arXiv:2601.09081v1 [cs.DS] 14 Jan 2026

面向高速网络接口卡定时器管理、支持动态更新的分组排序队列

Zekun Wang、Binghao Yue、Weitao Pan、Jianyi Shi、Yue Hao

摘要—**随着网络功能由硬件卸载，网络接口卡（NIC）承担起大量有状态、高精度和高吞吐量任务，其中定时器是关键的支撑组件。然而，现有定时器管理方案存在软件负载重、精度低、缺乏硬件更新支持以及溢出等问题。本文提出了两种用于优先级队列的新操作——更新和分组排序，以实现硬件定时器管理。据我们所知，本研究首次提出一种硬件优先级队列，通过组合和传播基本操作来修改队列中元素的优先级，从而支持更新操作。分组排序机制通过设定分组边界优先级来改变排序过程和元素插入位置，确保溢出后仍能正确计时。我们采用一维（1D）脉动阵列与移位寄存器相结合的混合架构实现设计，并通过面向流表超时管理的数据包级仿真进行验证。结果表明，在28纳米工艺下，深度为4K、位宽为16位的定时器队列可实现超过500兆赫（175百万包每秒，精度为12纳秒）的性能；在现场可编程门阵列上则可实现超过300兆赫（116百万包每秒）。值得注意的是，与现有设计相比，该设计的查找表和触发器用量分别减少了31%和25%**。

索引词—**优先级队列、定时器队列、脉动阵列**。

I. 引言

IN 在现代高性能网络架构中，定时器是协议栈管理、资源回收和可靠性保障中广泛使用的基础机制。

定时器对于软件定义网络（SDN）中的流表管理、远程直接内存访问（RDMA）中的队列对（QP）维护，以及 TCP 卸载引擎（TOE）中的重传不可或缺；它们既需要具备足够的精度以确保协议正确性，也需要支持动态更新以适应网络变化。

在 SDN 中，控制平面与数据平面相分离[1]。控制平面通过南向接口部署转发策略，这些策略以流规则的形式在交换机上实现 [2]。OpenFlow 是一种标准的南向协议，它通过两个参数定义流规则的到期时间：空闲超时和硬超时。它还提供了三种流表管理机制 [3]，即：流过期超时、主动流表项删除和流表项驱逐。常见的软件交换机（例如 Open vSwitch [4]）主要依靠 CPU 和软件定时器管理流表项超时。为合理配置流表项超时并大幅提升流表利用率，[5] 和 [6] 均针对空闲超时提出了结合动态超时调整与主动驱逐策略的方法。同一流的数据包到达时，超时值便会更新；当流表利用率超过阈值时，则删除特定表项。尽管利用率有所提升，这种基于软件的方法仍存在计时准确性低的问题，且超时更新需要软件参与，导致 CPU 和内存使用率较高，其中 CPU 使用率超过 80% [6]。因此，基于软件的动态定时器管理开销相对较高。

在RDMA技术中，可靠连接（RC）服务是其基本服务类型之一，而定时器构成了该服务的关键组件 [7]。可靠传输机制以QP为粒度进行管理；为每个QP配置专用定时器，以实现重传机制。随着QP活动量增加，随着连接数量的增长，RDMA性能会下降 [8]。将大部分重传机制卸载到硬件网卡可以解决这一限制，从而提升连接可扩展性 [9]。然而，传统的硬件定时器管理通常采用单周期检查方法 [10],[11]。该方法的一个主要缺点是，随着定时器规模扩大，计时精度会降低，因此难以应用于高速网络场景。

TCP可靠服务的关键保障之一同样是重传定时器 [12],，该定时器用于检测数据传输期间的响应丢失。 [13] 表明，根据往返时间（RTT）合理设置重传超时（RTO），有助于解决大量并发流的重传导致的链路空闲问题。随着将TCP功能卸载到网卡的TOE技术发展 [14],，在硬件层面收集RTT信息并配置硬件重传定时器，将有效提升重传效率。此外，更细的计时粒度也有利于恢复重传并准确测量时间间隔。然而，常见的SmartNIC通常缺少内置定时器，而TOE技术中的定时器往往由软件结合智能网络接口卡在软件层面实现 [15],[16]。

从上述讨论的典型定时器应用场景来看，当前大规模定时器

稿件创建于2025年10月；本工作得到国家重点研发计划（2023YFB4405100, 2023YFB4405102）资助。（通讯作者：Weitao Pan。）

Zekun Wang、Jianyi Shi 和 Yue Hao 就职于西安电子科技大学微电子学院，地址：中国西安 710126（电子邮箱：zekun-wang@stu.xidian.edu.cn；jyshi@mail.xidian.edu.cn；yhao@xidian.edu.cn）。Weitao Pan 和 Binghao Yue 就职于西安电子科技大学综合业务网国家重点实验室，地址：中国西安 710071（电子邮箱：wt-pan@mail.xidian.edu.cn；23011210684@stu.xidian.edu.cn）。

管理存在的不足可以总结如下：

- 基于软件的动态定时器管理开销相对较高；
- 硬件定时器的精度受规模扩展的限制；
- SmartNIC 缺乏高效的硬件定时器。

在众多定时器管理方法中，优先级队列具有可扩展性，且计时精度不受规模影响。然而，该方法依赖递增的外部参考计时器为目标任务设置到期时间。受硬件固定位宽的限制，定时器溢出不可避免，进而导致时序错误；迄今尚未找到最优解决方案。为实现支持动态更新的硬件定时器队列，并解决溢出引起的时序错误，我们提出了更新操作和分组排序。该设计基于一维脉动阵列与移位寄存器的混合架构实现，最终应用于流表项超时管理。代码和测试环境的 GitHub 链接将在最终稿中提供。

本文的主要贡献如下：

- 从移位角度出发，我们分析了基本操作的实现。通过组合并传播这些基本操作，我们实现了一种更新操作，可在不影响队列排序结果的情况下，对队列中已有元素进行动态更新；
- 提出一种分组排序机制，支持在入队过程中动态修改搜索逻辑。该方法利用定时器的最高有效位（MSB）作为分组标志，确保溢出后仍具有计时准确性，同时硬件开销较低；
- 采用一维脉动阵列与移位寄存器的混合架构来实现硬件优先级队列。计时精度可配置，最高可达 12 ns；
- 基于28纳米工艺库，在支持4K深度和64K个优先级的情况下，该设计的运行频率可超过500兆赫，吞吐量最高可达175百万包每秒；在现场可编程门阵列上实现时，运行频率可达345兆赫，吞吐量为116百万包每秒。

II. 相关工作

对于大规模定时器队列的管理，常见的方案有三种：单周期检查方法、有序列表法和多级队列法。其中，有序列表法还包括时间轮和优先级队列。本节将分析这三种方法，以及相关硬件优先级队列的实现。

单周期检查方法会在一个计时精度间隔内遍历所有定时器，并将它们的时间值减 1。当定时器降至 0 时，就会触发相应事件。由于遍历时间与定时器规模有关，如果系统包含大量定时器，计时精度将受到严重影响。Jingzhao RDMA NIC [10] 采用硬件定时器进行重传，并通过遍历方案管理这些定时器，定时器更新则通过重新加载实现。修改单个定时器需要 2 个周期；在基于 28 纳米工艺、工作频率为 450 兆赫的条件下，遍历 16K 个定时器需要 72 微秒。ERNIC [11] 支持基于硬件的自动数据包重传，并以相同方式管理硬件定时器。扫描所有 QP 定时器必须在 6554 个周期内完成。此外，还配置了一个快速时钟定时器，用于补偿“短时间内反复重新加载”导致的时序错误，从而确保计时精度。尽管如此，该方案仍面临可扩展性瓶颈，且计时精度有限，因此适用于对精度要求较低的边缘 RDMA 设备。

