计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(2): 477−504, 2026

数据中心网络 RDMA 拥塞控制技术综述

张毓涛　杨　惠　李　韬　黄曼蒂　李天云　孙志刚
（国防科技大学计算机学院　长沙　410073）
（zhang_yt@nudt.edu.cn）

Survey of RDMA Congestion Control Techniques for Data Center Networks

Zhang Yutao, Yang Hui, Li Tao, Huang Mandi, Li Tianyun, and Sun Zhigang
（*College of Computer Science and Technology*, *National University of Defense Technology*, *Changsha* 410073）

Abstract　Congestion control is one of the key technologies for realizing high-performance data center networks, and it affects important network performance indicators such as throughput, latency, and packet loss rate. Over the past 20 years, with the continuous expansion of the scale of data centers and the increasing requirements of upper-layer applications for network performance, the deployment of remote direct memory access (RDMA) technology based on lossless underlying networks has received widespread attention within the industry. However, the priority-based flow control (PFC) mechanism, while maintaining a lossless network, introduces problems such as head-of-line blocking, leading to a decline in network performance or even network paralysis. As a crucial auxiliary means for achieving a lossless network, how to design a practical RDMA congestion control mechanism has become a hot issue. By dividing the congestion control process into congestion awareness and congestion regulation, this paper comprehensively reviews the research achievements in this field: Firstly, from the perspectives of explicit feedback and latency, different representative algorithms for congestion awareness are elaborated and summarized in detail; Secondly, representative algorithms for congestion regulation are introduced in detail from the dimensions of rate and window, and their advantages and disadvantages are summarized; Some optimization work of algorithms and congestion control algorithms based on reinforcement learning methods are supplemented; Finally, the existing challenges in this field are summarized and discussed.

Key words　data center networks (DCN); remote direct memory access (RDMA); RDMA over converged Ethernet (RoCEv2); congestion control (CC); congestion awareness; congestion regulation

摘　要　拥塞控制是实现高性能数据中心网络的关键技术之一，影响吞吐量、延迟、丢包率等重要网络性能指标。过去 20 年间，随着数据中心规模不断扩大，上层应用对网络性能的要求不断提高，基于无损底层网络的远程直接内存访问（remote direct memory access, RDMA）技术在数据中心的部署受到了业内广泛关注。然而，基于优先级的流控（priority-based flow control, PFC）机制在维护无损网络的同时会引入头阻塞等问题，导致网络性能下降甚至网络瘫痪。作为实现无损网络的关键辅助手段，如何设计实用的 RDMA 拥塞控制机制成为了热点问题。通过将拥塞控制过程划分为拥塞感知与拥塞调整，全面综述了该领域的研究成果：首先从显式反馈与延迟的角度详细阐述并总结了不同的拥塞感知代表算法；其次从速率和窗口的维度对拥塞调整代表算法进行了详细介绍并对其优缺点进行了总结；而后补充了部分算法的

收稿日期：2024-07-20；**修回日期**：2025-04-21
基金项目：国家国防科技工业局重点实验室基金项目（WDZC20245250113）
This work was supported by the Key Laboratory Foundation of State Administration for Science, Technology and Industry for National Defense (WDZC20245250113).
通信作者：李韬（litao@nudt.edu.cn）

优化工作以及基于强化学习方法的拥塞控制算法;最后总结并讨论了该领域存在的挑战。

关键词 数据中心网络;远程直接内存访问;RoCEv2;拥塞控制;拥塞感知;拥塞调整

中图法分类号 TP393

DOI: 10.7544/issn1000-1239.202440623 **CSTR:** 32373.14.issn1000-1239.202440623

近20年来,数据中心在全球范围内蓬勃发展。随着数据中心从最初的计算中心阶段发展到如今的算力中心阶段,网络规模不断扩大,对网络品质的要求不断提高[1]。尽管目前传输控制协议/网际协议(transmission control protocol/internet protocol, TCP/IP)协议栈在数据中心仍占统治地位[2-3],但是该协议栈的弊端逐渐显露,业内正积极尝试将以远程直接内存访问(remote direct memory access, RDMA)技术为代表的新技术部署于数据中心[2,4]。

RDMA实现对硬件有较强的依赖,有限的网卡片上资源使得RDMA重传采取了资源占用与消耗较小的回退*n*步(go-back-*n*, GBN)重传策略[5-7],即使在丢包率极低的情况下,该策略也会导致性能的急剧下降[8],因此RDMA的部署需要数据中心提供一个底层无损网络[2,8-14],该网络不会因内部问题发生丢包,除非链路受到外界干扰或网络存在错误配置。

数据中心利用基于优先级的流量控制(priority-based flow control, PFC)[15-16]增强的聚合以太网(converged Ethernet, CE)来实现无损网络,然而逐跳流量控制方法会带来一系列问题,例如头阻塞、PFC暂停帧风暴、死锁等[2-3,8,10-14,17-22]。为了降低触发PFC的频率,需要对RDMA拥塞控制进行精细地设计与研究[8,17,21]。

当前国内外的综述文章缺少对近年来RDMA拥塞控制机制的详细分析,例如:Huang等人[23]将无损数据中心网络的拥塞控制方案划分为主动式拥塞控制和响应式拥塞控制,对早期的RDMA拥塞控制算法进行了介绍;Hu等人[5]将数据中心RDMA传输设计划分为单路径拥塞控制、多路径传输和其他解决方案,其中单路径拥塞控制方案对部分主流算法进行了简单的阐述;曾高雄等人[24]将RDMA拥塞控制作为一种发展趋势简要地引出;杜鑫乐等人[25]则将RDMA拥塞控制、流调度和负载均衡作为数据中心流量控制的组成部分,剖析了部分拥塞控制算法;Yan等人[26]将RDMA拥塞控制算法按照发送方驱动、交换机驱动和接收方驱动3个维度划分,其侧重点与本文并不相同。本文将尝试较为全面、系统地对当前的数据中心RDMA拥塞控制机制进行分析与总结。同时,考虑到一些早期的TCP拥塞控制算法启发了RDMA的相关工作,本文尝试在RDMA拥塞控制算法之后补充相关的TCP拥塞控制算法。

1 背 景

1.1 RDMA在数据中心的部署

近20年来,数据中心相继经历了计算中心阶段、信息中心阶段、云中心阶段和算力中心阶段。随着云计算、大数据、人工智能、物联网等产业的加速发展,数据中心的规模将进一步扩大。据统计,我国的数据中心市场规模在2021年为1 800亿元,在2022年达到了2 200亿元[1]。可以预见的是,数据中心规模在当前及未来一段时间内将继续保持增长趋势。

伴随着规模的不断扩大,数据中心应用程序对网络性能的要求也随之提高。例如,网络搜索、在线零售和广告推荐等软实时应用对网络延迟十分敏感[13,21,27-40];云存储、云互联和云安全等云服务业务需要网络提供高带宽的同时不能产生过高的CPU开销[4,8-9,20,36,41-45];分布式机器学习、分布式数据库和分布式计算等分布式应用要求网络同时提供高带宽和低延迟的特性[3,8-9,14,22,41-42,46-52]。尽管TCP/IP协议栈在数据中心网络仍占据统治地位,但显然已无法满足上述要求[3,8,11,13],主要原因在于以操作系统为中心的TCP/IP协议栈执行一次完整的数据收发流程需要进行数据复制、系统调用、中断处理等操作,导致了CPU开销大、延迟高,并对吞吐量等其他性能产生影响。

具有内核旁路功能的RDMA是一项高度依赖硬件实现的高性能数据传输技术,数据收发的大部分操作流程绕过了操作系统内核,转而由硬件执行。双端的CPU仅在控制通路参与,在数据通路中,网卡能够直接访问远端内存,避免了数据在用户空间与内核空间的多次复制,报文的封装与解封操作均由网卡硬件完成,无需经过内核协议栈的逐层处理。得益于零拷贝技术、操作系统内核旁路和CPU卸载技术,RDMA能够同时满足高吞吐量、低延迟以及低CPU开销的目标。图1展示了以内核为中心的传统TCP通信方式和内核旁路的RDMA通信方式。

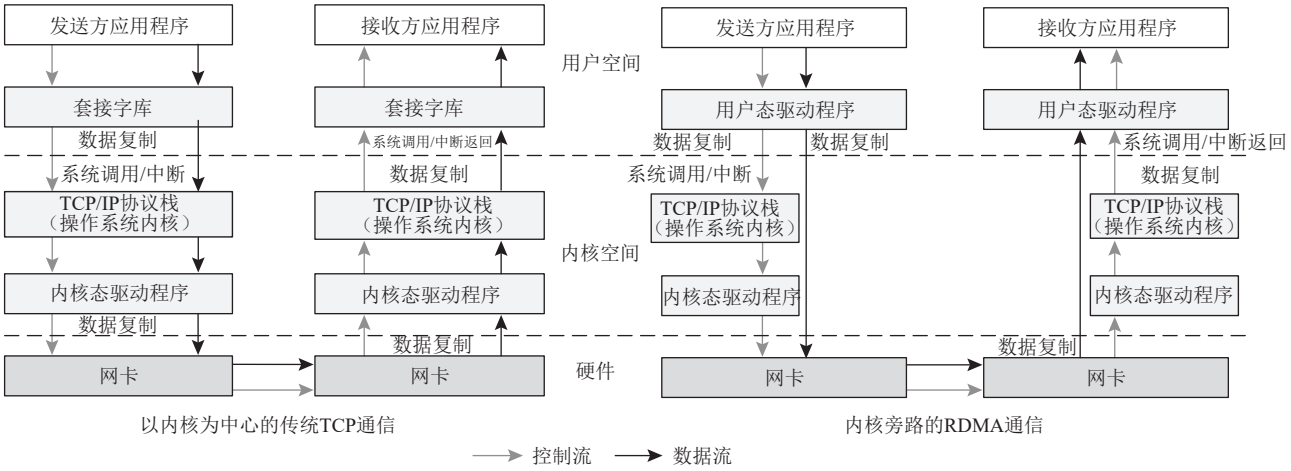


Fig. 1 Comparison of RDMA and TCP communication methods

图 1 RDMA 与 TCP 的通信方式对比

1.2 RDMA 的 3 种实现方式

RDMA 最初在无线带宽（infiniband, IB）架构上以 IB 协议的形式实现[53]，该架构要求传输层、网络层和数据链路层均使用 IB 协议，尽管能够提供较优的性能，但是使用 IB 协议进行组网通常需要定制的 IB 交换机和 IB 网卡，这与数据中心普遍使用的以太网并不兼容且成本昂贵。互联网广域远程直接内存访问协议（internet wide area RDMA protocol, iWARP）[54]同样能够实现 RDMA 数据传输。该协议在以太网数据链路层、TCP/IP 实现的网络层基础上，将传输层由上至下划分为 3 层子协议。尽管该技术可以利用以太网交换机和 iWARP 网卡实现，但是 TCP/IP 协议栈本身对延迟和吞吐量并不友好，因此相较于 IB 以及基于融合以太网的远程直接内存访问（RDMA over converged Ethernet, RoCE）而言，iWARP 的性能表现并不出众。

RoCE 协议当前有 2 个版本，如图 2 所示。RoCEv1[55]的架构与 IB 架构的不同之处在于前者利用以太网数据链路层代替了后者的链路层，因此 RoCEv1 的报文结构与 IB 报文相比添加了一个 2 层以太网报文头，这使得 RoCEv1 只能部署在 2 层以太网之上；RoCEv2[56]除了使用以太网链路层外，其网络层基于 UDP/IP 协议实现，因此 RoCEv2 的报文结构增加了 UDP、IP 和 2 层以太网报文头，可部署在 3 层以太网之上。RoCEv2 协议实现了基于以太网基础设施的组网方案，通常仅需 RoCEv2 网卡的支持。综上，RoCEv2 因其成本低廉、性能优越且兼容于以太网的特性，已成为数据中心高性能数据传输的主要选择。

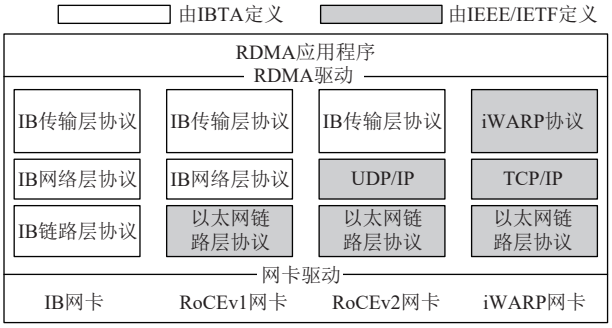


Fig. 2 Three implementations of RDMA

图 2 RDMA 的 3 种实现方式

1.3 拥塞控制机制的重要性

拥塞这一概念反映的是网络状况：在分组交换网络中，当数据包数量超过了存储转发节点的处理能力时，网络性能将大幅下降。尽管此时的带宽利用率仍保持在较高水平，但延迟将急剧增大，丢包的可能性也将急剧升高。解决这一问题的根本途径是调整数据包发送速率或发送数量以缓解网络设备的处理压力。

RoCEv2 拥塞控制算法的实现主要依赖于网卡设计，然而网卡有限的硬件资源难以支持 RoCEv2 实现高效且复杂的重传策略[3,5-7,22,52]。GBN 策略的低效性使 RoCEv2 对丢包较为敏感，因此需要一个无损网络以实现高性能数据传输。数据中心网络利用 PFC 协议来实现这一目标[3,15]。

如图 3 所示，为了向上层用户提供更好的服务质量（quality of service, QoS），交换机或网卡的入/出端口缓冲区被划分为 8 个优先级队列，同一优先级队列可能存在多条数据流[15]。PFC 规定，当交换机或网卡的入口缓冲区某队列达到缓冲区阈值时，向上游网络设备发送 PFC 暂停帧（PFC PAUSE frame），使数据源暂停发送，避免因持续发送数据包造成网络丢包。然而 PFC 策略也会带来一些严重的性能问题，

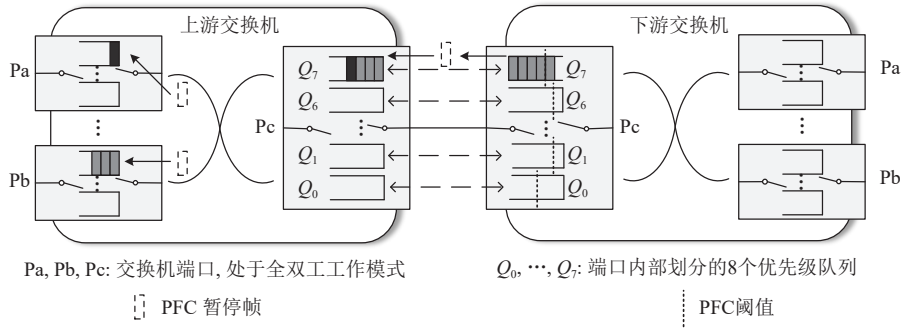


Fig. 3 Working principle and trigger mechanism of PFC

图 3 PFC 的工作原理和触发机制

例如头阻塞(head-of-line, HOL)、死锁、PFC 暂停帧风暴等[2-3,8,10-14,17-22,52]。造成上述性能问题的一个重要原因是当入端口队列中的数据包数量超过阈值时, PFC 向上游端口队列泛洪 PFC 暂停帧, 导致拥塞树扩散或错误循环依赖。

图 4 举例说明了 incast 流量模式引起的 PFC 受害流问题。1 号流和 2 号流为长流/大象流, 其源和汇分别为 SH1, RH1 和 SH2, RH2, 假设二者被映射到 P3 和 P4 的同一优先级队列中。源为 SH3~SH*n*, 汇为 RH2 的短流/老鼠流同时启动并造成 P6 拥塞。在此场景下, 有 2 个原因导致受害流问题: 1)PFC 暂停帧以队列为单位反压。端口 P4 的队列因下游端口拥塞而逐渐增大, 当其大小达到 PFC 阈值时, 向数据源(SH1 和 SH2)发送 PFC 暂停帧, 导致数据源暂停发送数据包。2)端到端拥塞控制协议无法识别拥塞流。端口 P3 收到下游的 PFC 暂停帧直到数据源停止发送, 期间 P3 的队列仍会增长, 一旦队列大小超过标记阈值, 1 号流的数据包在传输重启后同样会被标记。综上, 尽管 1 号流未对 P6 端口发生的拥塞情况有任何影响, 但是因为与 2 号流在同一队列中排队而成为受害流。

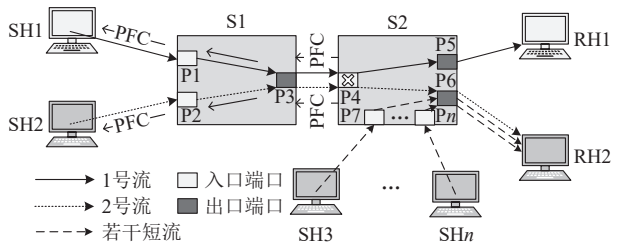


Fig. 4 Victimization flow problems caused by incast

图 4 incast 造成的受害流问题

一个有效的拥塞控制机制在一定程度上有利于避免触发 PFC 策略, 在实现无损网络的同时尽可能规避上述严重的性能问题[2-3,8-9,12,14,17,21,50,57]。近年来, 随着技术与工艺的发展, 网络设备实现了更多的功能, 例如显式拥塞通知(explicit congestion notification, ECN)、高精度时间戳测量往返时延(round-trip time, RTT)和带内网络遥测(in-band network telemetry, INT)等[58-63], 结合 P4 交换机与 FPGA 可编程网卡的可编程性, 拥塞控制机制在此基础上得到了较大的创新。

拥塞控制过程可以视为拥塞感知和拥塞调整的组合。拥塞感知为拥塞程度的度量提供了直接途径, 是端到端拥塞控制的基石[12,64], 其精确性以及感知反馈的速度将从根本上决定一个拥塞控制算法的优劣。为了兼顾吞吐量、延迟、稳定性、公平性以及收敛速度等重要指标, 拥塞调整的过程需要参考网络的拥塞程度。表 1 按相关论文发表时间展示了部分数据中心 RDMA 拥塞控制算法。

Table 1 Congestion Control Algorithms of Data Center RDMA

表 1 数据中心 RDMA 拥塞控制算法

算法	年份	拥塞感知信号	拥塞调整方式
DCQCN[8]	2015	ECN	速率
TIMELY[65]	2015	RTT	速率
Patched TIMELY[57]	2016	RTT	速率
RoGUE[20]	2018	RTT	速率+窗口
HPCC[9]	2019	INT	速率+窗口
PCN[10]	2020	ECN	速率
Swift[50]	2020	RTT	速率+窗口
RoCC[11]	2020	网内反馈	速率
FastTune[66]	2021	INT	速率+窗口
HierCC[49]	2021	网内反馈	速率
LDCP[13]	2021	ECN	窗
RCC[22]	2021	RTT	窗口
PowerTCP[51]	2022	INT	速率+窗口
RET[21]	2022	RTT	速率
PACC[52]	2022	网内反馈	速率
Bolt[67]	2023	INT	窗口
ACC[14]	2024	ECN	速率

