第 319–336 页，Cham，2021 年。Springer International Publishing。

[18] Georgios P. Katsikas、Tom Barbette、Dejan Kosti´c、Rebecca Steinert 和 Gerald Q. Maguire Jr. Metron：以底层硬件的真实速度运行 NFV 服务链。载于第 *15* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 18*），第 171–186 页，华盛顿州伦顿，2018 年 4 月。USENIX 协会。

[19] Joongi Kim、Keon Jang、Keunhong Lee、Sangwook Ma、Junhyun Shim 和 Sue Moon。NBA（网络均衡机制）：面向异构处理器的高性能数据包处理框架。载于第十届欧洲计算机系统会议论文集，Eu- roSys '15，美国纽约，2015年。计算机协会。

[20] Sameer G. Kulkarni、Wei Zhang、Jinho Hwang、Shri-ram Rajagopalan、K. K. Ramakrishnan、Ti mothy Wood、Mayutan Arumaithurai 和 Xiaoming Fu。Nfvnice：面向 NFV 服务链的动态反压与调度。载于*ACM*数据通信特别兴趣小组会议论文集，SIG- COMM '17，第 71–84 页，美国纽约，2017年。ACM。

[21] Surendra Kumar、Mudassir Tufail、Sumandra Majee、Claudiu Captari 和 Shunsuke Homma。《数据中心中的服务功能链用例》。互联网草案 draft-ietf-sfc-dc-use-cases-06，互联网工程任务组，2017 年 2 月。进行中。

[22] Yanfang Le、Hyunseok Chang、Sarit Mukherjee、Limin Wang、Aditya Akella、Michael M. Swift 和 T. V. Lakshman。*UNO*：统一主机与智能网卡卸载，实现灵活的数据包处理，第 506–519 页。计算机协会，纽约，纽约州，美国，2017 年。

[23] Tamás Lévai、Felicián Németh、Barath Raghavan 和 Gabor Retvari。Batchy：面向服务级目标的数据流图批量调度。载于第 *17* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 20*），第 633–649 页，圣克拉拉，加利福尼亚州，2020 年 2 月。USENIX 协会。

[24] MAWI工作组流量档案：WIDE 至上游互联网服务提供商的中转链路每日流量轨迹。可通过以下网址获取：http://mawi.wide.ad.jp/mawi/，2022年9月。

[25] Sarah McClure、Amy Ousterhout、Scott Shenker 和 Sylvia Ratnasamy。《面向微秒级任务的高效调度策略》。载于第*19*届 *USENIX* 网络系统设计与实现研讨会（*NSDI 22*），第1–18页，美国华盛顿州伦顿，2022年4月。USENIX 协会。

[26]Nitinder Mohan、Lorenzo Corneo、Aleksandr Za-vodovski、Suzan Bayhan、Walter Wong 和 Jussi Kangasharju。使用延迟剪枝精简边缘研究。载于 Ben Zhao、Heather Zheng、Harsha V. Madhyastha 和 Venkat N. Padmanabhan（编），*HotNets '20*：第 *19* 届 *ACM*网络热点主题研讨会，线上活动，美国，*2020* 年 *11* 月 *4–6* 日，第 182–189 页。ACM，2020 年。

[27] Amy Ousterhout、Joshua Fried、Jonathan Behrens、Adam Belay 和 Hari Balakrishnan。《Shenango：为延迟敏感型数据中心工作负载实现高 CPU 效率》。载于第 *16* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 19*），第 361–378 页，美国马萨诸塞州波士顿，2019年2月。USENIX 协会。

[28] Shoumik Palkar、Chang Lan、Sangjin Han、Keon Jang、Aurojit Panda、Sylvia Ratnasamy、Luigi Rizzo 和 Scott Shenker。《E2：NFV 应用框架》。载于第 *25* 届操作系统原理研讨会论文集，SOSP '15，第 121–136 页，美国纽约，2015年。计算机协会。

[29] Aurojit Panda、Sangjin Han、Keon Jang、Melvin Walls、Sylvia Ratnasamy 和 Scott Shenker。《Netbricks：摆脱 NFV 中的 V》。载于第 *12* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 16*），第 203–216 页，美国佐治亚州萨凡纳，2016 年 11 月。USENIX 协会。[30] Jonathan Perry、Amy Ousterhout、Hari Balakrishnan、Devavrat Shah 和 Hans Fugal。《Fastpass：一种集中式"零队列"数据中心网络》。载于*2014* 年 *ACM SIGCOMM* 会议论文集，SIGCOMM '14，第 307–318 页，美国纽约，2014年。计算机协会。[31] Rishabh Poddar、Chang Lan、Raluca Ada Popa 和 Sylvia Ratnasamy。《SafeBricks：为云中的网络功能提供防护》。载于第 *15* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 18*），第 201–216 页，美国华盛顿州伦顿，2018 年 4 月。USENIX 协会。[32] George Prekas、Marios Kogias 和 Edouard Bugnion。《Zygos：为微秒级网络任务实现低尾延迟》。载于第 *26* 届操作系统原理研讨会论文集，SOSP '17，第 325–341 页，美国纽约，2017年。计算机协会。[33] Henry Qin、Qian Li、Jacqueline Speiser、Peter Kraft 和 John Ousterhout。《Arachne：核心感知型线程管理》。载于第 *13* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 18*），第 145–160 页，美国加利福尼亚州卡尔斯巴德，2018 年 10 月。USENIX 协会。[34] Yuxin Ren、Guyue Liu、Vlad Nitu、Wenyuan Shao、Riley Kennedy、Gabriel Parmer、Timothy Wood 和 Alain Tchana。《面向可扩展、动态多租户边缘云的细粒度隔离》。载于*2020* 年 *USENIX* 年度技术大会（*USENIX ATC 20*），第 927–942 页。USENIX 协会，2020 年 7 月。[35] Justine Sherry、Shaddi Hasan、Colin Scott、Arvind Krishnamurthy、Sylvia Ratnasamy 和 Vyas Sekar。《让中间盒成为别人的问题：作为云服务的网络处理》。载于*ACM SIGCOMM 2012* 网络通信应用、技术、架构与协议会议论文集，SIGCOMM '12，第 13–24 页，美国纽约，2012年。计算机协会。