表 I 定时器管理方案比较

来源	实现平台	管理方法	计时精度
jingzhao[10]	硬件	单周期检查	微秒级
AccelTCP[15]	软件	时间轮	微秒级
CONNECTX-6[19]	硬件	未知	低于 20 纳秒
所提方案	硬件	优先级队列	12 纳秒

有序列表法通过根据任务的时间值对任务进行排序来管理定时器，并且只需检查当前值是否已到期。它可以通过时间轮或优先级队列实现。以时间轮[17]为例，它通过循环数组实现。数组中的每个元素对应一个列表，每个列表存储到期时间相同的任务。当前时间由一个指针表示。每经过一个时间间隔，指针就移动一个单元，以检查其所指向的列表中是否存在过期任务。通常，如果当前列表中的元素尚未全部出队，指针就不会跳到下一个时间单元。因此，列表的容量会限制计时精度。这种方法常用于软件实现。优先级队列法的工作方式如下：注册时，按到期时间升序排列任务。出队时，每次只需比较第一个元素与外部参考时间的关系。此外，其计时精度与任务规模无关。

多级队列法采用分层技术实现大范围计时，其中每一级代表不同的时间粒度，并可使用单个时间轮或优先级队列来实现。然而，该方案在添加定时器时会消耗大量资源；当定时器数量较多时，系统性能将受到限制 [18]。

我们在表 I 中总结了现有的各种定时器管理方法。具体而言，Jingzhao RDMA NIC 的计时精度与遍历规模相关，并且与定时器数量成正比。在通过软件实现的时间轮管理中，计时精度通常处于微秒级，例如， [15] 中将时间间隔设置为 100 µs。相比之下，我们所提的硬件定时器队列方案实现了纳秒级计时精度，并确保在有限位宽内溢出后仍能进行准确计时。

近年来，学术界也对优先级队列的硬件实现开展了一定程度的研究

表 II 优先队列功能比较

优先级队列	应用	架构	支持操作				排序方法	
			入队	出队	移除	更新	全局	组
BSRTQ[21]	任务调度	一维移位寄存器	√	√	√	×	√	×
BMW-Tree[23]	数据包调度	树形脉动阵列	√	√	×	×	√	×
AnTiQ[24]	定时器队列	一维脉动阵列	√	√	√	×	√	×
所提方案	定时器队列	一维混合结构	√	√	√	√	√	√

重点涉及处理器任务调度、数据包调度和定时器队列管理等应用领域。

对于面向任务队列的硬件调度器， [20] 基于二叉堆架构实现了软硬件协同管理的优先级队列。然而，作者认为，丢弃和减键操作很少使用，且在硬件中实现起来较为复杂。因此，只有入队和出队操作由硬件实现，而丢弃和减键操作则由软件处理。 [21] 开发了一种基于移位寄存器的硬件优先级队列，名为 BSRTQ（基本移位寄存器任务队列），支持入队、出队和移除操作。该队列通过集中控制处理入队指令，并通过分布式控制处理其他队列控制指令。不过，它不支持硬件更新操作。

对于可编程数据包调度场景， [22] 提出了一种基于一维脉动阵列的单指令多数据（SIMD）优先级队列。除了支持推入和弹出操作外，该队列还增加了替换操作，用于将最高优先级元素替换为新插入的元素。[23] 提出了一种基于树的脉动阵列结构，该结构同样仅支持推入和弹出操作，但能够实现高吞吐量和大规模扩展。得益于脉动阵列实现方式，每项操作的时间复杂度均与队列深度无关。

针对定时器管理场景中定时器队列移除操作效率低的问题， [24] 提出了一种基于脉动阵列实现的优先级队列，支持 push、pop、drop 和 peek 四种操作。每个单元均由状态机控制。由于 push 和 pop 操作完成所需的周期数不同，队列内部可能出现数据冲突，需通过额外的控制逻辑予以解决。该方案使用单调递增计时器进行计时，因此不可避免地会出现外部计时器溢出问题——论文既未提及这一问题，也未说明解决方法。如果采用 [25] 中的方案，即通过增加位宽来解决时间戳回绕问题，那么在工作频率为 1 GHz 时，32位计数器每 4 秒就会回绕一次。改用 64位计数器后，回绕间隔延长至 584 年，在一定程度上缓解了超时溢出问题。然而，从实现角度来看，资源开销几乎翻倍。相比之下， [26] 中通过 d 叉堆实现的硬件定时器仅使用标志位标记溢出异常，却没有说明溢出发生后如何处理发生后。

硬件优先级队列主要用于数据包调度器，其本质上不同于定时器队列。尽管二者都基于优先级（时间值）对元素进行排序，但数据包一旦入队，便无法从队列外部访问。此外，不存在同一个数据包重新进入队列并修改其优先级的情况。因此，只需支持两种基本操作——入队和出队。与此同时，为应对网络流量峰值，数据包调度将高吞吐量和低延迟作为核心要求。以 BMW [23]和 BBQ [27]等硬件优先级队列为例，这些结构均针对工作守恒调度 [28]设计：只要链路空闲，数据包就会立即转发。在定时器队列中，只有超时周期到期后才会执行出队操作——这与非工作守恒调度[28]相一致。在等待高优先级任务期间，链路保持空闲，更加重视出队时序的准确性。此外，定时器队列需要访问并修改队列中元素的时间值，以实现动态更新，因此移除成为一项必要的基本操作。表 II 对上述具有代表性的硬件优先级队列实现进行了比较，其中没有一种支持更新操作。将这些技术引入优先级队列的定时器管理中，既无法实现对现有元素的优先级修改，也无法解决由固定位宽导致的溢出问题。

III. 操作概述

本节首先分析优先队列基本操作的执行流程，然后说明更新操作和分组排序的原理。

A. 优先队列的基本操作

在优先队列中，元素需要根据其优先级（即排名）进行排序。实际应用中，每个元素还关联一个标识符，用于识别该元素。在为数据包调度器设计优先队列时，这个标识符也称为元数据。常见的优先队列支持以下三种基本操作：