2 拥塞感知

根据拥塞感知信号来源的不同，本节将拥塞感知划分为基于显式反馈的拥塞感知和基于延迟的拥塞感知。其中，基于显式反馈的拥塞感知根据当前主流算法的显式反馈信号，可再进一步细分为 ECN、INT 以及网内反馈。本节先对上述感知信号的采集原理进行说明，再描述相关主流算法，最后对各类型的优缺点进行总结。

2.1 基于显式反馈的感知

显式反馈信号在拥塞感知中得到了广泛的应用，按照端侧获得的信息，可以将显式反馈划分为 ECN 标记、INT 标记和网内反馈。本节将尝试对上述显式反馈原理以及基于各种反馈信息的感知算法进行详细阐释。

2.1.1 ECN 标记的显式反馈法

1993 年 Floyd 等人[68]在 TON 期刊上提出了 RED（random early detection）方法。RED 通过比较队列的平均长度与阈值区间来决定对数据包采取的动作，如图 5 所示，当队列的平均大小在最小阈值 $K_{\min}$ 以下时，RED 不会标记数据包；高于最大阈值 $K_{\max}$ 时，RED 将以 1 的概率标记数据包；若在 $K_{\min}$ 和 $K_{\max}$ 之间，则以一个关于平均队列大小的概率函数对数据包进行标记。与传统的队列管理方法相比，RED 能够实现较短的队列长度，允许网络存在一定的突发，避免流的全局同步现象。

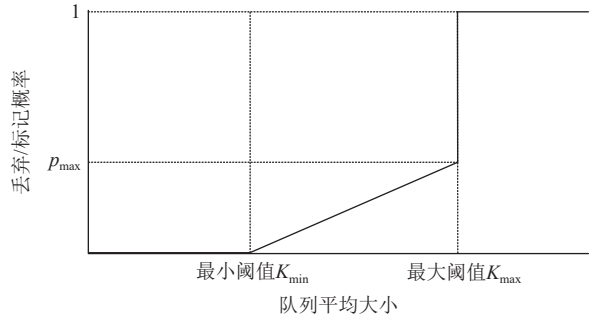


Fig. 5 RED mechanism

图 5 RED 机制

2001 年由 Ramakrishnan 等人[69]撰写的 RFC 3168 标准将 RED-ECN 方法纳入到 TCP/IP 网络中。作为一种主动队列管理（active queue management, AQM）机制，RED 为交换机 ECN 功能的实现提供了途径：结合上述 RED 方法，交换机对 IP 报文头中的 ECN 字段进行标记，将网络中的拥塞情况通告给接收方。ECN 字段及其含义如表 2 所示。

若数据包携带 ECN 使能传输（ECN-capable transport, ECT）标记，则交换机在拥塞时会将该字段标记为经历拥塞（congestion experienced, CE）；若数据包携带非 ECN 使能传输（Not-ECN-capable transport, Not-ECT）标记，则不会对数据包进行标记。ECT 和 Not-ECT 均由发送方指定。

Table 2 ECN Fields and Their Meanings as Defined by RFC 3 168

表 2 RFC 3 168 定义的 ECN 字段及其含义

ECN 字段		含义
ECT 位	CE 位	
0	0	Not-ECT：不支持 ECN 标记
0	1	ECT 字段：支持 ECN 标记
1	0	ECT 字段：支持 ECN 标记
1	1	CE：经历拥塞

微软、Nvidia Mellanox 公司和加利福尼亚大学圣芭芭拉分校的研究人员在 2015 年 SIGCOMM 会议上提出的 DCQCN 算法[8]是利用 ECN 标记进行拥塞感知的经典算法，该算法使得数据中心大规模部署 RDMA 成为了可能。它参考了 QCN[16,70]和 DCTCP[27]的相关工作。

以太网标准 QCN[16,70]是一个作用于网络第 2 层的拥塞控制算法，由 2 个子算法组成：拥塞点（congestion point, CP）算法和响应点（reaction point, RP）算法。其中 CP 是指发生拥塞的网络设备，一般是指数据中心的交换机，也可以是接收方主机或网卡；RP 是指调整拥塞的网络设备，一般是指发送方主机或网卡。图 6 展示了 QCN 的拥塞控制机制。

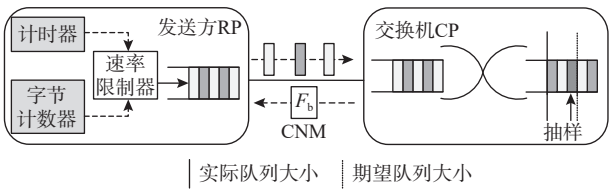


Fig. 6 Congestion control mechanism of QCN

图 6 QCN 的拥塞控制机制

CP 算法支持对交换机出口端口队列大小进行周期性采样以计算当前的拥塞程度 F_b，根据 F_b 的正负决定是否向某抽样数据包的源返回拥塞通知消息（congestion notification message, CNM），数据包抽样的概率与$|F_b|$有关。RP 算法将根据 CNM 以及维护的定时器和字节计数器来对发送方的发送速率进行调整。

DCTCP 算法由 Alizadeh 等人[27]在 2010 年的 SIG-

COMM 会议上提出，是数据中心网络的早期经典拥塞控制算法之一。DCTCP 是一种增强 TCP 对 ECN 反应的算法，TCP 窗口的减小程度取决于 ECN 标记数据包的比例[29]，尽管不支持 RDMA 在数据中心的部署，但其关键思想仍然值得借鉴。

DCTCP 算法的拥塞感知部分对 RED-ECN 方法进行了修改：将 K_{min} 与 K_{max} 设为相同值 K，并以瞬时队列大小采样代替平均队列大小，这样做的好处在于部署简单且发送方反应迅速。具体来说，若瞬时队列长度大于阈值 K，则数据包将被标记为 CE，否则不对数据包做任何处理。接收方为每一个数据包返回一个 ACK 数据包（per-packet ACK），若收到携带 CE 标记的数据包，则在对应的 ACK 数据包头部字段标记 ECN 响应（ECN-echo, ECE）。

DCQCN 的拥塞感知借鉴了 DCTCP 的做法，拥塞调整则同时借鉴了 QCN 和 DCTCP 的做法。DCQCN 的拥塞感知包括 2 个子算法：CP 算法和通知点（notification point, NP）算法，NP 是指将拥塞感知信号返回给 RP 的网络设备，可以是交换机，也可以是接收方。CP 算法定义为当交换机某出端口队列的瞬时大小超过阈值 K 时，交换机将对 ECT(0) 或 ECT(1) 状态的数据包以概率 1 进行标记。NP 算法则指定接收方网卡在 N 微秒内最多生成 1 个拥塞通知数据包（congestion notification packet, CNP），即 N 微秒内仅对 1 个带有 CE 标记的数据包做出反应。为了避免 CNP 在反向路径上排队，DCQCN 将其设置为高优先级传输。

在 2020 年的 NSDI 会议上，清华大学等高校与研究机构提出了 PCN 算法[10]。相较于以前的工作，该算法提出：1）定义交换机的出端口为真拥塞、无拥塞和准拥塞 3 种状态，在此基础上提出了新的 CP 算法。2）将拥塞感知细分为拥塞检测与拥塞识别，其中拥塞检测阶段进行拥塞信息的采集，由 CP 算法实现；拥塞识别阶段利用上述信息对真正的拥塞流进行判断，由 NP 算法实现。

PCN 将真拥塞定义为缓冲区队列积累了大量数据包且出口链路过载的状态；无拥塞则是缓冲区队列无数据包积累，出口链路正常工作的状态；准拥塞端口的缓冲区队列有数据包积累，出口链路由于 PFC 暂停帧和 PFC 重启帧（PFC RESUME frame）的交替作用而处于间歇性过载。以图 4 为例，P6 为真拥塞状态，P5 为无拥塞状态，P3 因为受到 P4 发送的 PFC 暂停帧和 PFC 重启帧的交替影响而处于准拥塞状态。

PCN 定义的 RP 算法负责调整发送速率，CP 算法负责标记数据包，NP 算法负责判断真正的拥塞流、计算接收速率以及返回 CNP。本节仅对为拥塞感知阶段设计的 CP 算法和 NP 算法进行介绍。

CP 算法实现了名为 NP-ECN（none-PAUSE ECN）的标记方法。该方法根据出端口是否处于 PFC 暂停帧抑制状态将数据包分为非暂停状态数据包和暂停状态数据包，前者即为真拥塞端口、无拥塞端口中的数据包以及 PFC 重启帧作用下进入准拥塞端口中的数据包；后者则是 PFC 暂停帧作用下在准拥塞端口中排队的数据包，包含了在 PFC 暂停帧作用期间由于其生效的滞后性导致的新入队数据包以及原本就处于队列中的数据包。对非暂停状态数据包，当交换机出口队列大于 0 时，以 1 的概率标记，否则不标记；对于暂停状态数据包，交换机维护了一个计数器，参数 PN 表示受 PFC 暂停帧作用而排队的数据包数量。当端口收到 PFC 重启帧并再次发送数据包时，PN 随着暂停数据包的发送递减，PCN 不对其标记；当 PN 递减为 0 后，PCN 采用与非暂停状态数据包一致的标记方法。PCN 的字段定义与 ECN 相同。

NP 算法逐流记录了 N 微秒内收到的数据包总数 $recNum$ 以及带 ECN 标记的数据包总数 $ECNNum$。前者随数据包递增，后者随携带 CE 标记的数据包递增，通过计算 $\frac{1}{recNum}ECNNum$ 是否大于 0.95 来做出拥塞判断：若大于 0.95，则将对应数据流判断为拥塞流，标记 CNP 报文 IP 头部 ECN 字段的 CE 位为 1；否则即为未拥塞流，此时的 CE 位标记为 0。

PCN 证明了当数据流发生拥塞时，其最佳发送速率为其接收速率，即接收方在每秒内接收的数据量大小。为了计算接收速率，PCN 接收方除了维护 $recNum$ 以及 $ECNNum$ 外，还会逐流记录 N 微秒内收到的数据包总大小 $recData$，该变量随数据包大小 $pSize$ 递增。对于拥塞流，其接收速率为 $\frac{1}{N}recData$，对于非拥塞流，其接收速率为 0。

不同于 DCQCN 只在接收到 CE 数据包时才返回 CNP 的做法，PCN 在 N 微秒结束时无论是否收到 CE 数据包均返回 CNP、CNP 数据包格式如图 7 所示，发送方通过 CNP 携带的接收速率对其发送速率进行快速而准确的调整，有效地避免了多轮启发式迭代过程。

清华大学和北京国家信息科学与技术研究中心在 2021 年的 SIGCOMM 会议上提出的 TCD 标记算法[12] 同样注重拥塞流的准确识别。与 PCN 类似，TCD



Fig. 7 Packet format of CNP

图 7 CNP 数据包格式

指出正是无损网络所依赖的 PFC 机制混淆了真正拥塞的数据流。为了能够区分拥塞流与受害流，TCD 同样定义了 3 种端口状态：拥塞状态、未拥塞状态和未确定状态。其中未确定状态是指端口处于 ON-OFF 模式（即受 PFC 暂停帧和 PFC 重启帧交替作用的模式）且队列中的数据包排队原因并不明确的状态（即无法确定排队原因是拥塞或 PFC 暂停帧的作用）。

TCD 认为拥塞感知的关键是正确判断 3 种状态的端口之间的相互转换，特别是未确定状态向其他状态的转换。在每个数据包从出端口离开队列时，TCD 都会计算 ON 模式的持续时间 T_{on} 并将其与 $\max(T_{on})$ 进行比较，结合队列长度 *QueueLength* 的演变特征来决定 3 种端口状态之间的转换，表 3 解释了转换过程。其中，ON 模式是指受 PFC 重启帧作用的模式，*LAST_STATE* 用于记录上一个数据包离队时的端口状态，*QueueLength* 表示队列长度，*threshold* 是队列长度阈值。

Table 3 Port State Transition Process Defined by TCD Algorithm

表 3 TCD 算法定义的端口状态转换过程

判断条件				状态转换
$T_{on}<\max(T_{on})$		*LAST_STATE* == 拥塞		拥塞→未确定
		LAST_STATE == 未拥塞		未拥塞→未确定
		LAST_STATE == 未确定		未确定→未确定
$T_{on}>\max(T_{on})$	*LAST_STATE* == 拥塞	*QueueLength* > *threshold*		拥塞→拥塞
		QueueLength < *threshold*		拥塞→未拥塞
	LAST_STATE == 未拥塞	*QueueLength* > *threshold*		未拥塞→拥塞
		QueueLength < *threshold*		未拥塞→未拥塞
	LAST_STATE == 未确定	*QueueLength* > *threshold*	队列未减小	未确定→拥塞
			队列减小	未确定→未拥塞
		QueueLength < *threshold*		未确定→未拥塞

TCD 根据端口状态来对数据包进行标记，若端口处于拥塞状态，则对其中的数据包进行 CE 标记；若端口处于未拥塞状态，则不对数据包进行标记；若端口处于未确定状态，则对其中的数据包进行 UE 标记。CE 标记可以覆盖 UE 标记，而 UE 标记则无法覆盖 CE 标记。TCD 认为网络中通过拥塞端口的数据流均是拥塞流，而仅通过未确定端口的数据流则有可能是受害流。TCD 对字段定义的修改如表 4 所示。

Table 4 ECN Fields and Their Meanings Defined by TCD

表 4 TCD 定义的 ECN 字段及其含义

ECN 字段		含义
ECT 位	CE 位	
0	0	不具备 TCD 传输功能
0	1	具备 TCD 传输功能
1	0	未确定状态
1	1	经历拥塞

华为公司称在 2021 年提出的互联网草案 LDCP[13] 与 RoCEv2 协议已在华为公有云安全运行了 1 年时间。由于能够维持恒定的低队列长度，几乎不发生丢包，因此该草案放弃了作为“第 2 道防线”的 PFC 机制。

LDCP 依据数据包所处的不同阶段被划分为：首 RTT 阶段的数据包（first-RTT-packets），即在第 1 个 RTT 内发送的数据包；稳定阶段的数据包（stable-stage-packets），即第 1 个 RTT 时间之后发送的数据包。前者的 ECN 字段为 00，LDCP 赋予其低优先级，并规定在队列大小超过阈值 K 时主动将其丢弃；后者的 ECN 字段为 10 或 01，LDCP 赋予其高优先级并采用与 DCQCN 相同的标记方法。按阶段划分数据包的主要目的是通过 first-RTT-packets 为数据流的启动探测合适的窗口，避免窗口过大造成网络拥塞或窗口过小造成带宽利用率过低。

接收方网卡对稳定状态数据包有 2 种反应方式：第 1 种是 per-packet ACK，即为每一个数据包生成并返回 ACK 数据包；第 2 种是延迟 ACK（delay ACK），即最多为 m 个数据包生成并返回一个 ACK 数据包，但是如果第 n 个数据包带有 CE 标记，则会立即发送 ACK 数据包（$n \leqslant m$）。对未标记数据包而言，其 ACK 数据包中 ECE 标记为 0；对标记数据包而言，ECE 标记为 1。发送方根据 ACK 数据包中 ECE 标记的不同值采取不同的拥塞调整策略。

除了上述 RDMA 拥塞控制算法外，数据中心早期大量面向 TCP 流量模式的拥塞控制算法也利用 ECN 进行拥塞感知。例如：Vamanan 等人[35] 在 2012 年的 SIGCOMM 会议上提出的 D2TCP 算法在 DCTCP 的基础上引入流量截止时间（deadline）作为除 ECN 标记之外的拥塞信号，二者协同计算得到表示网络拥塞程度与流量截止时间紧迫性的值。Zats 等人[30] 也在 2012 年的 SIGCOMM 会议上提出了 Detail。Detail

是跨层网络堆栈,该方案为交换机的每一个出口队列按优先级维护了一个耗尽字节计数器,该计数器的值表示新到达的数据包前面的同等或更高优先级的字节计数。当计数器的值超过其关联阈值时,交换机将对排队的低优先级数据包进行 ECN 标记。同年, Alizadeh 等人[29]在 NSDI 会议上提出的 HULL 架构利用串联在交换机队列之后的幽灵队列(phantom queue, PQ)来模拟链路占用率, PQ 利用基于链路利用率的 ECN 标记方法提前发出拥塞信号,通过牺牲一部分吞吐量达到显著降低平均延迟和尾部延迟的效果。Wu 等人[71]在 2012 年的 CoNEXT 会议上提出的 ECN*算法则借鉴了 DCTCP 的交换机 ECN 标记方法和标准 ECN 的发送方窗口调整方法,并指出了 ECN*阈值的上下限,实现了与 DCTCP 同样的性能。Stephens 等人[17]于 2014 年在 INFOCOM 会议上提出的 TCP-Bolt 利用与 DCTCP 相同的 ECN 标记方法,抛弃了慢启动过程,有效降低了流量完成时间(flow completion time, FCT)。

相比于丢包等传统的拥塞信号,基于 ECN 的拥塞感知能够将队列大小维持在一个较低的水平,有利于实现低延迟和较高的吞吐量;相比于 RTT,由于 ECN 标记在出口队列处发生,因此能够分离控制环路延迟与排队延迟,使其免受逐渐增大的队列的影响,有利于保持稳定[57]。同时, ECN 也具有以下局限性:

1)ECN 标记是一种粗粒度的拥塞信号,它仅能反映网络是否存在拥塞,并不能说明拥塞的严重程度[3,6,9,20-21,50,65-66]。这导致该类算法需采用启发式的拥塞调整,即经过多轮次的调整才能收敛到合适的速率,不仅会消耗较长的时间,也无法解决突发传输问题。另外该类算法还需要设置大量的参数用于适配不同的运行环境,参数的调整往往是一个复杂且困难的工作[9,66,71]。

2)ECN 标记阈值的恰当与否对性能的影响很大[21,27,36,71-72]:阈值过低有助于保持较短队列并减小延迟,但吞吐量会受到影响;阈值过高有利于提高吞吐量,但会损害延迟。在动态环境下,固定的阈值可能无法适应环境的变化[72]。

3)PFC 触发可能会“污染”队列大小,从而影响 ECN 标记的正确性,即无法识别真正的受害流[10,12]。近年来研究人员已经关注到了该问题并在拥塞控制层面提出了解决方法,例如 PCN 算法和 TCD 算法。此外,还有工作通过抛弃 PFC 机制[7]或替换 PFC 机制[73]同样能够缓解该问题。

2.1.2 INT 标记的显式反馈法

INT 是一种基于网络数据平面对网络状态进行监控的框架[74],其被划分为 3 种应用模式: INT-XD(export data), INT-MX(embed instruct(x)ion), INT-MD(embed data)。由于当前的 RDMA 拥塞控制解决方案大多采用 INT-MD 模式实现,因此本文仅对 INT-MD 模式的拥塞信息采集原理进行阐释。