[36] Arjun Singhvi、Junaid Khalid、Aditya Akella 和 Sujata Banerjee。《SNF：无服务器网络功能》。载于第 *11* 届 *ACM* 云计算研讨会论文集，SoCC '20，第 296–310 页，美国纽约，2020年。计算机协会。

[37] Chen Sun、Jun Bi、Zhilong Zheng、Heng Yu 和 Hongxin Hu。Nfp：在网络功能虚拟化中实现网络功能并行化。载于*ACM*数据通信特别兴趣小组会议论文集，SIGCOMM '17，第 43–56 页，纽约，纽约州，美国，2017年。计算机协会。

[38] Amin Tootoonchian、Aurojit Panda、Chang Lan、Melvin Walls、Katerina Argyraki、Sylvia Ratnasamy 和 Scott Shenker。Resq：在网络功能虚拟化中实现服务等级目标。载于第 *15* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 18*），第 283–297 页，伦顿，华盛顿州，2018年4月。USENIX 协会。

[39] Jianfeng Wang、Tamás Lévai、Zhuojin Li、Marcos A. M. Vieira、Ramesh Govindan 和 Barath Raghavan。Quadrant：可部署于云端的网络功能虚拟化平台。载于第 *13* 届云计算研讨会论文集，SoCC '22，第 493–509 页，纽约，纽约州，美国，2022年。计算机协会。

[40] Shinae Woo、Justine Sherry、Sangjin Han、Sue Moon、Sylvia Ratnasamy 和 Scott Shenker。《有状态网络功能的弹性扩缩容》。载于第 *15* 届 *USENIX* 网络系统设计与实现研讨会（*NSDI 18*），第 299–312 页，华盛顿州伦顿，2018 年。USENIX 协会。

[41] Jane Yen、Jianfeng Wang、Sucha Supittayapornpong、Marcos A. M. Vieira、Ramesh Govindan 和 Barath Raghavan。《满足跨平台网络功能虚拟化中的服务等级目标》。载于第 *16* 届新兴网络实验与技术国际会议论文集，CoNEXT '20，第 509–523 页，美国纽约州纽约，2020 年。计算机协会。

[42] Zhipeng Zhao、Hugo Sadok、Nirav Atre、James C. Hoe、Vyas Sekar 和 Justine Sherry。《在单台服务器上实现 100 Gbps 入侵防御》。载于第 *14* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 20*），第 1083–1100 页。USENIX 协会，2020年11月。

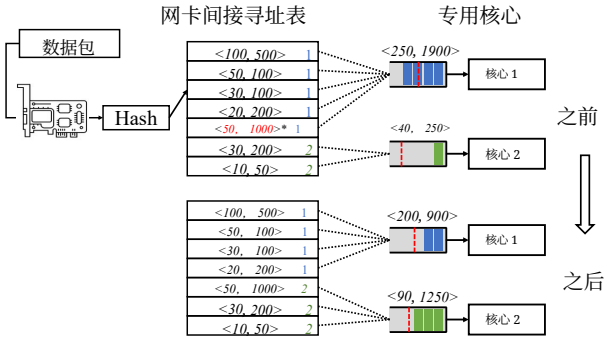
[43] Hang Zhu、Kostis Kaffes、Zixu Chen、Zhenming Liu、Christos Kozyrakis、Ion Stoica 和 Xin Jin。RackSched：用于机架级计算的微秒级调度器。载于第 *14* 届 *USENIX* 操作系统设计与实现研讨会（*OSDI 20*），第 1225–1240 页。USENIX 协会，2020年11月。

附录

A 服务器映射器的启发式算法

NetBlaze 的服务器映射器旨在通过持续重新分配流量，将流分配到处理当前流量需求所需的最少专用核心上，从而确保高核心效率。

NetBlaze 使用网卡 RSS 技术快速将流聚合映射到核心。我们不将其建模为受约束的混合整数线性问题（混合整数线性规划），而是提出一种用于流量重新分配的快速贪心启发式算法。图 A.1给出了一个将 RSS桶重新映射到核心的示例，以演示该算法。



图A.1： 服务器映射器利用接收端扩展（RSS），并更新网卡的入方向表，以应用桶到核心的映射。在每个决策间隔内，它会跟踪每个接收端扩展桶对应的流数量和数据包速率 $<f, r>$，每个桶对应网卡表中的一个条目。间隔结束时，服务器映射器会决定服务器上的专用核心数量：它会找出负载超过核心处理能力的过载核心，将其中最少数量的桶移至其他核心，然后尝试回收核心。在上述情况中，一个桶从核心 1 迁移到核心 2，以避免过载。