- 入队：向队列中添加一个新元素；
- 出队：从队列中取出优先级最高的元素；
- 移除：从队列中删除特定的目标元素；

从搜索和移动的角度来看，入队操作仅涉及搜索插入

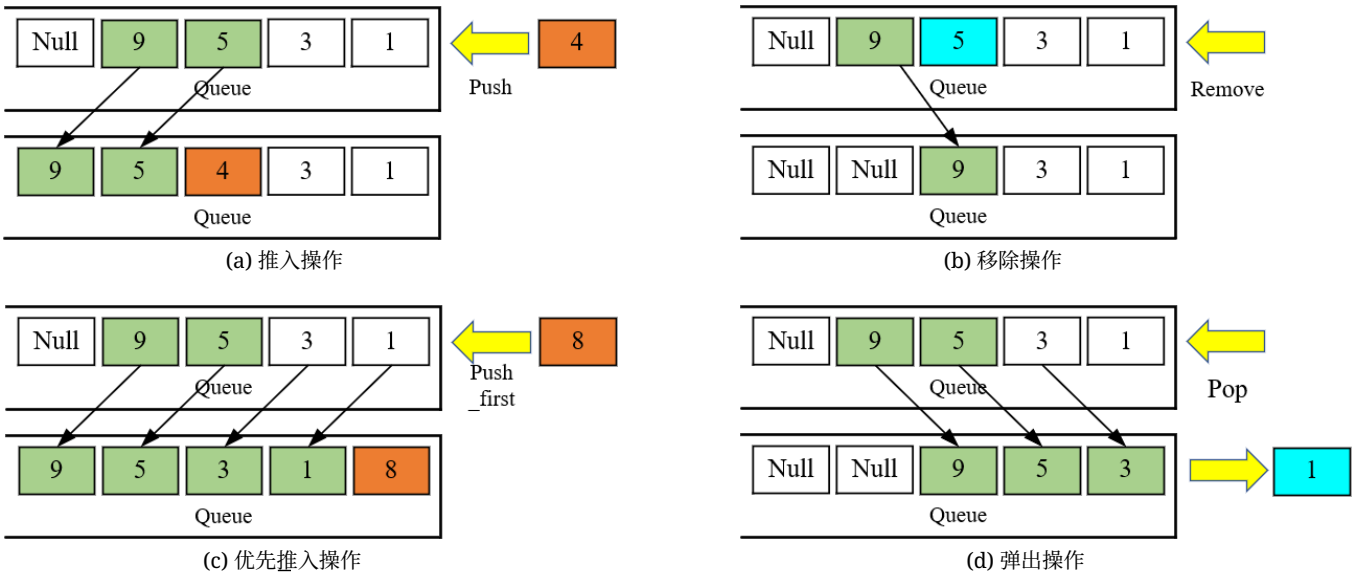


图1. 基本操作的分解

位置，而移除操作仅涉及搜索标识符。图 1 展示了各操作的分解过程。图 1(a) 描绘了推入操作：入队元素通过比较排名来搜索其插入位置。确定合适的位置后，其他元素会发生移位（即先移动，再设置）。由于设置和移动在同一位置不会冲突，因此可以同时执行。如图 1(b) 所示，移除操作首先搜索目标元素的标识符（如蓝色方块所示），然后右移（以绿色方块标示）以覆盖目标元素，从而完成删除过程——这称为搜索后移动。

从转换的角度来看，弹出操作可以视为移除操作的一种特殊情况。由于要删除的元素位于队列前端，因此无需进行搜索——只需直接移位覆盖即可，如图 1(d) 所示，其中蓝色块被弹出。类比可知，推入操作也必然存在一种特殊情况：无需搜索，直接将元素插入队列前端。我们将此操作称为“前端插入”，如图 1(c) 所示，其中红色块被放置在前端。 –

上述四种操作统称为基本操作，全部分为两个阶段：先搜索，然后移位和定位。移位过程始终存在，而是否进行搜索和定位则取决于具体的操作类型。虽然上述说明以一维结构为例，但同样适用于基于树的结构——因为搜索时只需选择树中的一个分支。在现有的基于树的结构 [23],中，元素插入所对应的分支由其路径上的计数器值确定。

B. 更新操作

更新操作既需要定位目标 ID，也需要根据优先级对其进行排序。从元素转换的角度来看，我们说明更新操作与基本操作之间的关系。为保持一般性，我们假设一个由多个有限深度子队列组成的无限深度队列 q_i；在单个有限深度子队列 q_i 中，两种搜索方法对应四种情形：

- 同时定位目标 ID 和优先级插入位置；
- 仅定位目标 ID，而未定位优先级插入位置（即目标元素的优先级更新为较低值）；
- 仅定位优先级插入位置，而未定位目标 ID（即目标元素的优先级更新为较高值）；
- 既未定位目标 ID，也未定位优先级插入位置；

在深度有限的队列 q_i 中，定位目标 ID 对应于移除操作。由于队列连续性，必须弹出相邻队列 q_{i+1} 的队首元素，即对 q_{i+1} 执行出队操作。如果未找到目标 ID，则应将移除操作传播至 q_{i+1}。定位优先级插入位置对应于通过左移和定位执行推入操作。当 q_i 中的元素左移时，由于队列连续性以及元素按优先级从高到低排列这一全局性质，q_i 的最后一个元素会移至 q_{i+1} 的前端。因此，当在 q_i 中执行入队操作有效时，会将前端插入操作传播至 q_{i+1}；如果在 q_i 中未找到优先级插入位置，则应将入队操作传播至 q_{i+1}。同时找到 ID 和插入位置，即同时发生左移和右移时，由于移位范围不同，只会发生部分元素位移，不会传播进一步的操作。因此，同时搜索目标 ID 和优先级时，上述四种情况将分别对应以下操作：

- 移除和入队操作均有效，且不会

表 III 操作传播表

	入队	移除	出队	前端插入
按 ID 查找 按秩查找	×	×	×	×
按 ID 查找 未找到秩	√	×	√	×
未按 ID 查找 按秩查找	×	√	×	√
未按 ID 查找 未找到秩	√	√	×	×

传播至后续队列；

- 仅移除操作有效，并将出队和入队操作传递到后续队列；
- 仅入队操作有效，并将前端插入和移除操作传递到后续队列；
- 将入队和移除操作传递到后续队列；

总之，更新操作可以分解为基本入队操作和移除操作的组合，并通过传递这两种基本操作来完成。传递情况如表 III 所示。这里，“按 ID 查找”表示在当前队列中查找该 ID，而“按秩查找”表示在当前队列中找到适当的插入位置。因此，更新操作可以通过实现这些基本操作的组合与传递来完成。

C. 分组排序

在优先级队列运行期间，如果具有更高优先级的元素被持续入队和出队，队列中的低优先级元素将永远无法出队——这种现象称为饥饿。这是因为搜索过程遵循这样的原则：在任意位置，优先级都按从高到低的顺序排列。正是这一特性，使得定时器溢出后无法确保计时准确性。如果将队列划分为多个组，在组内而非全局范围内强制采用优先级降序排列，同时组之间的排序保持灵活，则即使存在高优先级元素，也可以对低优先级元素进行出队，从而实际上为前者引入延迟。这种机制称为分组排序。

分组排序定义如下：队列中的元素根据一个边界优先级划分为两个组。每个组内的元素均按照优先级降序排列，而组之间的顺序则可以变化。在入队时，根据组间顺序搜索并插入元素。使用序理论中的符号，支持分组排序的优先级队列规则定义如下：

- 队列 Q 的优先级范围为 $R_Q = [P_{\text{最小值}}\ P_{\text{最大值}}]$，其中 $P_{\text{最大值}}$ 表示最高优先级，$P_{\text{最小值}}$ 表示最低优先级。定义一个边界优先级 P_l，满足 $P_l \in (P_{\text{最小值}}\ P_{\text{最大值}})$。子组 Q_1 的优先级区间为 $[P_l, P_{\text{最大值}}]$，，子组 Q_2 的优先级区间为 $[P_{\text{最小值}}\ P_l)$。每个子组内的元素按优先级降序排列（从最高到最低）。 Q_1 和 Q_2之间存在两种可能的顺序：$\{Q_1, Q_2\}$ 或 $\{Q_2, Q_1\}$；
- 当 head(Q) 的优先级 P_h 高于 P_l（即 $P_h \succ P_l$）时，入队排序规则遵循 $\{Q_2, Q_1\}$；
- 当 head(Q) 的优先级 P_h 不高于 P_l（即 $P_h \preceq P_l$）时，入队排序规则遵循 $\{Q_1, Q_2\}$；

表 IV 入队排序规则

队首与边界	排序规则	P_e 与 P_l	位置
$P_h \succ P_l$	$\{Q2, Q1\}$	$P_e \succ P_l$	图2(a)
		$P_e \preceq P_l$	图2(b)
$P_h \preceq P_l$	$\{Q1, Q2\}$	$P_e \succ P_l$	图 2(c)
		$P_e \preceq P_l$	图2(d)

根据上述规则，子组之间的关系由头部优先级 P_h 与边界优先级 P_l 的比较决定，而每个子组内的元素仍按优先级降序排列。对于入队元素 e，确定其插入位置分为两个阶段：首先，确定其所属子组；其次，将其优先级与该子组内其他元素进行比较，以找到插入位置。根据上述定义，如果入队元素 e 的优先级为 P_e，则入队排序规则总结于表 IV。