如图 8 所示,在 INT-MD 模式下,发送方(INT Source)往数据包嵌入 INT 标头;交换机(INT Transit)逐跳向数据包嵌入元数据;交换机(INT Sink)剥离 INT 标头,汇总元数据并将其发送回 INT Source。其中, INT Source 是在数据包中创建并插入 INT 标头的实体, INT Transit 是按照 INT 标头中的指令向数据平面收集元数据的实体, INT Sink 则是提取 INT 标头和相关元数据的实体。

用户可以通过 INT 指令来指定需要收集哪些 INT 元数据,常见的元数据有节点 ID、入端口标识符、入口时间戳、出端口标识符、出口时间戳、跳转延迟、出端口 TX 链路利用率、队列占用率和缓冲区占用率等。相较于 ECN 方法而言, INT 能够反馈的信息更加丰富。

阿里巴巴公司、哈佛大学、剑桥大学和麻省理工学院在 2019 年的 SIGCOMM 会议上提出的 HPCC 算法[9]指出 ECN 标记或 RTT 无法提供精细的拥塞信息,导致性能之间的权衡,因此 HPCC 主张利用 INT 机制获取精确的链路信息,实现精准的拥塞控制。HPCC 利用 INT-MD 模式感知拥塞的过程如下: INT

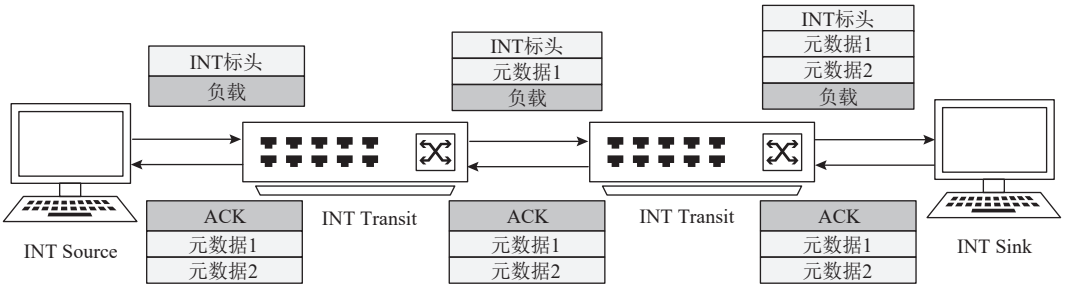


Fig. 8 Forwarding mode of INT-MD

图 8 INT-MD 转发模式

Source 将数据包发送到网络中，INT Transit 逐跳向数据包中插入元数据，最后由 INT Sink 汇总所有元数据并将其插入 ACK 数据包中，INT Source 根据 ACK 数据包中携带的详细信息进行拥塞调整。

HPCC 在具有 P4 可编程功能的交换机 ASIC 上实现了 INT 功能，需测量网络状态主要有：1)链路带宽容量 *B*(4 b)，表示当前出端口的速率(40 Gbps, 100 Gbps 等)，是一个枚举类型字段。2)时间戳 *TS*(24 b)，表示数据包离开当前出端口的时间。3)传输字节 *txBytes*(20 b)，表示从当前出端口发送的累计字节(128 B)，4)队列长度 *qlen*(16 b)，表示当前出端口的队列长度(80 B)。由于数据中心规模有限，HPCC 规定使用 40 B 的数据包标头对至多 5 跳的元数据进行记录。HPCC 的标头信息如表 5 所示。

Table 5 Header Information of HPCC

表 5 HPCC 的标头信息

HPCC 标头字段	字段大小/b	字段含义
nHop	4	路径上的交换机跳数
pathID	12	路径上所有交换机 ID 异或
B	4	出端口的速率（枚举值）
TS	24	出端口发出的时间戳
txBytes	20	出端口发送的累计字节
qlen	16	出端口当前队列长度

国防科技大学在 2021 年提出的 FastTune 算法[66]关注了数据中心流量在时间和空间维度上的动态变化。通过 INT 机制感知细粒度拥塞信号以解决数据流在空间维度上动态变化的问题，FastTune 采集的信息与 HPCC 相同。通过在第 1 个 RTT 内由交换机生成并返回携带 INT 元数据的 ACK 数据包(S-ACK 机制)，第 1 个 RTT 之后由交换机向来自接收方的 ACK 数据包插入 INT 元数据(H-ACK 机制)来解决时间维度动态变化的问题。

S-ACK 机制的目的是使发送方主机在第 1 个 RTT 内感知到网络拥塞。如图 9 所示，当数据流第 1 个数据包到达交换机时，交换机生成一个 S-ACK 数据包，并在出口处将 INT 元数据插入到 S-ACK 数据包并返回发送方。交换机维护一个名为 *NewACK* 的变量，该变量在交换机产生 S-ACK 数据包或转发 H-ACK 数据包后被设置为 False，表示交换机不再为后续到来的数据包做出响应。

H-ACK 机制的目的则是确保发送方获得最新的拥塞信息。其做法是，接收方主机生成 H-ACK 数据包并返回，交换机采集反向链路入端口，即正向链路

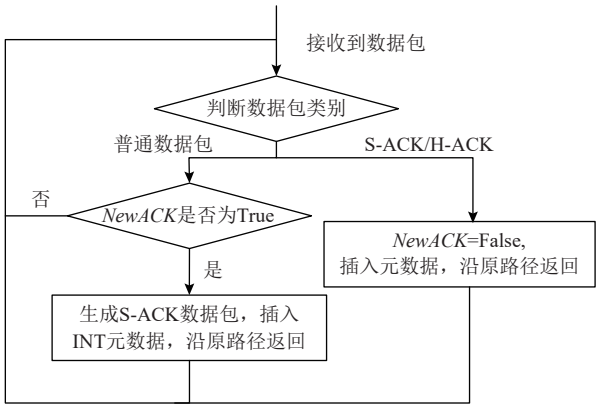


Fig. 9 Switch processing logic of FastTune

图 9 FastTune 的交换机处理逻辑

出端口处的拥塞信息，并把 INT 元数据插入 H-ACK 数据包中。值得注意的是，实现上述机制需要 ACK 数据包与其对应的数据包路径互相对称，同时需要交换机支持跨端口采集信息。

维也纳大学、柏林工业大学和普林斯顿大学在 2022 年的 NSDI 会议上发表了 POWERTCP 算法[51]，该算法将网络类比为电路，网络的绝对属性类比为电压(例如队列大小、RTT、ECN 等)，网络的变化属性类比为电流(例如队列大小变化、RTT 梯度等)。POWERTCP 证明了单纯基于电压的方案具有唯一收敛点，稳定性较好，但是处理抖动的能力较弱、收敛速度慢；与此相反，单纯基于电流的方案收敛较快，抖动较小，但是不存在唯一的收敛点。因此，POWERTCP 提出基于功率的方案，功率定义为电压与电流之积，从网络的角度看，该方案旨在同时对网络的绝对属性和变化属性做出响应，从而获取二者各自的优点。

在 POWERTCP 方案中，电压的定义是瓶颈链路的带宽延迟积(bandwidth delay product, BDP)与队列字节数之和，电流的定义是瓶颈链路的总发送速率与队列变化率之和，POWERTCP 证明在该定义下，功率等于瓶颈链路带宽与总发送窗口的乘积，即发送窗口与功率之间存在直接联系，因此可以通过计算功率来调节发送窗口。

POWERTCP 利用与 HPCC 相同的 INT-MD 机制来为功率计算提供必要的链路信息，采集的元数据以及整体流程与 HPCC 相同，即二者的拥塞感知相同。在第 3 节将阐述 POWERTCP 如何有效利用上述元数据来设计拥塞调整策略。

斯坦福大学和谷歌公司的研究人员在 2023 年的 NSDI 会议上提出的 Bolt 算法[67]关注到了 3 个问题，分别是控制环路过长的问题、未完成数据流无法及时抢占完成数据流释放的带宽导致的吞吐量降低

问题以及每个 RTT 仅进行 1 次的窗口调整导致的收敛缓慢问题。本节通过在交换机入端口处生成并返回带 INT 信息的次 RTT 控制(Sub-RTT control, SRC)数据包来解决第 1 个问题;通过基于信令的主动速率提升(proactive ramp up, PRU)以及带宽供应匹配(supply matching, SM)来解决第 2 个问题。

PRU 的目的是帮助其余流量快速抢占由完成流量释放的带宽:*last* 字段为“1”的数据包表示其所属流即将结束,当交换机收到该类型数据包时,将递增 *pru_token* 值;SM 的目的是帮助流量充分利用链路剩余带宽,交换机会记录相邻数据包之间的间隔,供应量为带宽与间隔的乘积,需求量为数据包大小,*sm_token* 的增量即为二者的差值。表 6 展示了 Bolt 需要维护的标头信息。

Table 6 Header Information of Bolt

表 6 Bolt 的标头信息

Bolt 标头字段	字段大小/b	字段含义
q_size	24	入端口的队列大小
link_rate	8	入端口的链路速率
data	1	普通数据包
ack	1	ACK 数据包
src	1	SRC 数据包
last	1	属于最后一个发送窗口
first	1	属于第一个发送窗口
inc	1	发送方应增大窗口
dec	1	发送方应减小窗口
reserved	1	保留字段
t_data_tx	32	原始数据包的发送时间戳

图 10 为 Bolt 定义的交换机处理逻辑:收到数据包后交换机入端口首先计算供应令牌 *sm_token*,其次判断当前队列大小 *cur_q_size* 与阈值 CC_{THRESH} 大小,若超过了阈值且 *pkt.dec* 标记为 0,则对数据包的 *q_size*, *link_rate*, *t_data_tx* 字段进行更新(对应图 10 的①),否则不进行任何操作。若未超过阈值且 *pkt.last*=1,当 *first* 字段为 0 时,将 *pru_token* 加 1(对应图 10 的②),否则也不进行任何操作。最后,当 *pkt.inc*=1 且 *pru_token*>0, *sm_token*>*MTU*,则保持 *pkt.inc* 字段不变,分别消耗一个 *pru_token* 和 *sm_token*(对应图 10 的③),否则将 *pkt.inc* 字段置为 0。

综上所述,Bolt 利用控制环路较短的 SRC 数据包进行窗口递减,利用控制环路较长的 ACK 数据包进行窗口递增。

相比于 ECN 方法,INT 能够提供更加丰富的信息,具有以下优势:1)相较于多轮迭代的启发式算法,基于 INT 的算法有利于发送速率或发送窗口快速收敛,避免反应过度或反应不足[67];2)减少了参数的数量,提高了算法在不同环境下的适应能力[67];3)可以利用 INT 信号将正反向路径上的拥塞信息分离,避免了反向路径延迟可能带来的影响[41]。

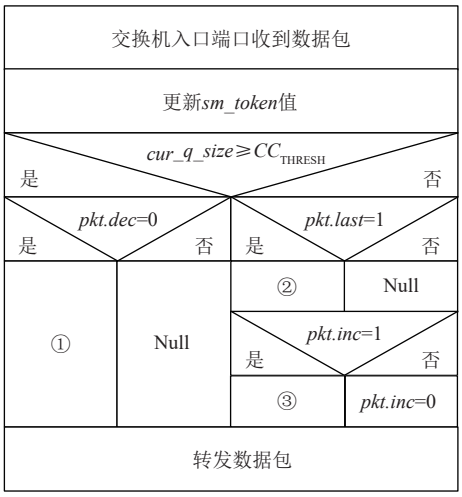


Fig. 10 Switch processing logic of Bolt

图 10 Bolt 的交换机处理逻辑

INT 也有以下不足之处:1)依赖硬件支持[3,9,67,75],有可能导致因交换机对 INT 的支持不同而使网络异构;2)在不修改终端主机与交换机的情况下可能与 RoCEv2 不兼容[3];2)向数据包插入元数据会导致交换机花费更长的时间处理数据包[75];3)数据包携带大量的元数据信息会对网络造成更大的负担[75];4)不同的路径使得数据包携带的元数据数量不同,导致数据包头大小不同,这与当前某些处理固定大小数据包头的主机堆栈互相矛盾[75]。

针对 INT 机制开销过大的问题,近年来学术界提出了概率网络带内遥测(probabilistic in-band network telemetry, PINT)机制,这部分内容将在第 4 节介绍。

2.1.3 网内反馈法

除了 ECN 标记和 INT 标记外,交换机主动计算的速率、主动分配的信令或其他信号同样可视为显式反馈的一种,本节对其中的部分方案进行简要介绍,并总结其优缺点。

针对发送方驱动的方案反馈延迟较长,无法快速获得拥塞信号的问题,2020 年,由思科公司、普渡大学、提契诺大学在 CoNEXT 会议上提出的 RoCC 算法[11]把速率计算任务由端侧转移到了交换机,发送方直接利用交换机 CNP 携带的速率进行调整。

如图 11 所示,在 RoCC 设计中,交换机额外包括

速率计算器、CNP 生成器和流量表 3 个组件。速率计算器通过定期采样队列大小 Q_{cur}，结合最大队列大小 Q_{max}、参考队列大小 Q_{ref} 等参数，对流速率进行计算。CNP 生成器将速率计算结果封装成 CNP 报文，并通过查询流量表，将其发送回发送方。

RoCC 的速率计算过程包括乘法递减（multiplicative decrease, MD）和比例积分（proportional integral, PI），相关变量及计算逻辑如图 12 所示。

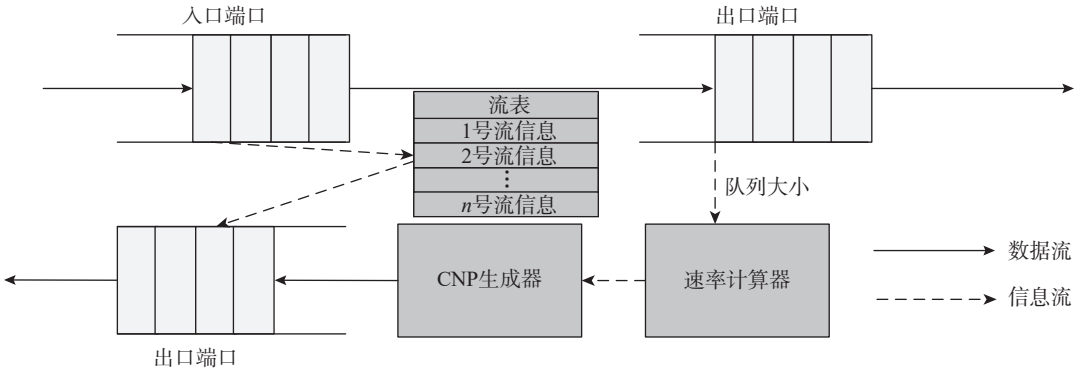


Fig. 11 Switch framework of RoCC

图 11 RoCC 交换机框架

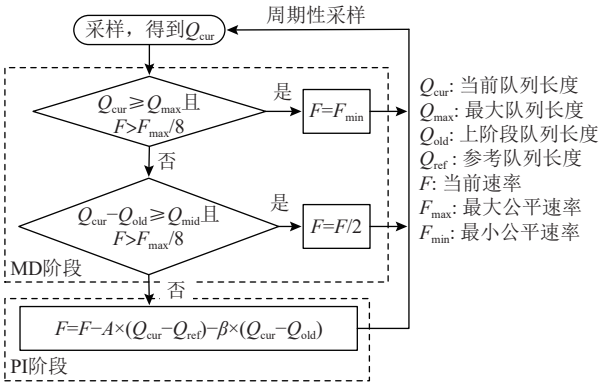


Fig. 12 Rate calculation process of RoCC

图 12 RoCC 的速率计算过程

北京邮电大学、华为公司和紫金山实验室在 2021 年的 APNet 会议上提出的 HierCC 算法[49] 认为数据中心的流量不确定性是大多数拥塞控制算法无法同时满足高吞吐量和确定性延迟的根本原因，在此基础上，HierCC 指出拥塞控制算法应具有聚合流量、短控制环和积极主动的速率分配 3 个特征。

HierCC 的框架如图 13 所示，ToR（top of rack）交换机实现了虚拟输出队列（virtual output queues, VOQ），其数量与网络中主机的总数 N 相同；除了 VOQ 外，ToR 交换机维护下游输出队列（downstream output queue, DOQ），其数量与该 ToR 交换机连接的主机数量 k 相同；ToR 交换机内部实现了 1 个 VOQ 调度器和 k 个 DOQ 调度器，VOQ 调度器用于向网络发送数据包，DOQ 调度器负责向 VOQ 发送信用（credits）。

聚合流量是指将来自相同 ToR 交换机、去往相同接收方的数据包聚合到同一 VOQ 中排队，有利于缓解流量的不确定性；短控制环是指发送主机的速率由第 1 跳 ToR 交换机来控制；主动速率分配是指 ToR 交换机之间利用信令实现聚合流的速率控制，发送方的速率由第 1 跳 ToR 交换机的聚合流速率计算得到。综合上述 3 种机制，HierCC 实现了更快速更准确的拥塞控制，有效降低了流量完成时间。

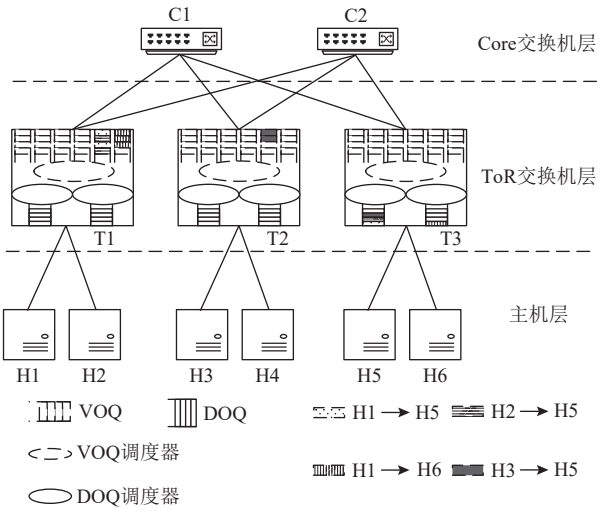


Fig. 13 Overall framework of HierCC

图 13 HierCC 整体框架

北京邮电大学、华为公司和紫金山实验室在 2022 年的 INFOCOM 会议上提出的 PACC 算法[52] 认为控制环路过长以及启发式调整算法阻碍了数据中心 RDMA 的大规模部署，在借鉴 RoCC 算法的基础上，PACC 格外关注部署问题。为此，PACC 仅对交换机逻辑进行修改，而终端则与 DCQCN 兼容。

PACC 的交换机算法主要由 3 个后台周期性进

程和 1 个函数组成：首先是周期性流表更新，当交换机收到上游数据包时，根据当前队列长度 Q_{cur} 和队列阈值 Q_{th} 对流表的字段进行更新；其次是周期性 CNP 数量计算，基于 PI 控制器对所有拥塞流共享的 CNP 总数 N_{all} 进行计算；再次是周期性平均队列大小计算，通过对比平均队列大小 Q_{avg} 和突发性队列阈值 Q_{burst} 区分并处理突发情况；最后是基于拥塞程度的 CNP 分配与生成函数，通过 3 个后台进程得到的数据实现基于权重的 CNP 分配，达到精确控制拥塞的效果。图 14 展示了上述 3 个过程及 1 个函数的联系。