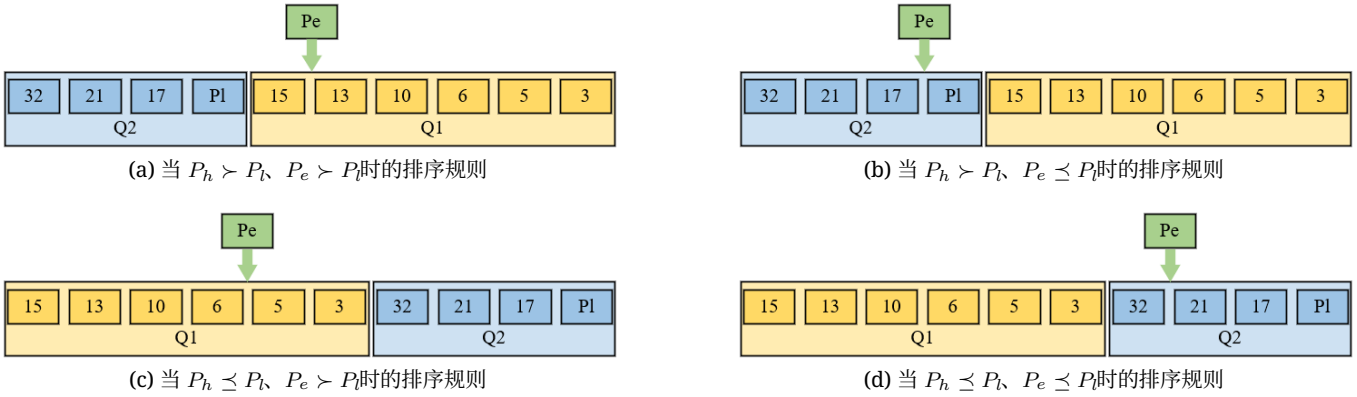
图2(a)和2(b)展示了排序规则，即整个队列按照优先级降序排列元素。高优先级元素依次出队，直到$P_h \preceq P_l$，此时优先级队列中已没有剩余的 Q1 元素，因此先将低优先级组 Q2 出队。当入队的元素 e 属于 Q1（即 $P_e \succ P_l$）时，根据排序规则，它会跳过与 Q2 中所有元素的比较，直到找到第一个空槽或 Q1 元素。只有在 Q2 中所有元素均已出队后，才允许 Q1 元素出队。因此，分组排序通过改变插入位置来调整元素顺序。发生定时器溢出（即优先级提升）时，图 2(c)中的排序规则允许溢出值延迟出队，从而解决溢出问题。

IV. 架构设计

本节首先设计优先级队列的整体架构，以及脉动单元及其时序。最后，我们设计一种比较器结构来实现分组排序。

A. 混合架构

硬件优先级队列的常见实现方案包括一维结构和基于树的结构。基于树的结构具有较强的可扩展性，但由于分支众多，不利于操作传播。在一维结构中，一维脉动阵列的运行频率高于移位寄存器；然而，与脉动阵列相比，移位寄存器的资源开销更低[30]。因此，我们采用一维脉动阵列与移位寄存器相结合的混合结构作为最终实现。为减少合并和传播时重复的元素移动



(a) 当 $P_h \succ P_l$、$P_e \succ P_l$时的排序规则

(b) 当 $P_h \succ P_l$、$P_e \preceq P_l$时的排序规则

(c) 当 $P_h \preceq P_l$、$P_e \succ P_l$时的排序规则

(d) 当 $P_h \preceq P_l$、$P_e \preceq P_l$时的排序规则

图2. 分组排序规则确定的插入位置

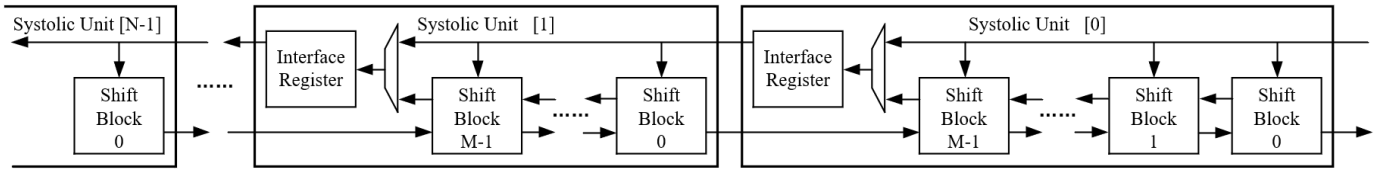


图 3. 混合架构

基本操作，我们扩大了单个脉动单元内的比较范围。此外，我们优化了在脉动阵列中维持优先级相同的元素先进先出（FIFO）顺序的实现方法。

如图 3 所示，混合结构（仅展示推入操作下的简化结构图）主要由两个模块组成：脉动单元和移位块。将 N 个脉动单元级联即可构成脉动阵列，每对脉动单元之间设有接口寄存器模型，用于将操作传递至下一级。每个脉动单元由 M 个级联的移位块组成，使队列的最大容量达到 $N * M$ 个元素。移位块中的每个元素包含两个字段：标识符和数据，其含义如下：

- 标识符：此字段对应每个元素的标识符，在队列中唯一。该字段初始化为全零；零不代表任何元素，表示当前位置为空；
- 数据：存储每个元素对应的时序值，并根据这些值的比较结果进行排序。该字段初始化为全一；

脉动单元的内部结构如图 4 所示，其中虚线表示控制逻辑，实线

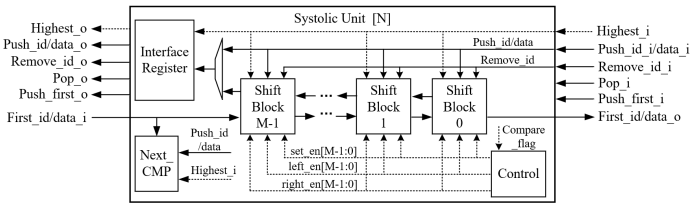


图 4. 脉动单元

表示元素传输。队列中的元素按 DATA 值递增（即优先级递减）的顺序从右向左排列。更新操作和入队操作有所不同：入队操作会插入一个新元素，而更新操作会修改一个已有元素；但在实现层，两者的处理方式相同：均通过 $Push\ id\ i$ 和 $Push\ data\ i$ 端口作为推送操作执行。

元素进入脉动单元时，其优先级会与总共 M+1 个元素进行比较：当前单元中的 M 个驻留元素，以及下一级单元的队首元素，分别由 $First\ id\ i$和 $First\ data\ i$标识）。如图 4 所示，这一比较由 Next_CMP 模型和各移位块中的比较器执行。此设计可预先解决一种潜在的低效情况：若两个有效操作同时传播，当新元素插入当前单元的尾部位置时，可能导致下一级的队首元素发生冗余移动。通过直接与下一级队首元素进行比较，该机制显著降低了周期开销。

所有这些比较的结果汇总为图 4 中名为 $compare\ flag$）的信号。鉴于 ID 搜索结果为独热码，而数据比较会产生由连续的 0 和 1 组成的二进制序列，因此可以使用布尔逻辑运算推导出每个移位块的控制信号。此外，每个脉动单元的 $Push\ first\ i$信号实现了优先推入操作，从而保持优先级相同的元素之间的先进先出顺序。 [30] 通过为每个元素添加专用标志位，解决了保持先进先出顺序这一问题。

图 5 给出了时序图，展示操作在多个脉动单元间的传播过程，其中每项操作都会产生 3 个周期的开销。当两项操作同时有效时

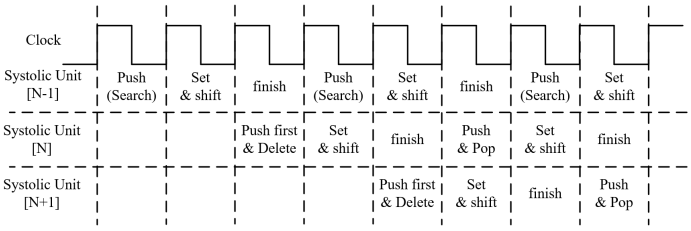


图5. 传播操作的时序图

通过布尔逻辑解析移位和置位信号的组合逻辑路径会相对较长。为缓解这一问题，搜索阶段与移位与置位阶段彼此分离，并分布在两个独立的周期中进行流水线处理。在完成阶段，寄存器完成定位；此时，下一级头元素趋于稳定，可以进行比较和搜索。同时，操作会传播至后续的脉动单元。