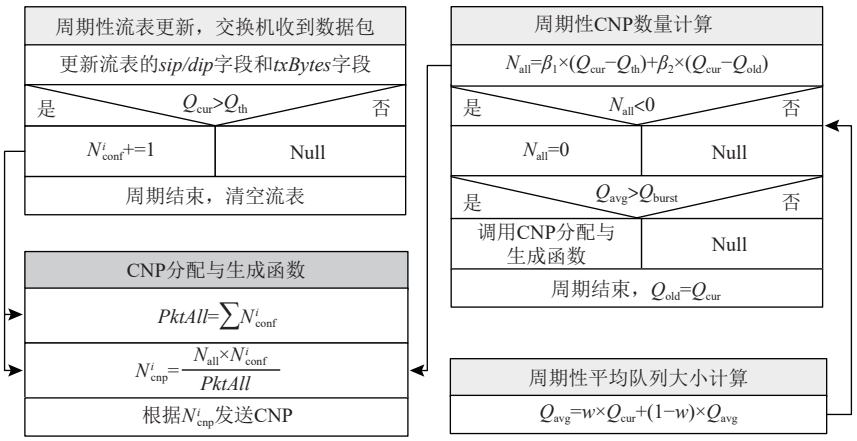


Fig. 14 Processing logic of PACC switch

图 14 PACC 交换机的处理逻辑

基于网内反馈的拥塞感知方法由交换机直接计算并反馈速率等信息，既缩短了控制环路，又能避免多轮启发式调整带来的诸多问题[3]；然而其复杂性从端侧转移到交换机实现使得部署难度更大，成本更高[3]；另外，交换机对速率等网内反馈的计算通常也是采取启发式算法，尽管不需要多轮迭代，但同样需要仔细设计参数以满足不同情况下的目标[51]。

2.2 基于延迟的感知

作为一个简单且有效的信号，延迟在数据中心 RDMA 拥塞控制中得到了较好的应用。其简单性体现在延迟作为衡量网络性能的基本要素，几乎所有类型的网络都提供了测量条件，不需要交换机的支持；其有效性体现在目前的大部分网卡都支持高精度硬件时间戳功能，延迟测量达到了微秒级别[76-80]，通过避免软件时间戳引入的额外耗时，延迟能够更加准确地反映网络状况[50]。目前大部分基于延迟的拥塞控制研究中，大多采用测量 RTT 的方式，因此下文将以 RTT 代替延迟进行论述。

谷歌公司于 2015 年在 SIGCOMM 会议上提出的 TIMELY 算法[65]是第 1 个利用 RTT 进行数据中心 RDMA 拥塞控制的经典算法，这主要得益于网卡支持高精度时间戳以及硬件生成 ACK 数据包，避免了因测量误差以及主机延迟带来的影响。

如图 15 所示，TIMELY 以数据段为单位进行突发传输，一个数据段本质上是一个固定大小的字节序列，接收方网卡为一个数据段而非单个数据包返回一个 ACK 数据包。

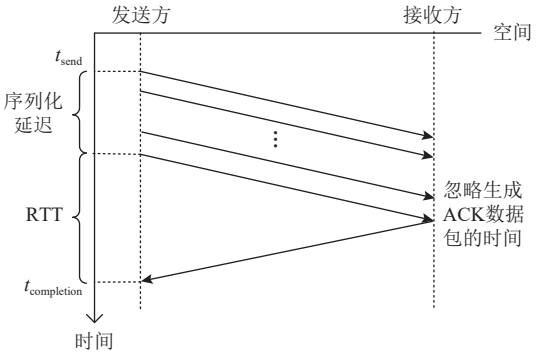


Fig. 15 Data segment sending process and RTT calculation of TIMELY

图 15 TIMELY 的数据段发送流程与 RTT 计算

TIMELY 对 RTT 的定义不同于以数据包为传输单位产生的定义：发送方主机将数据段第 1 个字节的发送时刻记录为 t_{send}，将对应于该数据段的 ACK 数据包的到达时间记录为 $t_{completion}$，二者都是被主机读取的硬件记录的时间。假设数据段大小为 *seg.size* 且该数据段以网卡线速率发送，RTT 的定义如式(1)所示。

$$RTT_{\text{TIMELY}}=t_{\text{completion}}-t_{\text{send}}-\frac{seg.size}{\text{NIC line rate}}。\tag{1}$$

发送方根据 ACK 数据包携带的 RTT 信息来计算 RTT 变化梯度并基于梯度指导拥塞调整，拥塞调整部分将在第 3 节中进行介绍。

威斯康星大学麦迪逊分校和伊利诺伊大学芝加哥分校在 2018 年的 SoCC 会议上提出的 RoGUE 算法[20] 意图通过重新设计软件实现的拥塞控制和丢包恢复机制来摆脱对 PFC 的依赖。其拥塞感知部分与 TIMELY 类似，传输单位都是数据段而非数据包，因此 RTT 的计算与式(1)相同，在此不予赘述。

2020 年，谷歌公司在 TIMELY 的基础上提出 Swift 算法[50]，该算法以 RTT 作为拥塞感知信号，在具有内核旁路功能的 Pony Express 协议栈中实现。与 TIMELY 相比：1)Swift 的发送单位是数据包而不是数据段；2)Swift 利用主机软件时间戳和网卡硬件时间戳对多个时刻进行记录；3)Swift 不仅关注网络的拥塞情况，还格外关注主机内部的拥塞情况。

图 16 展示了 Swift 数据包收发过程以及划分的不同延迟组成部分：①是发送方网卡发送延迟，表示数据包在网卡中等待发送的时间，可忽略；②是正向链路网络延迟，表示正向链路上序列化、排队和传播的时间；③是接收方网卡接收延迟，表示数据包被处理前在网卡中的等待时间；④是处理延迟，表示协议栈处理数据包以及生成 ACK 数据包的时间；⑤是接收方网卡发送延迟，表示 ACK 数据包等待发送的时间；⑥是反向链路的网络延迟，正反链路通常不对称；⑦是发送方网卡接收延迟，表示 ACK 数据包被发送方协议栈处理前的等待时间。综上可知：②与⑥反映网络延迟、③与④反映远程主机延迟。

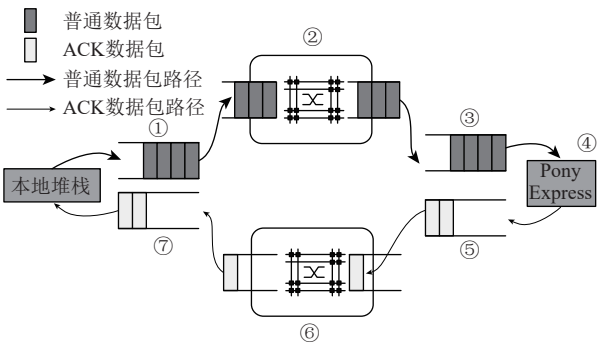


Fig. 16 Complete data sending and receiving process and RTT components of Swift

图 16 Swift 完整的数据收发过程以及 RTT 组成部分

结合上述定义，Swift 测量 RTT 及划分延迟的方法如图 17 所示，1)发送方主机记录数据包的发送时间 t_1(当网卡就绪时才将数据包交付网卡，即交付时刻等于发送时刻)；2)接收方网卡记录数据包的到达时间 t_2；3)接收方主机记录数据包开始被主机堆栈处理的时间 t_3；4)接收方主机记录 ACK 数据包的发送时间 t_4(ACK 数据包的交付时刻等于发送时刻)；5)发送方网卡记录 ACK 数据包的到达时间 t_5；6)发送方主机记录 ACK 数据包开始被主机堆栈处理的时间 t_6。

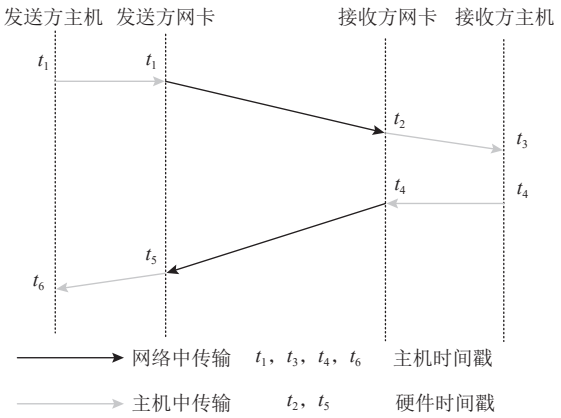


Fig. 17 RTT calculation process of Swift

图 17 Swift 的 RTT 计算过程

RTT 定义为 t_6-t_1，包括网络中的序列化、排队、传播时延，以及主机内的排队与处理时延。为了准确反映网络状况，Swift 定义了端内延迟和网络延迟。端内延迟为接收方的排队及处理时间(t_4-t_2)与本地排队时间(t_6-t_5)之和，即 $t_6+t_4-t_5-t_2$(接收方会把 t_4-t_2 插入 ACK 数据包中返回)；网络延迟为 RTT 减去端点延迟，即 $t_5+t_2-t_4-t_1$。

北京邮电大学和紫金山实验室在 2021 年的 ICNP 会议上提出的 RCC 算法[22] 利用单向延迟(single trip time, STT)信号将数据中心拥塞划分为最后 1 跳拥塞以及网内拥塞，并分别利用接收方显式窗口分配机制和比例积分微分(proportion integral differential, PID)窗口控制机制解决 2 种拥塞情况。RCC 将拥塞感知细分为拥塞判断机制和拥塞划分机制，前者利用 STT 对是否发生拥塞进行判断，后者利用 STT 内收到的总字节数对拥塞类别进行划分。

1)拥塞判断机制。RCC 在每个数据包头部插入发送时间戳，接收方根据该时间戳计算得到 STT。如式(2)所示，若连续 n 个 STT 均大于基本单向延迟(baseSTT)的 $1+\delta$ 倍，则可判断为发生拥塞。

$$STT_i > (1+\delta)\times baseSTT_i \text{。} \tag{2}$$

2)拥塞划分机制。接收方将每一轮 STT 内收到的总字节数记录为 B，若 B 大于最后 1 跳的单程带宽延迟积的 η 倍(η 表示预期链路利用率)，说明最后 1 跳带宽被充分利用，此时的拥塞类型为最后 1 跳拥塞，否则为网内拥塞。

南京大学于 2022 年在 ICNP 会议上提出的 RET

算法[21]证明了固定的目标延迟不利于延迟和吞吐量的博弈以及队列稳定性,因此 RET 尝试利用交换机提供的 ECN 标记来估计队列大小(标记阈值为 0),并实时更新目标延迟。RET 的 ECN 标记并不代表网络存在拥塞,而是表明队列是否为空:若数据包被标记,说明队列不为空,即存在排队延迟;反之则不存在排队延迟。

通过实时更新目标延迟,RET 将存在排队延迟的数据包(即被标记数据包)比例维持在 η 附近,根据经验,当 η=0.99 时,目标延迟达到最佳状态。RET 的接收方算法如图 18 所示,收到带 ECN 标记的数据包表示路径存在排队情况,此时需要降低目标延迟,其中 δ_t 表示更新步长;收到未携带 ECN 标记的数据包表示路径不存在排队情况,此时需要提高目标延迟,其中 *ConN_ECN* 用于限制提高次数,避免过度反应。

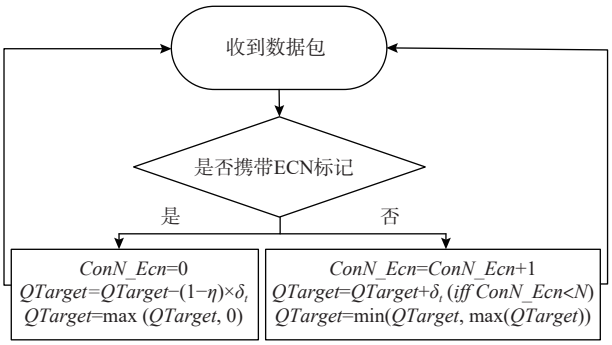


Fig. 18 The process of calculating the target delay for RET

图 18 RET 的目标延迟计算过程

面向互联网的 TCP 拥塞控制方案在发展早期便把延迟应用到拥塞感知中,由于互联网规模相较数据中心而言较大,延迟较高,因此对时间戳的精度要求较低,且能够容忍端系统内部因中断、调度等机制带来的额外延迟影响。相较而言,数据中心因规模小、高带宽、流量动态变化等特点对时间测量的精度和终端的处理速度具有很高的要求。鉴于互联网的 TCP 拥塞控制机制在一定程度上对数据中心的方案有所启发,本节在下面对早期的互联网和数据中心部分基于延迟的 TCP 解决方案进行简单介绍。

Brakmo 等人[81]于 1994 年提出的 TCP Vegas 算法是最早将延迟引入互联网 TCP 拥塞控制的方案,通过粗粒度测量 RTT,结合 baseRTT 与当前窗口,可以分别计算得到实际吞吐量与期望吞吐量,并根据二者的差值决定如何修改发送窗口。Wei 等人[82]于 2006 年提出了 FAST TCP 算法,该算法指出互联网中丢包等二进制拥塞信号表示拥塞程度的能力有限,通常会导致拥塞窗口和瓶颈队列振荡,即稳定性较差。而利用多比特的拥塞信号则有利于实现高利用率、公平性和稳定性等目标。该算法同时利用丢包与 RTT 来调整发送窗口。Tan 等人[83]于 2006 年在 INFOCOM 会议上提出的 CTCP 算法专门为高速长距网络设计,在基于丢包的标准 TCP 拥塞控制算法中添加测量延迟的组件,借鉴 TCP vegas 的思想,根据测量得到的延迟计算预期吞吐量与实际吞吐量的差值,不仅实现了带宽的有效利用,还能够实现 RTT 公平性和 TCP 公平性。

相对于需要交换机支持的显式反馈信号,RTT 优势体现在:1)RTT 的测量并不需要交换机的支持,对于算法的部署而言,该优势所带来的好处十分明显[50,64-65];2)RTT 能够直接反映链路上所有交换机中数据包因排队带来的延迟[64],即 RTT 能够直接反映拥塞情况,相比之下,ECN 等显式反馈信号可能会因为 QoS 等机制的影响无法反映真实的拥塞情况[65];3)利用多比特信息对 RTT 进行编码不仅能够表示拥塞的存在,还可以以较细的粒度反映拥塞严重程度[50,64-65];4)在 RTT 的基础上计算得到的 RTT 梯度能够较好地体现拥塞的发展趋势,基于 RTT 梯度的拥塞控制有利于快速收敛且抖动较小[51,57];5)RTT 有助于区分网络拥塞和主机拥塞[11,50]。

RTT 也有 5 个缺陷:1)尽管 RTT 能够反映网络和主机的拥塞程度,但是与 INT 等信号相比,其丰富性并不足以驱动发送方将速率或窗口直接调整到目标值,因此基于 RTT 的拥塞调整大多也是采用加性增大乘性减小(addictive increase and multiplicative decrease, AIMD)或乘性增大乘性减小(multiplicative increase and multiplicative decrease, MIMD)启发式算法[3,50,64-65]。2)RTT 通常会将反向链路的延迟包含在内,对发送方而言反向链路延迟是无关紧要的,甚至会带来一定的影响,因此需要采取额外的补救措施,例如赋予 ACK 数据包更高的传输优先级[50]。这并不是 RTT 特有的缺点,所有端到端的解决方案都会面临该问题。3)基于 RTT 的拥塞感知依赖于 RTT 的测量精度,这通常会对主机网络协议栈和网卡硬件功能提出额外的要求,需要同步主机时间与硬件时间。4)利用 RTT 进行拥塞调整的算法通常需要与 RTT 阈值进行比较,与 ECN 标记阈值类似,恰当的 RTT 阈值选择通常是一项困难的工作[65]。5)简单地基于 RTT 绝对值与 RTT 阈值的差值来对拥塞程度进行评估不利于算法适应抖动较大的网络环境,收敛速率较慢;而简单地基于 RTT 梯度进行拥塞感知则可能导致算法不存在单一且固定的收敛点[51,57]。

3 拥塞调整

拥塞感知为拥塞调整提供了依据，拥塞调整则直接决定了拥塞控制算法的效果。缓解拥塞的方法通常有 3 种：1)降低发送方的发送速率；2)减小发送方的发送窗口；3)同时降低发送速率和减小发送窗口。本节将根据上述 3 种划分方式阐述现有的拥塞控制算法，并分析各自的优缺点。

3.1 基于发送速率的拥塞调整

为了实现高吞吐、低延迟和快速的公平收敛性，DCQCN[8] 的 RP 算法借鉴了 DCTCP 和 QCN 的思路，设计了一个基于参数 α 的“折半”降速过程和 3 个逐渐激进的提速过程，并维护了表 7 中的变量和参数用于控制速率调整。其中：参数 *timer* 表示若在该参数规定时间内未收到 CNP，则触发一次速率提升事件；*bytes_counter* 表示当发送字节达到该参数规定的数量且未收到 CNP 时触发一次速率提升。DCQCN 将速率提升分为 3 个阶段，分别是快速恢复、主动增加和超增量增加。参数 F 和变量 T, BC 用于控制不同阶段之间的跳转。

如图 19 所示，①当收到 CNP 时，DCQCN 进入速率下降阶段，分别对 R_T, R_C, α 进行计算。其中，g 是 0~1 之间的恒定值，用于对 α 进行指数加权移动平均(exponential weighted moving average, EWMA)运算。②当 *timer* 或 *bytes_counter* 被触发且 T, BC 均小于 F，则进入快速恢复阶段，根据保存的 R_T 值将当前速率快速恢复到速率下降前的水平。③*timer* 或 *bytes_counter* 被触发且 F 位于 T 与 BC 区间内，此时处于主动增加阶段，对速率进行缓慢提升。④*timer* 或 *bytes_counter* 被触发且 T, BC 均大于 F，此时超主动增加以快速抢占可用带宽，其中 i 表示触发此次超主动增加的 T 值或 BC 值。

Table 7 Variables and Parameters Adjusted by DCQCN Related Rates

表 7 DCQCN 相关速率调整的变量与参数

名称		含义
变量	R_T	目标速率，用于快速恢复
	R_C	当前速率
	α	拥塞程度
	T	记录 *timer* 被触发的次数
	BC	记录 *bytes_counter* 被触发的次数
参数	R_{AI}	速率增量，用于主动增加
	R_{HAI}	超速率增量，用于超主动增加
	timer	相邻速率提升事件之间的最小间隔
	bytes_counter	相邻速率提升事件之间的最小字节数量
	F	控制 3 个速率提升阶段的跳转

DCQCN 通过建立流体模型，理论上证明了数据包的实际标记概率接近于 0，这说明在 K_{min} 附近存在固定队列长度点，即 DCQCN 能够有效地将队列长度控制在较低水平；此外，DCQCN 存在所有流量获得公平带宽的固定点，即 DCQCN 能够实现公平性。

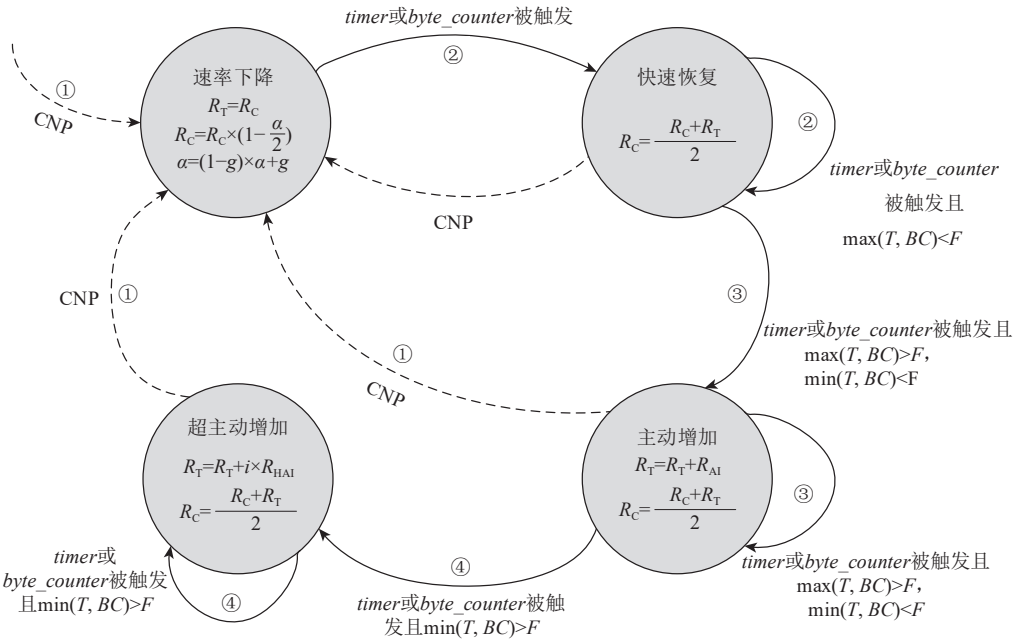


Fig. 19 Rate-adjusted state machine of DCQCN

图 19 DCQCN 的速率调整状态机

DCQCN 重点对吞吐量进行研究，实验结果表明 DCQCN 与 PFC 相结合的方案能够有效提高拥塞情况下的吞吐量；另外，由于将队列长度维持在较低水平，DCQCN 也能降低通信延迟。

TIMELY[65] 除了关注高带宽和低延迟的性能表现，还格外强调 RTT 在数据中心拥塞控制领域的价值，利用低时间阈值 T_{low} 和高时间阈值 T_{high} 将拥塞的调整划分为 3 个阶段，分别是加性增大（addictive increase, AI）阶段、基于 RTT 梯度的增大/减小阶段和 MD 阶段。过程如图 20 所示。

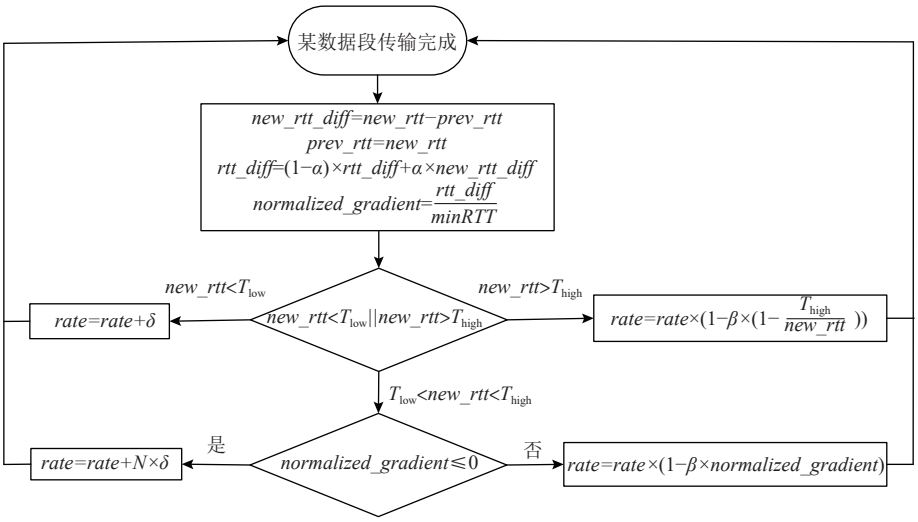


Fig. 20 Rate adjustment process of TIMELY

图 20 TIMELY 的速率调整过程

当数据段传输事件完成并收到来自接收方的 ACK 数据包时，发送方利用新测量的 RTT 值（*new_rtt*）减去旧的 RTT 值（*prev_rtt*）得到 RTT 差值（*new_rtt_diff*），并对 *new_rtt_diff* 进行 EWMA 处理，随后除以最小 RTT（*minRTT*）得到归一化 RTT 梯度（*normalized_gradient*）。其中，*minRTT* 是指理想网络环境下的 RTT 值。

发送方通过对比 *new_rtt* 与 T_{low} 和 T_{high} 的大小来决定进入不同的速率调整阶段。若 *new_rtt*<T_{low}，则进入 AI 阶段，其中 δ 表示增量常数；若 *new_rtt*>T_{high}，则进入 MD 阶段，其中 β 表示 MD 因子。若 *new_rtt* 在 T_{low} 和 T_{high} 之间，则进入基于 RTT 梯度的增大/减小阶段。在该阶段中，若 *normalized_gradient*<0，说明此时队列具有减小的趋势，进入 AI 阶段；反之，若 *normalized_gradient*>0，说明队列具有增大的趋势并进入 MD 阶段。

TIMELY 证明了与单一且固定的目标延迟法相比，RTT 梯度方法能够同时实现高吞吐量与低延迟目标，且队列大小的波动更为平缓。实验结果表明，在数百台服务器组成的 Clos 网络拓扑中，TIMELY 的平均 RTT 和尾延迟相较于 DCTCP 算法分别降低了 1 000% 和 1 300%，且并未观察到数据包丢失和 PFC 触发的情况。

PCN 算法[10] 的核心是提高算法在突发流量下的表现，尽可能避免触发 PFC 机制，为此 PCN 设计了新颖的拥塞识别和速率计算策略并将其转移到接收方执行，因此 RP 算法比较简洁：在收到 CNP 后，发送方首先检查其 ECN 字段的 CE 位。若 CE 为 1，此时发送速率以式(3)(4)进行调整。其中权重参数 w 用于控制 PCN 速率增长的激进程度。在发生拥塞时，w 应设置为最小值 w_{min}，代表此后的一段时间内的速率以温和的方式增长，该做法有利于发生拥塞后对速率的更新不会触发 PFC 机制。

$$w = w_{min}, \tag{3}$$

$$sendRate = \min\{sendRate, recRate \times (1-w)\}。 \tag{4}$$

若 CE 为 0，发送速率以式(5)(6)进行调整。权重参数 w 随着非拥塞调整次数的增多而逐渐增大，表示速率调整由温和逐渐过渡到激进状态，该做法有利于发送方快速抢占可用带宽，避免资源浪费。

$$sendRate = sendRate \times (1-w) + lineRate \times w, \tag{5}$$

$$w = w \times (1-w) + w \times w_{max}。 \tag{6}$$

得益于接收方对拥塞流的准确识别和接收方驱动的速率调整，PCN 能够迅速稳定队列大小，为流量分配公平带宽并降低突发情况下的 PFC 触发概率。在由 3 对服务器和 1 台交换机组成的哑铃拓扑环

境中，PCN 在抑制 PFC 暂停消息方面比 DCQCN 和 TIMELY 分别减少了 47% 和 90%；在快速收敛方面则比 DCQCN 和 TIMELY 快 25 倍和 45 倍；在降低延迟方面表现最佳，最多可相应减少 1 130% 和 1 320% 的延迟。

作为交换机驱动的拥塞控制算法，RoCC 算法[11]通过快速接收并利用拥塞信号，其设计目标更为广泛，除高带宽和低延迟外还能够满足公平性、稳定性和快速收敛的特性。

RoCC 的发送方并不需要进行复杂的操作。其拥塞调整主要包括 2 个部分：1）参考 CNP 携带的速率进行直接调整。当 RoCC 发送方收到来自交换机的 CNP 报文时，它首先检查当前 CNP 的源交换机是否与上一次进行速率调整参考 CNP 的源交换机相同，同时检查当前 CNP 携带的速率值是否小于当前的发送速率，若二者满足其一，则将发送速率替换为 CNP 携带的速率，该做法能够保证 RoCC 每次都对瓶颈链路做出响应。2）进行快速恢复操作。执行该操作的前提是长时间未收到满足上述条件的 CNP，发送方判定网络状况良好并以指数级增长当前的发送速率，直到发送速率达到最大值。利用上述机制，RoCC 能有效处理多瓶颈场景下的拥塞，同时有利于快速收敛和达到高带宽利用率。

在 N 个源节点通过一个交换机连接到单个目标节点的网络拓扑环境中，RoCC 与 DCQCN 相比降低了多达 700% 的 PFC 帧生成，实现了比 DCQCN 低 800% 的尾部延迟，并且在公平性、稳定性和收敛性方面均优于 DCQCN，TIMELY，HPCC。

清华大学等院校在 2024 年 NSDI 会议上发表的 ACC 算法[14]基于 TCD 的拥塞感知算法，针对无损网络拥塞控制提出 2 个关键设计原则：1）在无损网络中，ACK 数据包序列和到达速率是对排队数据包的数量和发送速率进行推断的工具；2）对拥塞流进行时间充足的暂停是抑制头阻塞的首要手段，同时受害流应根据拥塞程度调整发送速率。在这 2 个设计原则指导下，ACC 分别对拥塞流、受害流和未拥塞流采取不同的调整策略。

1）对拥塞流采取先暂停后启动的措施的关键在于暂停时间和启动速率的计算。如表 8 所示，为了计算暂停时间，ACC 首先要知道 RTT_1 和 RTT_2 积累的未确认数据包数量，前者是 RTT_1 发送的数据包总数 ΔS 与 RTT_2 接收的 ACK 总数 ΔA 之差；后者是 RTT_2 发送的数据包总数 ΔS 与 RTT_3 即将接收的 ACK 总数 $\Delta S \times \frac{R}{R_S}$ 之差，其中 R 是 ACK 数据包接收速率，R_S 是普通数据包发送速率。

Table 8　Description of Each Phases of RTT in ACC

表 8　ACC 各 RTT 阶段的说明

RTT 阶段	数据发送速率	数据包发送总量	ACK 数据包接收速率	ACK 数据包接收总量
RTT_1	R_S	ΔS	Null	Null
RTT_2	R_S	ΔS	R	ΔA
RTT_3（开始暂停）	Null	Null	R	$\Delta S \times \frac{R}{R_S}$
RTT_n（结束暂停）	R	ΔA	Null	Null

注：Null 表示无效。

对于启动速率，ACC 认为 ACK 数据包的到达速率在一定程度上可以反映接收方的接收速率，因此将暂停后的发送速率设置成暂停期间的 ACK 数据包到达速率。

2）对受害流采取自适应速率调整措施。ACC 认为带 UE 标记的 ACK 数据包的持续时间可以反映拥塞程度。若拥塞不严重且不扩散，则 ACC 发送方将很少收到带 UE 标记的 ACK 数据包；若拥塞严重导致扩散，ACC 发送方将在多个周期内都接收到带 UE 标记的 ACK 数据包。因此，ACC 规定，若接收带 UE 标记的 ACK 数据包的持续周期数大于阈值 P_{thre}，则将受害流的发送速率减半，否则不对其进行调整。

3）速率较小的未拥塞流应快速提高发送速率，速率较大的未拥塞流应缓慢提高发送速率。ACC 通过式（7）来对未拥塞流的速率进行调整，其中 *lineRate* 是指网卡的线速率，R_S 是指当前的发送速率，RI 是单周期内的最大增加量。在前 2 个周期内 ACC 将 RI 设置为 RI_{low}，之后将其设置为 RI_{high}，从而实现从温和到激进的增长策略。

$$R = R + \frac{lineRate - R_S}{lineRate} \times RI \text{。} \tag{7}$$

由于能够准确识别拥塞流并对不同类型的流量施加不同的调整策略，ACC 极大地减少了 PFC 暂停的数量，抑制了拥塞扩散，使得速率收敛速度和稳定性得到了一定的提高。ACC 算法在包含了 320 台服务器、20 台聚合交换机和 16 台核心交换机的 fat-tree 拓扑中进行了验证，结果表明，与 HPCC 相比，ACC 降低了 29% 的小流量平均 FCT 以及降低了 40% 的 99 百分位 FCT；与 DCQCN 和 TIMELY 相比能够降低 79.2% 和 96.7% 的死锁发生概率。

基于速率的拥塞调整算法可以由软件步调器或硬件步调器来实现，该方法具有以下优势：1）相比于

基于窗口的策略,控制速率能够有效地避免数据突发,对数据包传输的控制更加精确[8,65];2)基于速率的方法比基于窗口的方法更加容易实现[8]。然而也有以下缺点:1)若反馈信号延迟,单纯地利用速率进行拥塞控制会导致控制环路期间发送过多的数据包,从而对链路造成负担[9,11];2)对数据包进行速率控制可能会造成主机内拥塞[29];3)相比于窗口方案,基于速率的调整需要消耗更多的资源[84]。

3.2 基于发送窗口的拥塞调整

Swift[50]通过与目标延迟比较来调整窗口,目标延迟大小随着拓扑的变化和竞争流数量 N 的变化而扩展。与谷歌公司早期的工作相比,Swift 不仅考虑网络的拥塞,还关注到了主机拥塞的影响。由于是基于绝对目标延迟进行拥塞调整,因此 Swift 在降低平均延迟和尾部延迟方面表现出色,算法稳定性也有所提高。

Swift 利用网络拥塞窗口(fabric congestion window, FCWND)来控制网络拥塞,利用终端拥塞窗口(end-point congestion window, ECWND)来控制主机内(包括发送方和接收方)的拥塞,二者利用同一个算法对窗口进行调整,前者的算法输入是网络延迟,后者的算法输入是端点延迟,实际发送窗口取二者较小值。

Swift 认为,尽管可以设置一个足够大的目标延迟以保证全网拓扑中的所有流量达到较高的吞吐率,但是这样“不负责任”的做法会导致路径较短的流量形成更大的队列。因此,Swift 根据拓扑和负载对目标延迟进行扩展,具体做法是:1)根据拓扑扩展。数据包头中的 TTL(time-to-live)字段反映了流量的路径长短,可以将路径的长短转化为目标延迟的大小。2)根据负载扩展。Swift 通过模拟实验获得竞争流量数量 N 与使瓶颈链路达到饱和所需的目标延迟之间的关系,结果表明,目标延迟的变化与 $\sqrt{N}$ 成正比。

Swift 用于调整窗口的算法如下:若实际延迟高于目标延迟,则利用 MD 方法减小窗口,考虑到一个 RTT 周期内不能对同一个拥塞事件做出多次反应,该算法通过比较 RTT 与当前时间和上一次窗口减小时间的差值来决定是否执行 MD。Swift 利用式(8)对拥塞窗口($cwnd$)进行调整。其中 β 是 MD 因子,max_mdf 是最大 MD 因子。

$$cwnd = \max\left\{1-\frac{\beta\times(delay-target_delay)}{delay}, 1-max_mdf\right\}\times cwnd \text{。} \quad (8)$$

若实际延迟低于目标延迟,则进入 AI 阶段。Swift 允许在一个 RTT 周期内对多个 ACK 做出反应,若 $cwnd>1$,增量大小为 $ai/cwnd$;若 $cwnd<1$,增量 $ai/cwnd$ 被修改为 ai,用于快速恢复。若更新后的 $cwnd'<1$,Swift 将根据 RTT 值与窗口值计算数据包步进延迟,并利用速率控制组件在相邻数据包之间插入步进延迟来控制发送;若 $cwnd'>1$,步进延迟为 0,此时数据包的发送仅由窗口控制。如式(9)所示:

$$cwnd' = cwnd + \frac{ai}{cwnd}\times num_acked \text{。} \quad (9)$$

变量 num_acked 表示接收方的延迟 ACK 数据包对 num_acked 个数据包进行反馈。

Swift 在 60 台配备了 50 Gbps 网卡的主机组成的小型测试平台和 500 台配备了 100 Gbps 网卡的主机组成的大型测试平台中进行了实验,结果显示,Swift 能够维持小于 50 μs 的 99.9 百分位 RTT,比 DCTCP 低 400%;在 incast 场景下实现了线速吞吐量、低延迟和几乎零丢包,其公平性表现也要优于 DCTCP。

LDCP[13]的窗口调整算法 PAWA(per-ACK window adjustment)对每一个 ACK 数据包做出反应,其关键见解是利用 ACK 数据包向发送方授予或撤销“信用”,模仿接收方驱动的拥塞控制。对应于 2 种不同的 ACK 数据包生成方式,其拥塞调整部分也可以被划分为 2 种不同的算法。

第 1 种调整算法对应于 LDCP 的 per-packet ACK 策略,即如果发送方接收到 ECE 字段为 0 的 ACK 数据包,它将判定此时网络状态良好,并以 $\alpha/cwnd$ 为单位增大发送窗口;如果收到 ECE 字段为 1 的 ACK 数据包,它将判定此时网络中存在拥塞,将以 β 为单位减小窗口。α 和 β 都是 0~1 的参数。

第 2 种调整算法对应于 LDCP 的 delay ACK 策略,即此时的调整算法在上述基础上将增量或减量扩大了 n 倍,n 表示接收方的延迟 ACK 数据包对 n 个数据包进行集中反馈。

考虑到突发流量导致的极度拥塞使得发送窗口小于 1 的情况:在 ECE 字段为 0 时,LDCP 将增量设置为 γ,γ 表示最小窗口大小,通常为 1/4, 1/8, 1/16 等;在 ECE 字段为 1 时,窗口取 γ 与 $eta\times cwnd$ 之间的较大值,eta 通常设置为 1/2。

在 32 个节点的 2 层 Clos 网络中,LDCP 的快速启动算法能够在不引起网络拥塞的情况下最大限度地利用空闲带宽,稳定阶段算法能够维持稳定的低队列大小,整体性能表现优于 DCQCN。

RCC 算法[22]利用接收方驱动的显式窗口分配机制缓解最后 1 跳拥塞,且利用 PID 窗口控制机制缓解

网内拥塞。对于前者，RCC 接收方根据数据包的基本传输报文头（base transport header, BTH）统计连接的数量 N，并将每个连接的窗口大小设置为最后 1 跳带宽 C 与连接数量 N 之比；对于后者，RCC 接收方利用 PID 控制器根据估计的目标 STT（*targetSTT*）调整窗口。

如图 21 所示，PID 控制过程可以分为 2 个阶段，分别是计算控制因子 u 和更新发送窗口 *cwnd*。在第 1 阶段，误差 e 定义为测量得到的 STT 与 *targetSTT* 之差，该值同时作为比例项参与控制因子的计算，误差的导数 $\mathrm{d}e/\mathrm{d}t$ 则作为微分项参与计算，积分项为 0；第 2 阶段则利用 u 值和 tanh 函数对发送窗口进行调整。