原则上，弹出和前端插入操作不需要比较过程。然而，由于这两类操作需要同时传播，因此仍将其设计为在3个周期内完成。这样设计是为了避免两种操作必须并发传播时，因完成所需周期数不同而产生潜在冒险。

B. 分组排序的实现

在基于优先级队列的定时器队列管理中，计算到期时间的一种方法是将固定的超时时间加到外部参考计时器上。具体而言，数据的计算如公式（1）所示，其中 R_t 表示位宽为 W_r 的外部参考计时器，其位宽与数据的位宽一致。 R_t 以设定的计时精度递增，计时范围为 $[0, 2^{W_r}-1]$，对应的优先级从 P_{max} （最高）到 P_{min} （最低）。TO 表示为每个任务设定的超时时间，位宽为 W_o；因此，可配置的超时时间范围为 (0， $2^{W_o}-1$]，其中 $W_r > W_o$。

$$DATA = R_t + TO \tag{1}$$

根据公式 (1) 执行加法后，数据的位宽可能扩展至 W_r+1，这意味着发生了溢出。此外，随着时间推移，R_t 也会溢出。令P_e 表示待入队元素 e 的数据值。按全局优先级降序排序时，如果数据溢出而 R_t 没有溢出，则 P_e 必须小于 R_t。在这种情况下，溢出元素会立即出队，违反其预期的超时周期 TO。此处将溢出值记录为 $P'_e = TO - (2^{W_r} - R'_t)$。然而，溢出元素应仅在经过 TO 个时间单位后才到期。如图 2(c) 所示，采用分组排序时，经过 $2^{W_r} - R'_t$ 个时间单位后，图 2(c) 中 Q2 组的所有元素都会出队，队列将恢复为图 2(a) 所示的状态。随后，再经过 $TO - (2^{W_r} - R'_t)$ 个时间单位后， P_e 出队。如果不更新 P_e ，从入队到出队的总时长恰好为 TO 个时间单位。因此，当

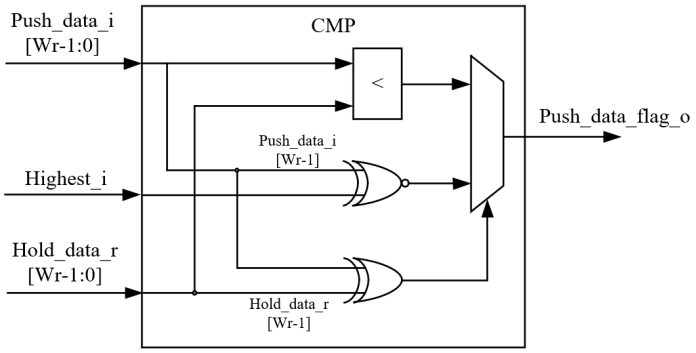


图 6. 溢出控制比较器

数据溢出时，图 2(c) 中的排序方法能够实现正确的出队时序。无论数据和 R_t是否溢出，采用图 2(a)、2(b) 或 2(d) 所示的方法都能实现恰当的出队时序。这表明，分组排序能够解决由溢出导致的时序错误。

根据分组排序规则，在定时器队列中实现分组排序时，需要设置一个边界值 P_l 。根据图 2(c) 所示的原理， P_e 无需与 Q2 中的任何元素进行比较；只有进入 Q1 的范围后，才会进行比较和插入。因此，必须调整插入比较逻辑。在本设计中，为了降低资源开销并利用二进制系统的特性，我们选择将优先级范围均匀划分，即 $P_l = 2^{W_r-1}$。因此，Q1 的范围为$[0, 2^{W_r-1}-1]$，Q2 的范围为 $[2^{W_r-1}, 2^{W_r}-1]$。此时，要求 $W_r > W_o + 1$。在实现中，只需比较新到元素的数据与队列中元素的数据的最高有效位，即可区分组 Q1 和组 Q2。若最高有效位相同，则在组内根据数据值进行完整优先级比较。若最高有效位不同，则跳过当前组中的比较。这种机制能够高效满足分组排序的要求。与大幅增加位宽相比，该方案有效降低了资源开销。图 6 展示了溢出控制比较器的实现方案，该比较器用于比较各移位块中的时序值，同时也对应于图 4 中 Next CMP 的实现。此处， $Highest\ i$ 表示队首元素数据的最高有效位，与图 4 中的 $Highest\ i$ 信号相对应。当 $HIGH$时，该信号表示队首元素的优先级大于或等于 P_l。同时，$Hold\ d\bar{a}ta\ r$ 是用于存储元素的移位寄存器信号。$Push\ data\ flag\ o$是比较结果，用于布尔逻辑运算。

– –
– – –

V. 用例

为展示更新和分组排序的有效性，我们将硬件定时器队列应用于管理流表条目的超时。通过配置两种计时精度和多个超时值进行了仿真。外部时钟参考定时器持续递增：数据包到达时，计算对应流的新到期时间并将其入队；

表 V 不同数据宽度 W_r

定时器宽度 W_r	p /周期	TO /周期	Pop 数量	Max 队列占用	Mpps
9, 10, 11, 12	6	127	14821	107	166.41
10, 11, 12	6	191	12635	127	166.26
10, 11, 12	6	255	11733	147	166.12
10, 11, 12	6	383	10649	185	165.81
11, 12	6	511	8938	215	165.52
12	6	1023	6708	321	164.27

当队首元素的时间早于外部参考计时器时，执行 apop 操作。所用数据包取自 [31] 中 UNIV1 数据集的一个子集。将数据集中 TCP 五元组相同的流计为激励。所有实验均测试了 2047 条流和总计 119,870 个数据包，队列深度设为 2048，时钟周期设为 2 ns——由此得到的理论队列吞吐量为 166.6 百万包每秒。由于所有实验中仿真队列的最大利用率均未超过 50%，因此未在仿真中加入移除操作；仅包含推送和弹出操作。

单个操作需要 3 个周期才能完成，且每次弹出操作只能使一个任务出队。队列中可能存在许多到期时间相同的任务；在这种情况下，采用交替推入和弹出操作方法，以减少到期时间相同的任务出队等待时间，以及外部更新或入队操作的等待时间。为确保在单个计时精度间隔内至少完成一个交替推入和弹出周期，将计时精度 p 设置为 6，这意味着外部参考计时器每 6 个时钟周期递增 1 次（12 ns）。

表 V 展示了不同定时器宽度 W_r 和超时值 TO 下的实验结果。可以观察到，对于给定的 p 和 TO，在所有测试的定时器宽度下，仿真结果均相同。这表明，在 p 和 TO 设置相同的条件下，所有仿真结果均与定时器宽度 W_r无关。其原因在于，流超时更新的数量与数据包到达频率相关：每个到达的数据包都会触发对相应流到期时间的更新。此更新过程无需软件参与队列控制，完全由传入的数据包驱动。这消除了软件算法进行定时器检查和流到达统计的必要性，有助于降低控制平面上的软件负载。此外，分组排序确保在到期时间溢出后仍能保持正确的顺序和计时。因此，在数据包到达频率相同的情况下，更新数量和到期时间的增量保持一致，从而产生相同的仿真结果。当 TO 设置为 127 时，操作吞吐量达到 166.41 百万包每秒，接近理论吞吐量，表明系统以接近工作保持调度的方式运行。随着超时值增加，弹出操作的数量减少，操作吞吐量下降，进一步证实系统处于非工作保持调度模式下运行。