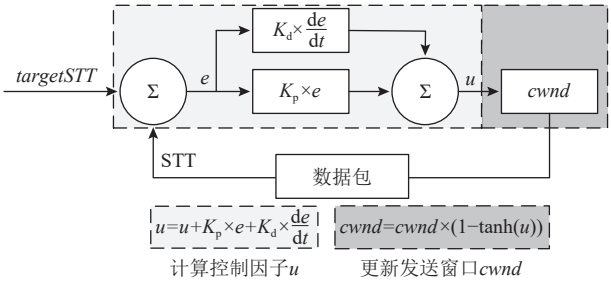


Fig. 21 Window PID control process of RCC

图 21 RCC 的窗口 PID 控制过程

得益于 RCC 能够对最后 1 跳拥塞做出快速反应，同时对网内拥塞采取了参考比例项和微分项的 PID 控制机制，该算法能够实现快速收敛和较好的稳定性，有利于减小延迟和流量完成时间。

在 320 台主机、20 台 ToR 交换机、20 台聚合交换机和 16 台核心交换机构成的 fat-tree 拓扑中，RCC 算法在网络搜索、数据挖掘、Web 服务器和缓存跟随者 4 种流量负载下与 HPCC, DCQCN, TIMELY 等算法进行了仿真比较，结果表明，RCC 在网络搜索工作负载下的平均 FCT 比上述 3 个算法分别低 9%, 30%, 45%，在数据挖掘工作负载下的平均 FCT 则低了 7%, 18%, 19%，在其余 2 类流量模式下的性能表现也整体优于 HPCC 等 4 个算法。

RET 算法[21]借鉴了 TIMELY 的延迟梯度法和 Swift 的目标延迟法。通过对比延迟梯度和目标延迟的大小，RET 不仅能够克服 TIMELY 不公平等缺点，还能在拥塞发生前做出反应。与 TIMELY 和 Swift 一样，RET 同样利用 AIMD 对窗口进行调整。

表 9 对 RET 拥塞调整算法涉及的变量进行了解释，图 22 表示 RET 的窗口调整过程。在一个包含了 288 台主机、12 台 Leaf 交换机和 6 台 Spine 交换机的 Leaf-Spine 拓扑结构中，RET 在不到 2 ms 的时间内快

Table 9 Explanation of Variables and Parameters in RET Algorithm

表 9 RET 算法的变量和参数说明

分类	具体名称	含义
变量	*RttDiff*	加权后的当前延迟梯度
	Delay	当前延迟
	DelayPrev	上阶段延迟
	RttDiffPrev	上阶段加权后的延迟梯度
	QDelay	排队延迟
	QTarget	目标延迟
	NumberAcked	延迟 ACK 响应的数据包数量
	cwnd	窗口大小
参数	*BaseRTT*	基本 RTT
	α	加权延迟梯度中新旧延迟值的权重
	β_g	延迟梯度 MD 因子
	β_r	目标延迟 MD 因子
	Mdf_t	最大 MD 因子
	δ_r	加性增量

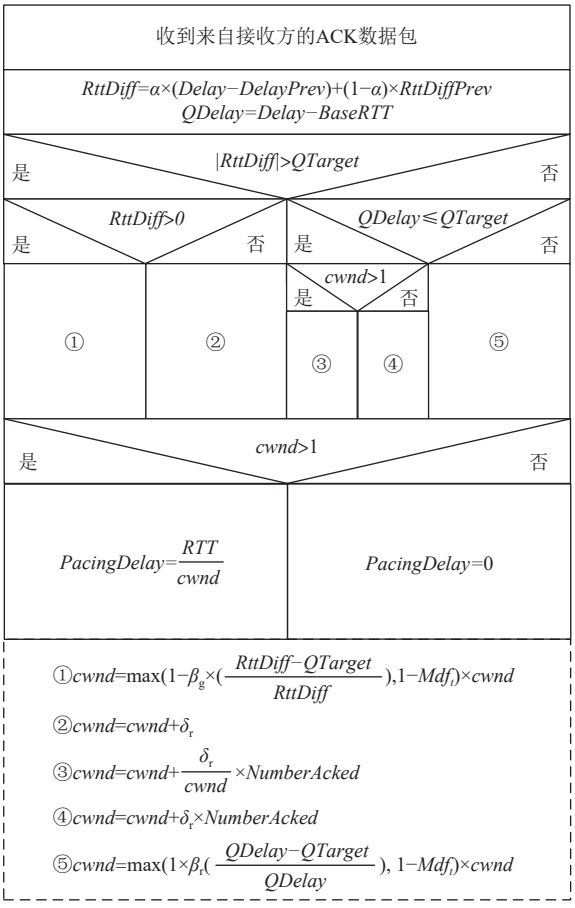


Fig. 22 Window adjustment process for RET

图 22 RET 的窗口调整过程

速稳定了公平速率，显示出良好的公平性；在不同的流量竞争情况下，RET 能够保持比 RSWIFT（Swift 算

法的变体)短 300% 的队列长度; 在 500 个并发流到达时, RET 能够在不到 2 ms 的时间内排出约 4 MB 的积压数据, 而仅依赖目标调整的 RSWIFT 需要的时间是 RET 的 2 倍; 在其他工作负载下, RET 在 FCT 方面的表现均要优于 RSWIFT 和 DCQCN。

Bolt 算法[67]在主机侧分别对 SRC 数据包和 ACK 数据包做出不同反应。对 SRC 数据包, Bolt 认为每窗口一次(once-per-window)的调整策略显得比较笨拙, 容易造成利用率不足和利用率振荡。于是, Bolt 将流量窗口的最小可调整间隔设置为 SRC 数据包控制环路延迟 rtt_{src} 与目标队列大小 $target_{\text{q}}$ 之比, 其中 rtt_{src} 为当前时间 *now* 与数据包出口时间戳标记 *t_data_tx* 的差值, $target_{\text{q}}$ 利用流速率(*flow.rate*), SRC 携带的瓶颈链路速率标记(*link_rate*)和队列大小(*q_size*)由式(10)计算得到。若相邻 SRC 数据包间隔大于该比值, Bolt 将对窗口进行递减。

$$target_{\text{q}} = \frac{flow.rate}{link_rate} \times q_size 。 \tag{10}$$

响应 ACK 数据包的方式则相对简单, 当 ACK 数据包携带的 *inc* 标记为 1 时, 说明链路中存在可用带宽, 此时 Bolt 对窗口进行递增; 否则保持窗口大小不变。通过细粒度拥塞信号、短控制环路延迟以及快速占领可用带宽 3 个机制, Bolt 能够更加迅速地做出拥塞调整, 加快流速率收敛和缩短流量完成时间。

微基准测试实验结果表明, Bolt 的 SRC 机制能够在不到 2 个 RTT 的时间内完全排空拥塞队列; PRU 机制能够在流完成时立即捕获可用带宽, 几乎没有排队或欠利用现象; SM 机制能够在链路故障或路由变更等不可预测事件后快速恢复利用带宽。

基于窗口的拥塞控制能够保证进入网络的数据包数量是固定的, 提高了对反馈延迟的容忍度[9,22], 其实施难度较低, 但是无法避免数据包突发[84]。

3.3 基于速率和窗口相结合的拥塞调整

RoGUE 算法[20]借鉴 TCP Vegas 算法来对窗口进行调整。具体而言, 每当完成一个数据段传输并获得相应的 RTT 值 *curr_rtt* 后, RoGUE 将结合基本 RTT 值 *base_rtt* 计算实际窗口与预期窗口之差 *diff*, 如式(11)所示。若 $diff>\beta$, 则对窗口进行加法减小; 若 $diff<\alpha$, 则对窗口进行加法增大。

$$diff = cwnd \times \frac{curr_rtt - base_rtt}{curr_rtt} 。 \tag{11}$$

为了避免单纯依靠窗口所带来的突发, RoGUE 还利用硬件速率限制器对发送速率进行控制, 发送速率 *rate_limit* 为 *cwnd* 与 *base_rtt* 之比。

RoGUE 旨在为不需要 PFC 机制的 RoCE 网络提供有效的拥塞控制和恢复机制。实验结果表明, RoGUE 在丢包率为 0.025% 时性能良好, 能够处理几乎所有的丢包情况; 在没有 PFC 机制的情况下, RoGUE 的收敛性和公平性优于 DCQCN 和 TIMELY, 平均吞吐量接近公平份额。

HPCC 算法[9]利用较为精确的 INT 信号实现了 MIMD 算法。该算法具有 2 个关键思想: 其一是利用窗口和速率对发送方的行为进行限制以避免突发, 同时避免因反馈延迟导致的网络负载过重; 其二是在 1 个 RTT 周期内仅对窗口首个数据包对应的 ACK 数据包做出反应, 避免反应过度造成带宽浪费。

HPCC 通过窗口来对链路 j 上的 inflight 字节数进行控制, 初始窗口大小 $cwnd_{\text{init}}$ 设置为发送方网卡带宽 B_{NIC} 与 *BaseRTT* 的乘积。HPCC 认为如果能将每条链路的 inflight 字节总数 I_j 保持在略小于该链路带宽 B_j 与 *BaseRTT* 乘积的水平, 则一定不会出现排队与拥塞, 因此 HPCC 把目的 inflight 字节总数设置为 $0.95\times B_j\times BaseRTT$, 窗口的大小以式(12)进行变化, 其中 k_j 是实际 inflight 字节数与目的 inflight 字节数之比, $cwnd_{\text{ai}}$ 是确保公平性而设置的窗口增量。

$$cwnd = \frac{cwnd}{\max\{k_j\}} + cwnd_{\text{ai}} 。 \tag{12}$$

HPCC 认为, 如果对每一个到来的 ACK 都做出响应将导致窗口的过度调整, 原因在于 1 个 RTT 周期内的大部分 ACK 数据包所反映的拥塞情况是相同送窗口 $cwnd^{\text{c}}$ 并“记住”每次窗口调整后发送的第 1 个数据包只有收到该数据包对应的 ACK 数据包时才会更新 $cwnd^{\text{c}}$ 值。此时的发送窗口调整函数修改为式(13)。

$$cwnd = \frac{cwnd^{\text{c}}}{\max\{k_j\}} + cwnd_{\text{ai}} 。 \tag{13}$$

为了避免流量突发, HPCC 在窗口的基础上还利用速率对数据包进行控制, 发送速率设置为 *cwnd* 与 *BaseRTT* 之比。综上所述, HPCC 利用 MIMD 加快了流量的收敛速度, 引入 AI 提高了流量公平性, 同时综合利用了窗口与速率控制提高了稳定性, 通过少量带宽换取零排队实现了超低延迟。

FastTune 算法[66]对窗口的调节在很大程度上参考了 HPCC 的思路, 即通过 *txBytes*, *qLen*, *ts* 等信息计算链路利用率、通过设定预期链路利用率作为参考值、通过参考窗口 $cwnd^{\text{c}}$ 实现 per-RTT 调整、通过将发送窗口转换为发送速率以控制突发等, 本文不再赘述。

POWERTCP 算法[51]的优化目标集中于流完成时间，同时不会影响吞吐量和延迟等基本性能要求。其拥塞调整部分由 3 个函数组成，分别为：1）NEWACK。由 ACK 数据包到达事件触发功率计算和窗口更新函数。2）NORMPOWER。利用 ACK 数据包携带的 INT 信息计算功率 Γ_{smooth} 为更新窗口提供依据。3）UPDATEWINDOW。利用 Γ_{smooth} 和前一阶段的窗口大小 $cwnd_{\text{old}}$ 来调整当前的窗口大小 $cwnd$。

具体来说，NORMPOWER 函数利用 ACK 携带的队列大小 $qsize$、传输字节数 $txBytes$、链路带宽 B 和时间戳 TS，结合事先商定的 $BaseRTT$ 以及历史信息来计算瓶颈链路 BDP、队列大小、传输速率以及队列变化率。其中，链路 BDP 和队列大小之和为电压，传输速率和队列变化率之和为电流，功率等于电压与电流的乘积。

UPDATEWINDOW 函数利用计算出的归一化功率 Γ_{smooth} 和 $cwnd_{\text{old}}$ 来计算新窗口值，如式（14）所示。其中 γ 是 EWMA 参数，β 是加性增量。

$$cwnd=\gamma\times\left(\frac{cwnd_{\text{old}}}{\Gamma_{\text{smooth}}}+\beta\right)+(1-\gamma)\times cwnd\ 。\qquad(14)$$

实验结果表明，POWERTCP 在不同程度网络负载环境下的长流、短流完成时间均要优于 HPCC、DCQCN 和 TIMELY 算法；在 incast 流量模式下，POWERTCP 能够快速减轻拥塞，保持稳定的吞吐量，而 TIMELY 和 HPCC 在拥塞发生后吞吐量会出现不稳定的情况。

基于窗口与速率的调整算法显然同时具备二者的优点，既能避免数据突发，又可以限制在控制环路期间发送的数据包数量，提高网络稳定性。代价是需要更多的硬件组件或逻辑组件，给终端主机的资源造成负担。

4 算法优化及补充

4.1 针对上述部分算法的优化

网络场景多样性和网络内在动态性使得通用拥塞控制算法的设计极为困难[84]，例如大象流和老鼠流共存的混合环境、多对 1 传输的 incast 环境和多对多传输的多瓶颈环境等。另外，性能之间的掣肘使得拥塞控制算法难以实现全面的性能表现，例如吞吐量与延迟、吞吐量与稳定性之间存在着一定的权衡[9]。为了克服或缓解上述问题带来的影响，已有工作对现有算法进行优化，本文将对部分优化工作进行介绍。

微软公司和哥伦比亚大学在 2016 年的 CoNEXT 会议上发表的文章[57]就 DCQCN 和 TIMELY 算法以及它们所代表的 ECN 标记感知方法和延迟梯度感知方法进行了讨论，并提出了 Patched TIMELY 算法来对 TIMELY 公平性不足进行补救。

1）DCQCN 和 TIMELY 的对比。DCQCN 具有唯一实现流量公平性的固定点；若采用推荐的参数值，DCQCN 将以指数级的速度收敛；若控制环路的延迟较低，DCQCN 在固定点附近能保持较好的稳定性，然而其稳定性易受流量数量的影响。相较而言，TIMELY 理论上没有固定点，即 TIMELY 无法实现公平性；队列长度和流量速率不会收敛；系统将支持处于振荡状态。

2）TIMELY 与 Patched TIMELY 的对比。TIMELY 在实际使用中通过数据段突发实现了公平性和收敛性，但其收敛速度仍然不理想。另外，由于数据段突发会引起队列波动，因此其稳定性较差，链路利用率较低。为此，本文提出了 Patched TIMELY 算法用于解决上述问题。如图 23 所示，Patched TIMELY 利用加权函数 w 将 RTT 梯度（$normalized_gradient$）转换为权重（$weight$），结合归一化 RTT 测量值（new_rtt）与 RTT 参考值（rtt_ref）之差（$error$）将速率调整函数修改为基于 $weight$ 和 $error$ 的函数，使得 Patched TIMELY 具有唯一的固定点，并保证其收敛速度和稳定性。

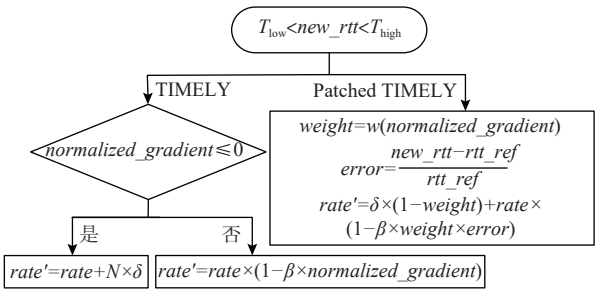


Fig. 23 Comparison of adjustment strategies for TIMELY and Patched TIMELY

图 23 TIMELY 和 Patched TIMELY 的调整策略对比

3）ECN 与延迟的对比。出端口的 ECN 标记方法将排队延迟和反馈延迟分离开来，而延迟却无法将二者解耦，因此随着队列增大，延迟携带的拥塞信息滞后；此外，基于延迟的方案难以同时实现流量公平性和固定队列大小；最后，ECN 携带的拥塞信号只会随着反向链路的拥塞而被推迟，而延迟携带的拥塞信号将会被反向链路的拥塞污染，因此 ECN 对反向链路拥塞的容忍度更高。综上所述，本文认为基于 ECN 的方案保持的队列长度更短且更稳定，有利于减小 FCT。

西安交通大学和清华大学 2018 年在 TON 上提

出的 CEDM 标记方法[72]针对突发流量对 ECN 进行了适当优化：在数据包进入出口队列和离开出口队列时通过瞬时队列长度 *queue_length*、缓冲区阈值 *K*1 和 *K*2 以及队列变化率 *avgs* 对数据包进行标记，引入 2 个阈值以及队列变化率的主要目的是排除突发流量引起的短暂队列变化对标记产生影响，避免拥塞反应过度以及队列长度振荡。

根据 IP 报文头中的 ECN 字段，CEDM 的数据包可以划分为 4 种类型，如表 10 所示，入队前和出队后的数据包要么是未标记数据包，要么是拥塞数据包，而未确定数据包只会在排队时出现。

Table 10 codepoints Defined by CDEM and Their Meanings

表 10 CDEM 定义的 codepoints 及其含义

ECN 字段		含义
ECT 位	CE 位	
0	0	不具备 ECN 传输功能
0	1	未确定数据包
1	0	未标记数据包
1	1	经历拥塞

图 24 展示了 CEDM 在出口队列处入队和出队的标记逻辑：①入队长度 *enqueue_length*>*K*1，队列斜率 *avgs*>0，则将未标记状态修改为未确定；②出队长度 *dequeue_length*>*K*1，*avgs*>0，则将未标记状态修改为拥塞状态；③*enqueue_length*>*K*2，则将未标记状态修改为拥塞状态；④*dequeue_length*>*K*2，则将未标记/未确定状态修改为拥塞状态。除上述 4 种情况外均不会修改数据包字段。CEDM 还对 *K*1 和 *K*2 的值进行了细致的推理，使队列具备防止缓冲区溢出以及容纳突发流量的能力。

香港科技大学和微软公司在 2019 年的 CoNEXT

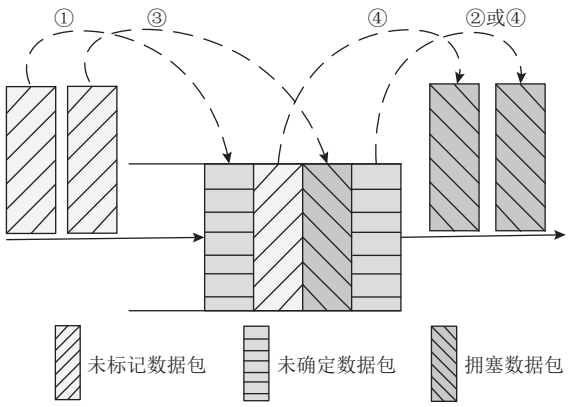


Fig. 24 Labeling strategies for CEDM

图 24 CEDM 的标记策略

会议上提出的 ECN#标记方法[36]认为在计算 ECN 标记阈值时应考虑 RTT 的变化，否则会对吞吐量和延迟的权衡产生不良影响。ECN#做出该判断的前提是将网络组件对数据包的处理延迟包含在 RTT 内（类似于 Swift 的做法），通过实验 ECN#证明了上层网络组件的数量、类型和负载程度会导致 RTT 成倍数变化，因此有必要在确定标记阈值时将 RTT 变化考虑在内。