在相同场景下，我们测试了一个不支持

表 VI 定时器精度 p=12

定时器宽度 W_r	p /周期	TO /周期	Pop 数量	Max 队列占用	Mpps
12	12	127	11742	145	166.12
12	12	191	10674	184	165.81
12	12	255	8949	214	164.52
12	12	383	7600	271	164.92
12	12	511	6716	321	164.28
12	12	1023	4406	496	161.66

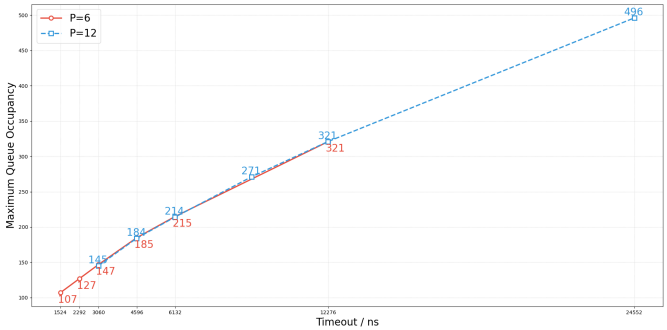


图7. 不同精度下的最大值队列占用

组排序的定时器队列。对于表 V 中最大队列占用量为 107（即计时精度$p = 6$、超时 $= 127$）的实验，我们通过增加位宽来修改所提方案，以满足需求。在总时长为 809,383 ns 的仿真中，两种方法均得到相同的仿真结果。然而，采用位宽扩展方法的定时器队列需要 W_r 17 位的定时器宽度，而支持组排序的定时器队列只需要 W_r 9 位。如果延长仿真时间，到期时间还会进一步增加。非组排序队列的定时器宽度最终将超过 17 位，而启用组排序的队列则能以固定的 9 位宽度保持正确排序，不会随仿真时间增加。因此，组排序能够有效确保溢出后的出队计时准确性，同时显著降低资源开销。

此外，计时精度 p 设为 12。仿真结果见表 VI。对比表 V 和表 VI 可得出以下结论：随着计时精度提高，弹出操作的数量减少，而最大队列占用量增加——这一现象与仅增加超时时观察到的现象一致。原因在于，提高计时精度或延长超时，都会使定时器在队列中的停留时间延长。这反过来会增加更新的概率，并减少弹出操作的数量。对比表 V 和表 VI 还可观察到，当 $p \times TO$ 值接近时，仿真结果也接近。以最大队列占用量为例（如图7所示），计时精度为 $p = 6$ 时的深度曲线与 $p = 12$ 时的深度曲线高度重合，且趋势一致。这是因为数据包激励相同，并且当 $p \times TO$ 值接近时，流在队列中的总停留时间

相近，因此仿真结果也相近。因此，要保持相同的有效超时，仅修改计时精度或仅增加超时都能达到相同目的。由此，这种定时器队列设计能够满足高精度计时、规模扩展和动态硬件更新的要求。

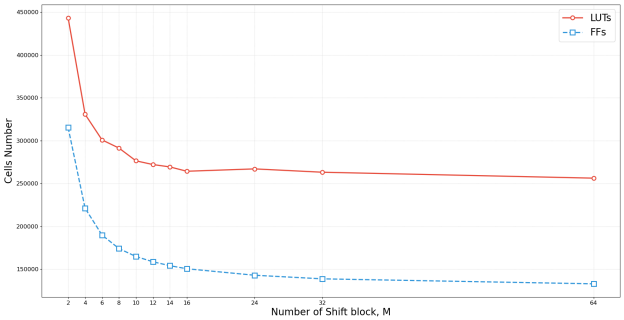
VI. 评估

A. 原型实现

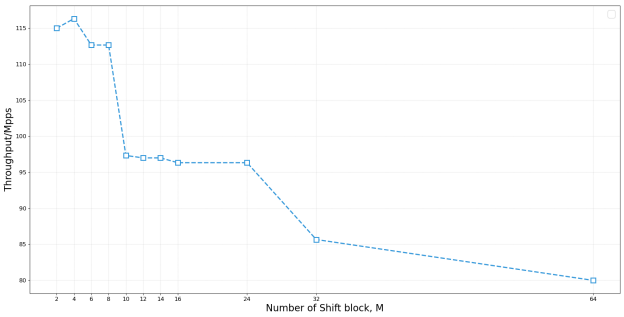
我们所提出的硬件优先级队列采用 Verilog 编写，并使用 SystemVerilog 进行验证。为评估其性能和资源开销，我们分别基于现场可编程门阵列和 28纳米工艺进行了评估。现场可编程门阵列的综合与实现使用 Xilinx Vivado 2024.2 完成，平台为 Xilinx VCU118开发板，该开发板具有 1182K 个查找表和 2364K 个触发器。队列的可重配置参数包括：标识符宽度、数据宽度、脉动单元的数量（N），以及每个脉动单元中的移位块数量（M），并要求 $M \geq 2$。通过设置不同的 N 和 M 值，我们研究了在相同队列深度下时钟频率和资源开销的变化规律。此外，为了证明我们的设计能够有效解决固定位宽硬件定时器的溢出问题，我们将其资源开销与一种不支持溢出的结构在相同仿真场景下进行实现时的资源开销进行了比较。最后，我们还综合了开源定时器队列 AnTiQ [24] 以供比较。

B. 成本与性能

队列深度固定为 4K，支持 16 位定时器（即 64K 个优先级）。配置不同的 N 和 M 值，以研究在相同队列容量下各种结构的性能和资源开销。由于队列深度为 4K（即队列最多可容纳 4096 个元素），标识符宽度设为 12 位。不同结构的资源开销和吞吐量如图 8 所示。在队列深度和优先级数量相同的情况下，随着 M 增大，查找表和触发器数量的下降趋势逐渐放缓，甚至趋于稳定。当 $M = 2, N = 2024$ 相应地，且面积达到最大值。该结构的组合逻辑延迟最短，因此理论上吞吐量应达到最大值。然而，由于现场可编程门阵列中存在布线延迟（占总延迟的大部分），最大吞吐量 116 百万包每秒是在 $M = 4$ 时实现的。随着 M 增大、N 减小（移位块总数保持不变），脉动单元数量减少，从而使查找表和触发器总数迅速下降。吞吐量也迅速下降，这是因为在单个脉动单元内进行广播比较时，总线负载随 M 增大，导致组合逻辑延迟增加。广播负载的增加也会加剧现场可编程门阵列布局后的布线延迟。总线负载和布线延迟增加的共同作用降低了现场可编程门阵列的工作频率，进而降低吞吐量。



(a) 查找表和触发器消耗

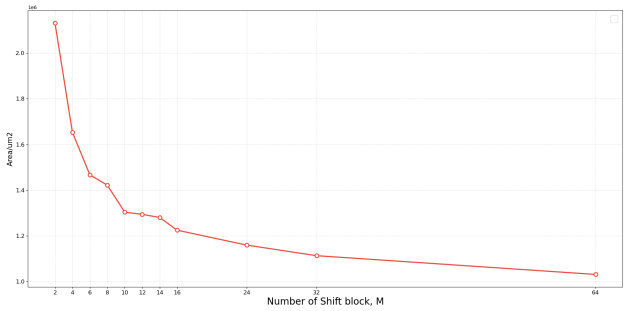


(b) 最高频率

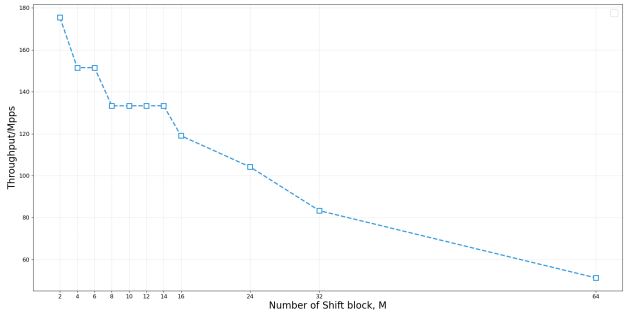
图 8. 现场可编程门阵列实现结果

作为一种支持容量为4096 的基于寄存器的优先级队列，PIFO [28] 仅需 1 个周期即可完成一次操作，但吞吐量只有 40 百万包每秒。相比之下，我们所提架构每次操作需要 3 个周期，但吞吐量提高了 2.9 倍。这一优势主要归功于一维脉动阵列，它降低了总线负载，从而缩短了关键路径延迟。另一方面，R-BMW [23], 同样支持 4K 的容量，其吞吐量达到 192 百万包每秒（为我们设计的 1.6× ），这主要是因为它仅需 2 个周期即可完成一次操作。我们设计的操作周期较长，是因为需要处理更新操作，而更新操作可能涉及两个同时执行的基本操作。为应对这一复杂性并降低组合逻辑延迟，我们特意将比较阶段和定位阶段划分为两个独立的周期，因此一次操作需要 3 个周期。由此，我们的吞吐量达到 R-BMW 的 60%。

基于上述参数的架构采用28纳米工艺进行了综合。图9展示了所得的面积和吞吐量。队列深度保持不变时，随着M增大、N减小，总面积呈下降趋势，但降幅逐渐减小。相反，增大M会增加总线负载，进而延长关键路径延迟并降低吞吐量。在M=2时，最大吞吐量达到175百万包每秒，工作频率为526兆赫，关键路径延迟为1.82 ns。这满足了分组仿真提出的 2 ns 周期时间要求，对应的计时精度为 12 ns。面积-延迟积（ADP）分析表明，对于4K队列深度，M=14时的ADP最小，吞吐量为133百万包每秒。次优ADP出现在M=6时，相应吞吐量为151百万包每秒。作为比较，支持相同队列

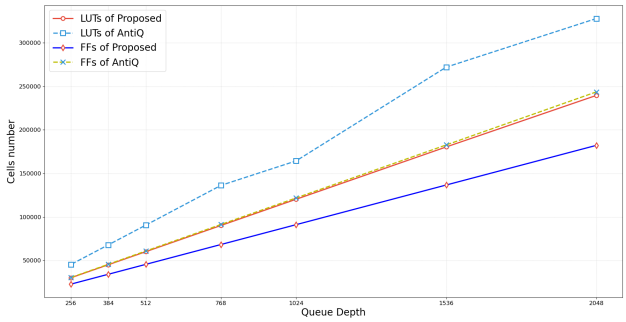


(a) 综合面积

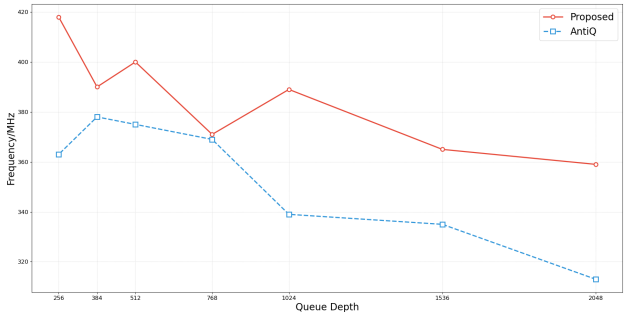


(b) 最高频率

图9. 28纳米综合结果



(a) 查找表和触发器用量



(b) 最高频率

图 10. 在现场可编程门阵列上与 AnTiQ 的比较

深度的PIFO综合后，关键路径延迟为16.93ns ，面积约为1,125,966 μm^2。虽然其面积与我们在M=24 和N=171时的设计相当，但我们提出的架构将关键路径延迟降至仅2.7 ns，使延迟降低了84%。

C. 比较

开源定时器队列 AnTiQ [24]作为我们的主要基准。为进行公平比较，我们将标识符和数据宽度均设为 16 位，以支持 64K 个不同的优先级。为突出所提方案的资源效率，我们设定 M=2，这对应于脉动单元面积最大的配置。随后，我们针对从 256 到 2048 的七种队列深度，将所提方案与 AnTiQ 进行比较分析，评估性能和开销。实现结果如图 10 所示。对于给定的队列深度，我们的架构最高工作频率达到 418 兆赫，高于 AnTiQ 的 378 兆赫。在所有测试深度下，这一频率优势均保持一致。在队列容量为 256 时，性能差距最为显著：我们的设计运行速度比 AnTiQ 快 15%，同时查找表数量减少 33%，触发器数量减少 25%，详见表VII。值得注意的是，即使在 M=2（即脉动单元面积最大的配置）下，我们的设计在评估的七种队列深度中，查找表使用量平均仍减少 31%，触发器使用量平均减少 25%。表VII 还列出了队列容量为 768 时的资源差异，此时两种设计的工作频率最为接近。这一资源

表VII 与 AnTiQ 的成本与性能比较

	N	M	深度	频率	查找表(%)	触发器(%)
AnTiQ	/	/	256	363	3.83%	1.28%
所提方案	128	2	256	418	2.54%	0.96%
AnTiQ	/	/	768	369	11.52%	3.86%
所提方案	384	2	768	365	7.63%	2.89%

效率源于我们的架构选择：AnTiQ 在每个脉动单元内采用有限状态机进行控制，而我们的设计则使用单一控制逻辑，通过布尔运算实现，并由单元内的多个移位寄存器共享。在操作开销方面，AnTiQ 执行一次移除需要 3 个周期，入队需要 2 个周期，因此一次更新操作需要 5 个周期，并可能引发操作冲突。相比之下，我们的设计以统一的 3 周期延迟执行所有操作，包括更新。在相同条件（28纳米工艺、相同队列深度和元素位宽）下进行综合后，AnTiQ 的关键路径延迟稳定为 3.06 ns。我们的架构在 M=2 设置下保持最低 1.82 ns的关键路径延迟，降低约 40%，且面积减少 13%。此外，AnTiQ 不支持溢出处理。因此，从功能、性能和资源开销的角度来看，我们的设计具有明显优势。

VII. 结论

为实现支持动态更新、缓解位宽溢出导致的时序异常的高精度大规模定时器管理，我们开发了一种优先队列架构。本研究提出了两项关键技术：更新操作和分组排序机制。更新操作通过组合并传递基本操作来实现，从而修改现有元素的优先级。分组排序机制通过改变搜索顺序，确保溢出后过期任务能够按正确时序出队。我们的设计采用混合架构，将一维脉动阵列与移位寄存器相结合，已成功应用于流表超时管理，实现硬件动态更新，并确保溢出后仍能保持计时准确性。

更新与分组排序能力的结合，使这种优先队列架构非常适用于其他动态场景，例如处理器硬件调度器、动态数据包调度算法以及基于硬件的防饥饿机制。在未来的工作中，我们计划引入RAM宏块，在不牺牲性能的前提下降低资源开销。此外，我们还计划将这一设计集成到有状态处理网卡中，以评估其在真实高速网络系统中的性能。