考虑到实施难度，ECN#并未实现动态标记阈值，而是利用较高百分位的 RTT 来计算阈值以保证较高吞吐量和突发容忍（ECN 标记阈值 K 的计算公式：$K=\lambda\times C\times BaseRTT$，其中 λ 是 0~1 的参数，C 是瓶颈链路带宽，*BaseRTT* 是基本往返时间），同时在长期队列堆积时进一步标记数据包以保证较低延迟。在具体实现中，ECN#将队列长度转换为排队时间 *sojourn_time* 参与运算。

ECN#标记算法可总结为 3 个函数，分别是：瞬时标记函数；持久标记函数；持久队列堆积检测函数。当数据包离队时，首先进入瞬时标记函数，若数据包排队时间（*sojourn_time*）大于瞬时阈值（*ins_target*），则进行标记；否则进入持久标记函数，首先调用检测函数来判断当前队列是否处于持久堆积状态，根据其返回结果 *detected* 以及其他参数来保证一个持久标记间隔（*pst_interval*）内仅标记 1 个数据包，其中持久队列堆积检测函数认为只有当 *sojourn_time* 在大于持久阈值（*pst_target*）的状态保持 *pst_interval* 的时间才可将队列视为持久堆积状态。

谷歌公司和米兰大学 2023 年提出的 Poseidon 算法[41]底层拥塞感知机制是 INT 机制，用于指导拥塞控制的信号是由 INT 机制采集的最大每跳延迟（maximum per-hop delay, MPD）。其原理可以概括为：利用交换机的 INT 技术，Poseidon 能够获取网络中瓶颈跳的延迟（MPD），将 MPD 与基于流速率调整的最大每跳延迟目标（maximum per-hop delay target, MPT）进行比较，以自适应的方式决定增加或减少拥塞窗口大小。

HPCC, POWERTCP, Bolt 在各节点采集的大量元数据随着节点累加，Poseidon 则仅采集每跳的延迟信息并实时替换，最终保留 MPD。其优势在于数据头空间大大减小，减轻了交换机和网络的负担；同时由于将拥塞信息准确定位到瓶颈跳，有利于实现数据流的最大最小公平性。

Poseidon 作为谷歌 TIMELY 和 Swift 的后继方案，其底层拥塞感知机制由早期算法的时间戳标记更改为更加先进的 INT 标记，有能力直接采集数据包在

某节点的排队时间，端主机无需对时间信息进行复杂运算；三者相同之处为均使用延迟作为拥塞感知信号，TIMELY 和 Swift 基于端到端延迟测量，拥塞信号包含了所有跳的拥塞信息，Poseidon 则基于瓶颈跳的排队时间，端主机仅响应瓶颈跳拥塞。

哈佛大学、南加州大学和伦敦玛丽大学在 2020 年的 SIGCOMM 会议上提出的 PINT 算法[75] 旨在限制每个数据包中添加的信息量的同时，提供与 INT 类似的网络可见性。其基本原理为：1）通过在多个数据包上编码请求的数据，允许每个数据包的开销限制在用户设定的范围内，最低可至每个数据包 1 位；2）使用概率编码技术，通过近似来满足用户设定的开销预算，对于上层用户而言，遥测数据的近似值已足够。

PINT 在拥塞控制领域的应用具有以下优点：1）PINT 不仅将收集统计数据相关的开销降至最低，而且能够获取近似甚至更好的性能；2）由于降低了开销，PINT 机制有助于提高算法的可扩展性；3）由于 PINT 允许网络在不同的网络负载和拓扑变化下灵活地收集遥测数据，使得拥塞控制算法能够更好地适应网络条件的变化。

然而，由于本身的概率特性，PINT 也存在以下缺陷：1）概率编码不确定。基于 PINT 的拥塞控制机制意味着不是每个数据包都会携带完整的遥测信息，可能导致在某些情况下无法获得足够的信息来准确判断网络状态。2）对短流不友好。对于以短流为主的网络环境，PINT 可能无法有效地收集足够的遥测数据，因为它们没有足够的数据包来分散编码遥测信息。3）实际部署较为复杂。PINT 需要核心交换设备的支持，而当前支持 PINT 的交换机型号有限，在一定程度上不利于 PINT 机制在拥塞控制领域的大规模应用。

4.2 基于强化学习的拥塞控制方法

基于规则的拥塞控制算法在特定环境、特定任务下有助于实现良好性能，但在环境或任务发生变化时需要网络管理员花费数周的时间重新设置规则与参数[84-86]。鉴于机器学习（machine learning, ML）较高的灵活性和适应性，网络领域正探讨将 ML 应用于拥塞控制，即基于学习的拥塞控制。

文献 [84] 将基于学习的拥塞控制算法分为面向性能的算法和数据驱动的算法，前者通常需要提前设定效用函数并手动修改其中的参数，后者则依赖于数据集进行实时学习。数据驱动的算法根据其采用的技术可以划分为监督学习、无监督学习和强化学习（reinforcement learning, RL）三类，该文指出基于学习的拥塞控制大部分研究集中在强化学习这一类。然而，由于数据中心网络通常是单一管理阈下的有线网络环境，拓扑结构与流量状态相对稳定，因此相较于互联网等高带宽延迟积的网络，面向数据中心的强化学习拥塞控制研究并不是主流方向[84]。

强化学习本质上是一类闭环问题[87]，强化学习系统由代理与环境组成，其中代理是做出行为决策的实体，环境是向代理反馈行为代价的实体。代理实体的策略函数用于指导其具体行动，价值函数用于定义强化学习的目标；环境实体的奖励函数用于反馈每次行动的收益，观测函数向代理反馈环境的变化信息。其具体过程可概括为：代理在状态 S_t 下执行行动 A_t，环境受到 A_t 的影响并向代理反馈一个奖励信号 R_t 以及观测信号 O_t，代理利用 R_t 和 O_t 决定下一阶段的状态 S_{t+1} 以及所要做出的行动 A_{t+1}，在此过程中代理要保证价值函数达到最大。该过程可以被建模为马尔可夫决策过程（Markov decision processes, MDP）[87-88]。表 11 所示为强化学习问题与拥塞控制问题的实体与函数对应关系。

Table 11 Connection of RL with Congestion Control

表 11 强化学习与拥塞控制的联系

强化学习的实体与函数	拥塞控制的实体与函数
代理	发送方
环境	交换机或接收方
策略函数	拥塞调整函数
价值函数	网络性能整体表现
奖励函数	网络性能变化
观测函数	拥塞感知函数

以色列理工学院、英伟达研究所和巴伊兰大学在 2021 年提出的 PCC-RL 算法[88] 指出基于强化学习的拥塞控制算法在实际应用中面临多代理（即多个发送方的行动可能互相影响）、部分可观测（即并不是所有的环境变量都可以反馈）和多目标（即多个网络性能目标导致价值函数相对复杂）三个问题，因此 MDP 并不完全适用于拥塞控制机制。为此，ADPG 将拥塞控制视为部分可观测的多代理多目标强化学习任务，并建模为部分可观测马尔可夫过程（partially observable Markov decision process, POMDP）。

PCC-RL 规定每个代理只观测与自身相关的部分环境变量而不是全局信息；每个代理拥有一个价值函数；所有代理共享相同的策略函数。在上述规定的制约下，PCC-RL 创造性地引入了策略梯度方案，

通过精心设计奖励函数的结构来推测能够获取最大奖励的策略。同年,该团队在 CCGrid 会议上发表的文章[85]讨论了在网卡上部署基于强化学习的拥塞控制算法面对的计算能力有限问题,通过将神经网络架构精简为决策树大大降低了推理时间,并对推理得到的策略的质量影响微乎其微。鉴于奖励函数的设计与推理以及精简神经网络的内容并不是本文的重点,对此不再赘述。

5 总结与展望

近年来,大模型的迅速发展对计算、网络与存储构成的分布式系统提出了更高要求。为了匹配计算与存储的指数增长趋势,数据中心网络必须为分布式系统间通信提供高带宽、低延迟、零丢包与高可靠等特性。作为一种高性能网络实现技术,RDMA 在数据中心的部署逐渐得到了更广泛的关注。

设计良好的拥塞控制机制是维持 RDMA 性能优势的重要手段之一。本文通过将拥塞控制划分为感知与拥塞调整对国内外的研究现状进行了详述,并对不同感知方法和调整方法的优劣进行总结,最后阐述了算法优化和基于学习的拥塞控制工作。

根据发送速率或窗口的计算实体进行算法划分是比较常见的划分方法。在发送方驱动的算法中,发送方接收来自交换机或接收方的拥塞信号,启发式地对速率或窗口进行调整,上文提到的 DCQCN, TIMELY, RoGUE, HPCC, Swift, FastTune, LDCP, PowerTCP, RET, Bolt, ACC 属于该范畴。交换机驱动的算法充分利用了现代交换机的计算能力和可编程能力,通过直接向数据源返回更新后的速率或窗口有效地缩短了控制环长度,但是难以大规模部署,RoCC, HierCC, PACC 算法属于该范畴。接收方驱动的算法相较于发送方能够更早地利用拥塞信息进行速率或窗口计算,同时可以将接收方的接收能力考虑在内,PCN, RCC 算法属于该范畴。

数据中心 RDMA 拥塞控制已获得 10 余年的关注与发展。结合当前研究成果,该领域面临的挑战可以系统地总结为“快、准、智、广、全”。“快”表现为难以快速识别并响应拥塞;“准”则是指难以精准识别拥塞流与受害流,难以精确调整发送速率或窗口;“智”揭示了人工智能方法当前难以在拥塞控制领域展开广泛应用;“广”体现在难以大规模部署新算法;“全”意味着难以兼顾网络拥塞与主机拥塞,难以兼顾各种流量模式。对其详细阐述如下:

1)第 1 个 RTT 内感知拥塞。作为支撑各种上层应用的基础设施,数据中心存在形式丰富的流量模式,其中老鼠流突发造成的首个 RTT 内拥塞是一个亟待解决的问题。端到端拥塞控制机制需要大于一个 RTT 的时间才能将拥塞感知信号或拥塞调整信号返回到发送方,因此在发送方对拥塞做出反应之前老鼠流就已停止,即端到端拥塞控制算法无法解决老鼠流突发造成的拥塞问题。当前的解决方案通常是利用交换机的可编程能力,由交换机返回拥塞信号,使发送方在第 1 个 RTT 内对网络拥塞做出反应。

2)拥塞流的精准识别与控制。交换机共享缓冲区的有限队列划分及排队机制,以及 PFC 机制在队列粒度上的识别共同造成了受害流问题。当受害流与拥塞流在同一出端口的相同队列中排队且超过标记阈值时,受害流将被无辜地标记,最终导致其发送速率或窗口被减小;若下游触发了 PFC 机制,受害流甚至会被暂停。解决该问题的根本途径是精准识别并控制拥塞流,这通常需要自定义交换机的标记策略以及终端的识别策略。

3)速率或窗口的精确调整。拥塞信号的丰富度一定程度上限制了拥塞调整的准确性。纵观网络设备及算法的发展历史,早期的 TCP Reno 算法以丢包为拥塞信号,以“慢启动、拥塞避免、快速重传和快速恢复”为拥塞调整手段,即丢包携带的二进制信息决定了拥塞调整不得不基于一定的经验法则;当前较为先进的算法则基于交换机的 INT 功能,或是基于绝对延迟和延迟梯度的 PID 控制器,有望实现在少量迭代次数后将速率或窗口调整到精确值。

4)人工智能具有通过自适应学习和决策优化网络传输性能的优势,在网络拥塞控制领域的应用越来越广泛,然而也面临着诸多挑战。首先,模型的训练与推理通常需要大量的计算资源,这要求充足的硬件支持,如高速处理器和专用加速器,使得相关算法在网卡上的部署变得困难;其次,模型在网络环境快速变化时可能会出现灾难性遗忘,无法有效适应新的任务;此外,模型的可解释性也是一大挑战,因为黑盒模型难以诊断和修正潜在的决策偏差。为了应对上述挑战,首先可以通过模型蒸馏和网络结构优化减少对硬件资源的需求,以降低部署成本;其次可采用多任务学习和迁移学习等技术,提高模型对新环境的适应能力,减轻灾难性遗忘的影响;最后,结合可解释性技术,增强模型的透明度和可控性,以便更好地理解和改进拥塞控制策略。这些方法为人工智能在拥塞控制领域的应用提供了新的解决思路。

5)新算法的大规模部署。定制网卡与定制交换机能否批量生产决定了算法能否在短期内大规模部署。学术界的许多先进算法通常需要异构硬件的支持，部署面临困难；工业界则更加注重算法的落地应用，因此微软、谷歌、阿里巴巴等大型科技公司在面对该挑战时具有更大的优势。

6)网络拥塞与主机拥塞。网络拥塞是指网络设备存储转发能力有限导致的数据包在网络中大量积聚并扩散的情况。主机拥塞是指端主机存储处理能力有限导致数据包在主机排队甚至反压到网络中的情况。前者通常是当前拥塞控制方案的主要设计目标，后者在某种意义上可类比为流量控制，近年来许多算法已尝试解决该问题。上述 2 种类型拥塞需要软硬件协同设计解决，通常来说底层硬件提供快速缓解网络拥塞的处理逻辑，上层软件则需要为缓解主机拥塞提供评估依据。

7)面向多流量模式的通用算法设计。数据中心流量在时间和空间尺度表现出了一定程度的动态特征，例如在时间上表现为不可预测性以及在空间上表现为局部聚集性。一些拥塞控制算法在特定流量模式下表现出色，然而当流量模式改变后其性能也将随之下降。如何利用有限的资源，通过算法设计来实现一个较为通用的拥塞控制算法是一项兼具挑战与价值的工作。

正如 RFC 1925 所说："It is always something. Good, Fast, Cheap: Pick any two (you can't have all three)[89]"。在网络领域，"Good"通常指高质量、高性能和高可靠的网络，"Fast"则是指快速的响应时间或完成时间，"Cheap"意味着较高的成本经济性，这三者通常无法兼得。实现"快、准、智、全"的根本目的是提高底层网络的性能与可靠性，加速数据流完成时间，提高上层应用的表现，即实现"Good"与"Fast"，这通常需要增加网卡或交换机设计的复杂度。然而，对硬件的修改不可避免会引入更高的成本，不利于算法的大规模部署，即无法满足"广"的需求，实现"Cheap"目标。实际上，拥塞控制算法的设计需要综合考虑网络拓扑、流量模式和应用需求等多方面因素，并对设计目标进行优先级排序，以达到性能与成本以及性能的不同方面之间的权衡。

随着人工智能技术与应用的快速演进，RDMA 拥塞控制作为更高带宽、更低延迟数据传输的基础保障技术，其创新发展与迭代部署势在必行。本文预测，学术界作为开展创新研究最活跃的群体，利用可编程网络设备解决快速拥塞感知、拥塞流的精确识别以及准确的拥塞调整等难点问题将是未来的主流研究方向；以微软、谷歌、英伟达和阿里巴巴等大型科技企业为代表的工业界则会更加关注算法创新与商品设备成本之间的权衡，利用其规模优势推动形成新的事实标准。

作者贡献声明：张毓涛和杨惠负责相关工作调研和整理、论文架构设计、算法的仿真测试以及论文的撰写和修改；李韬指出了该领域的未来发展方向，并对论文进行了修改；黄曼蒂参与了论文架构设计与修改；李天云补充了基于强化学习的拥塞控制方法；孙志刚负责论文审阅并提出指导意见。张毓涛和杨惠为共同第一作者。

参 考 文 献

[1] China Communications Services Digital Infrastructure Industry Research Institute. White paper on the development of China's data center industry[EB/OL]. [2024-03-24]. https://aimg8.dlssyht.cn/u/551001/ueditor/file/276/551001/1684888884683143.pdf (in Chinese)
(中国通服数字基建产业研究院. 中国数据中心产业发展白皮书[EB/OL]. [2024-03-24]. https: //aimg8.dlssyht.cn/u/551001/ueditor/file/276/551001/1684888884683143.pdf)

[2] Guo Chuanxiong, Wu Haitao, Deng Zhong, et al. RDMA over commodity ethernet at scale[C]//Proc of the 2016 ACM SIGCOMM Conf. New York: ACM, 2016: 202−215

[3] Yan Dingyu, Liu Yaping, Zhang Shuo, at al. PCNP: A RoCEv2 congestion control using precise CNP[J]. Computer Nestworks, 2024, 247(C): 110453

[4] Simons J. RDMA in the cloud: Enabling high-bandwidth, low-latency communication in virtual environments for HPC[EB/OL]. [2024-03-24]. https://octo.vmware.com/wp-content/uploads/2014/11/ndm2014-sc14-simons3.pdf

[5] Hu Jinbin, Shen Houqiang, Liu Xuchong, et al. RDMA transports in datacenter networks: Survey[J]. IEEE Network, 2024, 38(6): 380−387

[6] Mittal R, Shpiner A, Panda A, et al. Revisiting network support for RDMA[C]//Proc of the 2018 Conf of the ACM Special Interest Group on Data Communication. New York: ACM, 2018: 313−326

[7] Wang Zilong, Luo Layong, Ning Qingsong, et al. SRNIC: A scalable architecture for RDMA NICs[C]//Proc of the 20th USENIX Symp on Networked Systems Design and Implementation (NSDI'23). Berkeley, CA: USENIX Association, 2023: 1−14

[8] Zhu Yibo, Eran H, Firestone D, et al. Congestion control for large-scale RDMA deployments[C]//Proc of the 2015 ACM Conf on Special Interest Group on Data Communication. New York: ACM, 2015: 523−536

[9] Li Yuliang, Miao Rui, Liu Hongqiang, et al. HPCC: High precision congestion control[C]//Proc of the 2019 ACM Special Interest Group on Data Communication. New York: ACM, 2019: 44−58

[10] Cheng Wenxue, Qian Kun, Jiang Wanchun, et al. Re-architecting congestion management in lossless Ethernet[C]//Proc of the 17th USENIX Symp on Networked Systems Design and Implementation (NSDI'20). Berkeley, CA: USENIX Association, 2020: 131−148

[11] Taheri P, Menikkumbura D, Vanini E, et al. RoCC: Robust congestion control for RDMA[C]//Proc of the 16th Int Conf on Emerging Networking Experiments and Technologies. New York: ACM, 2020: 17−30

[12] Zhang Yiran, Liu Yifan, Meng Qingkai, et al. Congestion detection in lossless networks[C]//Proc of the 2021 ACM SIGCOMM 2021 Conf. New York: ACM, 2021: 370−383

[13] Dai Huichen, Fu Binzhang, Tan Kun. LDCP: PFC-free low delay control protocol[EB/OL]. [2024-03-30]. https://datatracker.ietf.org/meeting/108/agenda/tsvwg-drafts.pdf

[14] Zhang Yiran, Meng Qingkai, Hu Chaolei, et al. Revisiting congestion control for lossless Ethernet[C]//Proc of the 21st USENIX Symp on Networked Systems Design and Implementation (NSDI'24). Berkeley, CA: USENIX Association, 2024: 131−148

[15] IEEE 802.1. 802.1Qbb – Priority-based flow control[EB/OL]. [2024-04-10]. https://1.ieee802.org/dcb/802-1qbb/

[16] Alizadeh M, Atikoglu B, Kabbani A, et al. Data center transport mechanisms: Congestion control theory and IEEE standardization[C]//Proc of the 46th Annual Allerton Conf on Communication, Control, and Computing. Piscataway, NJ: IEEE, 2008: 1270−1277

[17] Stephens B, Cox A L, Singla A, et al. Practical DCB for improved data center networks[C]//Proc of the 2014 IEEE Conf on Computer Communications. Piscataway, NJ: IEEE, 2014: 1824−1832

[18] Hu Shuihai, Zhu Yibo, Cheng Peng, et al. Deadlocks in datacenter networks: Why do they form, and how to avoid them[C]//Proc of the 15th ACM Workshop on Hot Topics in Networks. New York: ACM, 2016: 92−98

[19] Shpiner A, Zahavi E, Zdornov V, et al. Unlocking credit loop deadlocks[C]//Proc of the 15th ACM Workshop on Hot Topics in Networks. New York: ACM, 2016: 85−91

[20] Le Yanfang, Stephens B, Singhvi A, et al. RoGUE: RDMA over generic unconverged Ethernet[C]//Proc of the 2018 ACM Symp on Cloud Computing. New York: ACM, 2018: 225−236

[21] Tang Jian, Xu Tingting, Nguyen C, et al. Tuning target delay for RTT-based congestion control[C]//Proc of the 30th IEEE Int Conf on Network Protocols (ICNP). Piscataway, NJ: IEEE, 2022: 1−11

[22] Zhang Jiao, Shi Jiaming, Zhong Xiaolong, et al. Receiver-driven RDMA congestion control by differentiating congestion types in datacenter networks[C]//Proc of the 29th IEEE Int Conf on Network Protocols (ICNP). Piscataway, NJ: IEEE, 2021: 1−12

[23] Huang Shan, Dong Dezun, Bai Wei. Congestion control in high-speed lossless data center networks: A survey[J]. Future Generation Computer Systems, 2018, 89(C): 360−374

[24] Zeng Gaoxiong, Hu Shuihai, Zhang Junxue, et al. Transport protocols for data center networks: A survey[J]. Journal of Computer Research and Development, 2020, 57(1): 74−84 (in Chinese)
(曾高雄, 胡水海, 张骏雪, 等. 数据中心网络传输协议综述[J]. 计算机研究与发展, 2020, 57(1): 74−84)

[25] Du Xinle, Xu Ke, Li Tong, et al. Traffic control for data center network: State of the art and future research[J]. Chinese Journal of Computers, 2021, 44(7): 1287−1309 (in Chinese)
(杜鑫乐, 徐恪, 李彤, 等. 数据中心网络的流量控制: 研究现状与趋势[J]. 计算机学报, 2021, 44(7): 1287−1309)

[26] Yan Dingyu, Liu Yaping, Zhang Shuo, at al. A survey of RoCEv2 congestion control[C]//Proc of the 7th Int Conf on Information Science, Communication and Computing. Berlin: Springer, 2023: 42−56

[27] Alizadeh M, Greenberg A G, Maltz D A, et al. Data center TCP (DCTCP)[C]//Proc of the ACM SIGCOMM 2010 Conf. New York: ACM, 2010: 63−74

[28] Alizadeh M, Yang Shuang, Sharif M, et al. pFabric: Minimal near-optimal datacenter transport[J]. SIGCOMM Computer Communication Review, 2013, 43(4): 435−446

[29] Alizadeh M, Kabbani A, Edsall T, et al. Less is more: Trading a little bandwidth for ultra-low latency in the data center[C]//Proc of the 9th USENIX Conf on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2012: 1−14

[30] Zats D, Das T, Mohan P, et al. DeTail: Reducing the flow completion time tail in datacenter networks[C]//Proc of the ACM SIGCOMM 2012 Conf on Applications, Technologies, Architectures, and Protocols for Computer Communication. New York: ACM, 2012: 139−150

[31] Wilson C, Ballani H, Karagiannis T, et al. Better never than late: Meeting deadlines in datacenter networks[C]//Proc of the ACM SIGCOMM 2011 Conf. New York: ACM, 2011: 50−61

[32] Hong Chiyao, Caesar M, Godfrey P B. Finishing flows quickly with preemptive scheduling[J]. SIGCOMM Computer Communication Review, 2012, 42(4): 127−138

[33] Dean J, Barroso L A. The tail at scale[J] Communications of the ACM, 2013, 56(2): 74−80

[34] Grosvenor M P, Schwarzkopf M, Gog I, et al. Queues don't matter when you can JUMP them![C]//Proc of the 12th USENIX Symp on Networked Systems Design and Implementation (NSDI'15). Berkeley, CA: USENIX Association, 2015: 1−14

[35] Vamanan B, Hasan J, Vijaykumar T N. Deadline-aware datacenter TCP (D2TCP)[C]//Proc of the ACM SIGCOMM 2012 Conf on Applications, Technologies, Architectures, and Protocols for Computer Communication. New York: ACM, 2012: 115−126

[36] Zhang Junxue, Bai Wei, Chen Kai. Enabling ECN for datacenter networks with RTT variations[C]//Proc of the 15th Int Conf on Emerging Networking Experiments and Technologies. New York: ACM, 2019: 233−245

[37] Roy A, Zeng Hongyi, Bagga J, et al. Inside the social network's (datacenter) network[C]//Proc of the 2015 ACM Conf on Special Interest Group on Data Communication. New York: ACM, 2015: 123−137

[38] Evangelinos C, Hill C N. Cloud computing for parallel scientific HPC applications: Feasibility of running coupled atmosphere-ocean climate models on Amazon's EC2[J]. Ratio, 2008, 2(2): 1−6

[39] Dean J, Ghemawat S. MapReduce: Simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107−113

[40] Kohavi R, Longbotham R. Online experiments: Lessons learned[J]. Computer, 2007, 40(9): 103−105

[41] Wang Weitao, Moshref M, Li Y, et al. Poseidon: Efficient, robust, and practical datacenter CC via deployable INT[C]//Proc of the 20th USENIX Symp on Networked Systems Design and Implementation

(NSDI'23). Berkeley, CA: USENIX Association, 2023: 255−274

[42] Dragojevic A, Narayanan D, Hodson O, et al. FaRM: Fast remote memory[C]//Proc of the 11th USENIX Conf on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2014: 401−414

[43] Hwang J, Cai Qizhe, Tang Ao, et al. TCP ≈ RDMA: CPU-efficient remote storage access with i10[C]//Proc of the 17th USENIX Symp on Networked Systems Design and Implementation (NSDI'20). Berkeley, CA: USENIX Association, 2020: 127−140

[44] Cheng Huang, Simitci H, Xu Yikang, et al. Erasure coding in windows azure storage[C]//Proc of the 2012 USENIX Conf on Annual Technical. Berkeley, CA: USENIX Association, 2012: 1−12

[45] Poke M, Hoefler T. DARE: High-performance state machine replication on RDMA networks[C]//Proc of the 24th Int Symp on High-Performance Parallel and Distributed Computing. New York: ACM, 2015: 107−118

[46] Mitchell C, Geng Yifeng, Li Jinyang. Using one-sided RDMA reads to build a fast, CPU-efficient key-value store[C]//Proc of the 2013 USENIX Conf on Annual Technical. Berkeley, CA: USENIX Association, 2013: 103−114

[47] Wu Haitao, Feng Zhenqian, Guo Chuanxiong, et al. ICTCP: Incast congestion control for TCP in data-center networks[J]. IEEE/ACM Transactions on Networking, 2013, 21(2): 345−358

[48] Lao ChonLam, Le Yanfang, Mahajan K, et al. ATP: In-network aggregation for multi-tenant learning[C]//Proc of the 18th USENIX Symp on Networked Systems Design and Implementation (NSDI'21). Berkeley, CA: USENIX Association, 2021: 741−761

[49] Zhang Jiao, Zhang Yali, Guan Zixuan, et al. HierCC: Hierarchical RDMA congestion control[C]//Proc of the 5th Asia-Pacific Workshop on Networking (APNet'2021). New York: ACM, 2021: 29−36

[50] Kumar G, Dukkipati N, Jang K, et al. Swift: Delay is simple and effective for congestion control in the datacenter[C]//Proc of the Annual Conf of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication. New York: ACM, 2020: 514−528

[51] Addanki V, Michel O, Schmid S. POWERTCP: Pushing the performance limits of datacenter networks[C]//Proc of the 19th USENIX Symp on Networked Systems Design and Implementation (NSDI'22). Berkeley, CA: USENIX Association, 2022: 51−70

[52] Zhong Xiaolong, Zhang Jiao, Zhang Yali, et al. PACC: Proactive and accurate congestion feedback for RDMA congestion control[C]//Proc of the 2022 IEEE Conf on Computer Communications (INFOCOM 2022). Piscataway, NJ: IEEE, 2022: 2228−2237

[53] Infiniband Trade Association. InfiniBand architecture volume 1, general specifications, release 1.2.1[EB/OL]. [2024-04-23]. https://www.infinibandta.org/ibta-specification/

[54] Recio R J, Culley P R, Garcia D, et al. A remote direct memory access protocol specification[EB/OL]. [2024-04-23]. https://doi.org/10.17487/RFC5040

[55] Infiniband Trade Association. Supplement to infiniBand architecture specification volume 1 release 1.2.2 annex A16: RDMA over converged Ethernet (RoCE)[EB/OL]. [2024-04-23]. https://www.infinibandta.org/ibta-specification/

[56] Infiniband Trade Association. Supplement to infiniBand architecture specification volume 1 release 1.2.2 annex A17: RoCEv2[EB/OL]. [2024-04-23]. https://www.infinibandta.org/ibta-specification/

[57] Zhu Yibo, Ghobadi M, Misra V, et al. ECN or delay: Lessons learnt from analysis of DCQCN and TIMELY[C]//Proc of the 12th Int Conf on Emerging Networking Experiments and Technologies. New York: ACM, 2016: 313−327

[58] Intel. Intel E810 Ethernet network adapter[EB/OL]. [2024-04-27]. https://www.intel.cn/content/www/cn/zh/products/details/ethernet/800-network-adapters/e810-network-adapters/products.html

[59] Broadcom. Broadcom BCM957508-P1200G NIC datasheet[EB/OL]. [2024-04-27]. https://docs.broadcom.com/doc/957508-P1200G-DS1XX

[60] NVIDIA. NVIDIA Connectx-7 network adapter datasheet[EB/OL]. [2024-04-27]. https://nvdam.widen.net/s/csf8rmnqwl/infiniband-ethernet-datasheet-connectx-7-ds-nv-us-2544471

[61] MARVELL. MARVELL converged Ethernet network adapter[EB/OL]. [2024-04-27]. https://cn.marvell.com/products/ethernet-adapters-andcontrollers/fastlinq-cna-adapters/documents.html

[62] Huawei. Huawei SP600 NIC[EB/OL]. [2024-04-27]. https://support.huawei.com/enterprise/zh/doc/EDOC1100309168?idPath=23710424%7C251364409%7C21782478%7C15791 (in Chinese)
(华为. 华为 SP600 网卡[EB/OL]. [2024-04-27]. https://support.huawei.com/enterprise/zh/doc/EDOC1100309168?idPath=23710424%7C251364409%7C21782478%7C15791)

[63] Chelsio. Chelsio T7 DPU products[EB/OL]. [2024-04-27]. https://www.chelsio.com/wp-content/uploads/resources/t7-dpu-asic.pdf

[64] Lee C, Park C, Jang K, et al. DX: Latency-based congestion control for datacenters[J]. IEEE/ACM Transactions Networking, 2017, 25(1): 335−348

[65] Mittal R, Lam V T, Dukkipati N, et al. TIMELY: RTT-based congestion control for the datacenter[C]//Proc of the 2015 ACM Conf on Special Interest Group on Data Communication. New York: ACM, 2015: 537−550

[66] Zhou Renjie, Dong Dezun, Huang Shan, et al. FastTune: Timely and precise congestion control in data center network[C]//Proc of the 2021 IEEE Int Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom). Piscataway, NJ: IEEE, 2021: 238−245

[67] Arslan S, Li Yuliang, Kumar G, et al. Bolt: Sub-RTT congestion control for ultra-low latency[C]//Proc of the 20th USENIX Symp on Networked Systems Design and Implementation (NSDI'23). Berkeley, CA: USENIX Association, 2023: 219−236

[68] Floyd S, Jacobson V. Random early detection gateways for congestion avoidance[J]. IEEE/ACM Transactions on Networking, 1993, 1(4): 397−413

[69] Ramakrishnan K, Floyd S, Black D. The addition of explicit congestion notification (ECN) to IP[EB/OL]. [2024-05-02]. https://doi.org/10.17487/rfc3168

[70] IEEE. IEEE 802.1 qau-congestion notification[EB/OL]. [2024-05-02]. http://www.ieee802.org/1/pages/802.1au.html

[71] Wu Haitao, Ju Jiabo, Lu Guohan, et al. Tuning ECN for data center networks[C]//Proc of the 8th Int Conf on Emerging Networking

Experiments and Technologies. New York: ACM, 2012: 25−36

[72] Shan Danfeng, Ren Fengyuan. ECN marking with micro-burst traffic: Problem, analysis, and improvement[J] IEEE/ACM Transactions on Networking, 2018, 26(4): 1533−1546

[73] Qian Kun, Cheng Wenxue, Zhang Tong, et al. Gentle flow control: Avoiding deadlock in lossless networks[C]//Proc of the ACM Special Interest Group on Data Communication. New York: ACM, 2019: 75−89

[74] The P4. org Applications Working Group. In-band network telemetry (INT) dataplane specification[EB/OL]. [2024-05-12]. https://p4.org/p4-spec/docs/INT_v2_1.pdf

[75] Ben Basat R, Ramanathan S, Li Yuliang, et al. PINT: Probabilistic in-band network telemetry[C]//Proc of the Annual Conf of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication. New York: ACM, 2020: 662−680

[76] Chelsio. Chelsio T5 packet rate performance report[EB/OL]. [2024-05-13]. https://www.chelsio.com/wp-content/uploads/2013/08/Chelsio-T5-Packet-Rate-Performance-Report.pdf

[77] Silicom. Dual port 10 gigabit server adapter with precision time stamping[EB/OL]. [2024-05-13]. https://www.silicom-usa.com/wp-content/uploads/2016/08/PE210G2TSI9-10G-Precision-Time-Stamping-Server-Adapter.pdf

[78] ntop. Using hardware timestamps with PF RING[EB/OL]. [2024-05-13]. https://www.ntop.org/pf_ring/hardware-timestamps-with-pf_ring/#:~:text=PF_RING%20has%20been%20enhanced%20to%20support%20hw%20timestamps%2C,the%20NIC%20that%20received%20packets%20must%20be%20timestamped

[79] ntop. Who (really) needs sub-microsecond packet simestamps[EB/OL]. [2024-05-13]. https://www.ntop.org/pf_ring/who-really-needs-sub-microsecond-packet-timestamps/

[80] NVIDIA Mellanox. Highly accurate time synchronization with connectX-3 and timeKeeper[EB/OL]. [2024-05-13]. https://enterprise-support.nvidia.com/s/article/highly-accurate-time-synchronization-with-connectx--174--3-and-timekeeper--174-x

[81] Brakmo L S, O'Malley S W, Peterson L L. TCP Vegas: New techniques for congestion detection and avoidance[J]. SIGCOMM Computer Communication Review, 1994, 24(4): 24−35

[82] Wei D X, Cheng Jin, Low S H, et al. FAST TCP: Motivation, architecture, algorithms, performance[J]. IEEE/ACM Transactions on Networking, 2006, 14(6): 1246−1259

[83] Tan Kun, Song Jingmin, Zhang Qian, et al. A compound TCP approach for high-speed and long distance networks[C]//Proc of the 25th IEEE Int Conf on Computer Communications (INFOCOM' 2006). Piscataway, NJ: IEEE, 2006: 1−12

[84] Jiang Huiling, Li Qing, Jiang Yong, et al. When machine learning meets congestion control: A survey and comparison[J]. Computer Networks, 2021, 192: 108033

[85] Fuhrer B, Shpigelman Y, Tessler C, et al. Implementing reinforcement learning datacenter congestion control in NVIDIA NICs[C]//Proc of the 23rd IEEE/ACM Int Symp on Cluster, Cloud and Internet Computing (CCGrid). New York: ACM, 2022: 331−343

[86] Dong Mo, Meng Tong, Zarchy D, et al. PCC vivace: Online-learning congestion control[C]//Proc of the 15th USENIX Conf on Networked Systems Design and Implementation. Berkeley, CA: USENIX Association, 2018: 343−356

[87] Sutton R S, Barto A G. Reinforcement Learning: An Introduction[M]. Cambridge, MA: A Bradford Book, 2018

[88] Tessler C, Shpigelman Y, Dalal G, et al. Reinforcement learning for datacenter congestion control[J]. SIGMETRICS Performance Evaluation Review, 2022, 49(2): 43−46

[89] Callon R. RFC1925: The twelve networking[EB/OL]. [2024-05-02]. https://doi.org/10.17487/RFC1925



Zhang Yutao, born in 2000. Master. His main research interests include RDMA and data center congestion control.

张毓涛, 2000 年生。硕士。主要研究方向为 RDMA、数据中心拥塞控制。



Yang Hui, born in 1986. PhD, associate professor. Her main research interests include RDMA and network processor.

杨　惠, 1986 年生。博士, 副研究员。主要研究方向为 RDMA、网络处理器。



Li Tao, born in 1983. PhD, associate professor. His main research interest includes high-performance network system and chip.

李　韬, 1983 年生。博士, 副研究员。主要研究方向为高性能网络系统及芯片。



Huang Mandi, born in 1999. Master. Her main research interest includes RDMA.

黄曼蒂, 1999 年生。硕士。主要研究方向为 RDMA。



Li Tianyun, born in 2000. Master. Her main research interests include data center network, RDMA congestion control, and artificial intelligence.

李天云, 2000 年生。硕士。主要研究方向为数据中心网络、RDMA 拥塞控制、人工智能。



Sun Zhigang, born in 1973. PhD, professor. His main research interests include software-defined network, time-sensitive networking, network architecture, and network security.

孙志刚, 1973 年生。博士, 研究员。主要研究方向为软件定义网络、时间敏感网络、网络体系结构、网络安全。