参考文献

[1] A. Greenberg、G. Hjalmtysson、D. A. Maltz、A. Myers、J. Rexford、G. Xie、H. Yan、J. Zhan和H. Zhang，“一种用于网络控制与管理的全新4D方法”，*ACM SIGCOMM*《计算机通信评论》，第35卷，第5期，第41–54页，2005年。[2] M. Ku´zniar、P. Pere˘s´ıni和D. Kosti´c，“SDN流表须知”，载于被动与主动网络测量国际会议，第347–359页，Springer，2015年。[3] OpenFlow交换机规范（1.5.1版），Open Netw. Found. Std.，美国加利福尼亚州门洛帕克，2015年3月。[4] “Open vSwitch的设计与实现。”网络系统设计与实现研讨会（2015年）。[5] Y. Shen、C. Wu、Q. Cheng和D. Kong，“AFTM：一种适用于OpenFlow交换机的自适应流表管理方案”，2020年IEEE第22届高性能计算与通信国际会议；IEEE第18届智慧城市国际会议；IEEE第6届数据科学与系统国际会议（HPCC/SmartCity/DSS），斐济亚努卡岛库武，2020年，第917–922页，doi: 10.1109/HPCC-SmartCity-DSS50907.2020.00122。[6] B. Arslan、E. Kızılkaya、B. Yi˘git和F. Alag¨oz，“通过动态超时分配和主动清除最大化SDN流表效率”，*2024*年第*15*届未来网络国际会议（*NoF*），西班牙卡斯特尔德费尔斯，2024年，第10–18页，doi: 10.1109/NoF62948.2024.10741357。[7] InfiniBand行业协会。《InfiniBand架构规范》第1卷：1.4版。InfiniBand行业协会，2010年。[8] Xizheng Wang、Guo Chen、Xijin Yin、Huichen Dai、Bojie Li、Binzhang Fu和Kun Tan，Star：突破RDMA的可扩展性限制。载于第届*ICNP*论文集，[9] 2021 Wang、Zilong等。“SRNIC：可扩展RDMA网卡架构。”网络系统设计与实现研讨会（2023年）。[10] F. Yang、Z. Wang、N. Kang、Z. Ma、J. Li、G. Yuan和G. Tan，“JingZhao：专用网络时代的快速网卡原型设计框架”，arXiv，2025年。 [在线]。网址：https://doi.org/10.48550/arXiv.2410.08476。[11] AMD，“AMD嵌入式RDMA网卡LogiCORE IP产品指南（PG332）”，第4.2版，AMD，2024年11月。 [在线]。网址：https://docs.amd.com/r/en-US/pg332-ernic [12] Eddy，Wesley，编。“RFC 9293：传输控制协议（TCP）。”（2022年）。[13] V. Vasudevan 等，“面向数据中心通信的安全高效细粒度 TCP 重传”，载于*SIGCOMM* 会议论文集，西班牙巴塞罗那，2009年8月，第303–314页。[14] Andy Currid。TCP 卸载来助力：初探 TCP 卸载引擎——以及我们为何需要它们。Queue， [15] 2 3 58 65， 2004 ( ): – 5月。Moon、YoungGyoun 等。“AccelTCP：通过有状态 TCP 卸载加速网络应用程序。”网络系统设计与实现研讨会(2020)：77-92。[16] Shashidhara、Rajath 等。“FlexTOE：借助细粒度并行实现灵活 TCP 卸载。”网络系统设计与实现研讨会(2022)：87-102。[17] Varghese、G 和 A Lauck。“哈希化分层时间轮：实现定时器设施的高效数据结构”，*IEEE/ACM* 网络学报5.6 (1997)：824-834。[18] “面向网络与通信系统的高效二维索引队列定时器管理算法”，计算智能、通信系统与网络 (2011)。[19] “NVIDIA Mellanox ConnectX-6 以太网智能网卡数据手册。”Cisco，2021年5月，https://www.cisco.com/c/dam/en/us/products/collateral/servers-unified-computing/ucs-c-series-rack-servers/nvidia-mellanox-connectx-6-ethernet-smartnic-data-sheet.pdf。访问日期：2025年10月14日。[20] Kumar、NG Chetan 等。“实时与嵌入式系统中优先级队列管理的软硬件架构。”国际嵌入式系统期刊6.4 (2014)：319-334。[21] Y. Tang 和 N. W. Bergmann，“一种基于任务队列、面向基于 FPGA 的嵌入式实时系统的硬件调度器”，载于*IEEE* 计算机学报第64卷第5期，第1254-1267页，2015年5月1日，doi: 10.1109/TC.2014.2315637[22] I. Benacer、F. -R. Boyer 和 Y. Savaria，“一种快速、单指令多数据、可扩展优先级队列”，载于*IEEE* 超大规模集成电路（*VLSI*）系统学报，第26卷，第 10， 1939 1952， 2018， 10 1109 2018 2838044 期，第-页，10月。doi: . /TVLSI.。Yao、Ruyi 等。[23] “BMW 树：利用平衡多路排序树实现大规模、高吞吐量、模块化的 PIFO 实现。”*ACM SIGCOMM 2023* 会议论文集。2023年。[24] A. Nurmi、P. Lindgren、T. Szymkowiak 和 T. D. H¨am¨al¨ainen，“AnTiQ：一种支持常数时间任意元素删除的硬件加速优先级队列设计”，2023年第26届*Euromicro* 数字系统设计会议（*DSD*），阿尔巴尼亚 Golem，2023年，第462-469页，doi：10.1109/DSD60849.2023.00070。[25] Benacer、Imad、Francois-Raymond Boyer 和 Yvon Savaria。“一种面向基于流的网络的高速流量管理器架构。”2017年第15届*IEEE* 国际新型电路与系统会议（*NEWCAS*）。IEEE，2017年。[26] Septinus、K. 等。“网络系统中高要求定时器管理的可扩展硬件算法”，PARS 并行算法、计算机结构与系统软件 28.1 (2011)：58-67。[27] Atre、Nirav 等。“BBQ：一种用于硬件数据包调度的快速可扩展整数优先级队列。”网络系统设计与实现研讨会(2024)：455-475。[28] A. Sivaraman，“线速可编程数据包调度”，载于*ACM SIGCOMM* 会议论文集，2016年8月，第44–57页。[29] S. Collinson、A. Bai 和 O. Sinnen，“一种快速可扩展硬件优先级队列及多重压入优化”，2024年*IEEE* 国际并行与分布式处理研讨会（*IPDPSW*），美国加利福尼亚州旧金山，2024年，第134-140页，doi：10.1109/IPDPSW63119.2024.00038[30] Sung-Whan Moon、J. Rexford 和 K. G. Shin，“面向高速数据包交换机的可扩展硬件优先级队列架构”，载于*IEEE* 计算机学报，第 49， 11， 1215 1227， 2000 卷第期，第-页，11月。Benson、Theophilus 等。[31] “真实环境中数据中心的网络流量特征”，ACM/SIGCOMM 互联网测量会议 (2010)：267-280。



Zekun Wang 于2020年和2023年分别获得中国西安西安电子科技大学集成电路设计专业学士学位和硕士学位。目前，他正在西安中国西安电子科技大学微电子学院攻读电子与信息工程博士学位。他的研究方向是用于网络流量管理的可编程调度器和网络处理器架构设计，创新重点在于数据中心网络优化。



Binghao Yue 于2023年获得西安电子科技大学通信工程学士学位，目前正在西安电子科技大学通信工程学院攻读硕士学位。他的研究方向包括基于RDMA（远程直接内存访问）的网络加速、智能网络接口卡（片上系统网络接口卡）架构，以及边缘计算硬件优化。



Weitao Pan （IEEE会员）于2004年获得西安电子科技大学技术物理学院学士学位，并于2010年获得西安电子科技大学微电子学院博士学位。他目前是西安电子科技大学综合业务网国家重点实验室副教授。其当前研究方向包括超大规模集成电路设计方法和后硅验证。



Jiangyi Shi 于2007年在中国西安的西安电子科技大学获得博士学位。他在集成电路设计领域拥有20多年的经验，涵盖技术开发、计算机体系结构和功率放大器。他的研究方向包括电子设计自动化工具和射频设计。近期项目包括射频与混合信号高速电路设计、网络处理器设计、集成电路安全和可信技术。



Yue Hao 1958年出生于中国重庆。1991年在中国西安的西安交通大学获得博士学位。他现任西安电子科技大学微电子学院宽禁带半导体技术国家重点学科实验室教授。其研究方向为宽禁带半导体材料与器件。