软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2024,35(4):1993−2021 [doi: 10.13328/j.cnki.jos.006939]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

传输控制中的确认机制研究*

李 彤[1,2], 郑 凯[3], 徐 恪[4,5,6]

[1](数据工程与知识工程教育部重点实验室 (中国人民大学), 北京 100872)

[2](中国人民大学 信息学院, 北京 100872)

[3](华为技术有限公司 计算机网络与协议实验室, 北京 100085)

[4](清华大学 计算机科学与技术系, 北京 100084)

[5](北京信息科学与技术国家研究中心 (清华大学), 北京 100084)

[6](鹏城实验室, 广东 深圳 518055)

通信作者: 徐恪, E-mail: xuke@tsinghua.edu.cn

摘　要: Internet 传输层协议需要依赖于确认 (ACK) 机制提供的反馈信息, 实现拥塞控制和可靠传输等功能. 以 Internet 传输协议演化的历史为线索, 回顾传输控制领域中的确认机制, 并讨论现有确认机制中需要解决的问题; 基于“类型-触发条件-信息” 三要素, 提出按需确认机制及其设计原则, 重点分析确认机制和拥塞控制、丢包恢复等传输协议子模块之间的耦合关系; 结合设计原则, 详细阐述一种可行的按需确认机制实现——TACK 机制, 并对相关概念进行系统的、深入的分析和澄清. 最后结合按需确认机制面临的挑战, 给出几个有意义的研究方向.

关键词: 传输协议; 确认机制; 拥塞控制; 丢包恢复; 按需确认

中图法分类号: TP393

中文引用格式: 李彤, 郑凯, 徐恪. 传输控制中的确认机制研究. 软件学报, 2024, 35(4): 1993–2021. http://www.jos.org.cn/1000-9825/6939.htm

英文引用格式: Li T, Zheng K, Xu K. Research on Acknowledgment Mechanisms of Transmission Control. Ruan Jian Xue Bao/Journal of Software, 2024, 35(4): 1993–2021 (in Chinese). http://www.jos.org.cn/1000-9825/6939.htm

Research on Acknowledgment Mechanisms of Transmission Control

LI Tong[1,2], ZHENG Kai[3], XU Ke[4,5,6]

[1](Key Laboratory of Data Engineering and Knowledge Engineering (Renmin University of China), Ministry of Education of the People's Republic of China, Beijing 100872, China)

[2](School of Information, Renmin University of China, Beijing 100872, China)

[3](Computer Network and Protocol Lab, Huawei Technologies Co. Ltd., Beijing 100085, China)

[4](Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China)

[5](Beijing National Research Center for Information Science and Technology (Tsinghua University), Beijing 100084, China)

[6](Peng Cheng Laboratory, Shenzhen 518055, China)

Abstract: Internet transport-layer protocols rely on the feedback information provided by the acknowledgment (ACK) mechanism to achieve functions such as congestion control and reliable transmission. According to the evolution of Internet transmission protocols, the ACK mechanisms of transmission control are reviewed. The unsolved problems among the mechanisms are discussed. Based on the elements of “type-trigger-information”, the ACK mechanism based on demand and its design principle are proposed, and the coupling relationship between the ACK mechanism and other transmission protocol submodules (e.g., congestion control, packet loss recovery, etc.)

* 基金项目: 国家自然科学基金 (62202473, 61932016); 国家杰出青年科学基金 (61825204); 北京高校卓越青年科学家计划 (BJJWZYJH 01201910003011)
收稿时间: 2022-03-31; 修改时间: 2022-07-27; 采用时间: 2023-03-23; jos 在线出版时间: 2023-08-16
CNKI 网络首发时间: 2023-08-17

is emphatically analyzed. Subsequently, according to the design principle, the TACK mechanism, a feasible ACK mechanism based on demand, is elaborated, and relative concepts are systematically clarified. Finally, several meaningful research directions are provided according to the challenges encountered by the ACK mechanism based on demand.

Key words: transmission protocol; acknowledgment mechanism; congestion control; loss recovery; acknowledgment on demand

随着网络条件和应用需求的不断演进, 针对传输控制 (transmission control) 相关的研究如雨后春笋般持续涌现, 40 年来仍然经久不衰. 传输控制, 是指在计算机网络中控制报文在主机到主机之间进行传输的过程. 传输控制的基本性能指标是高吞吐和低时延, 主要通过拥塞控制 (congestion control) 和丢包恢复 (loss recovery) 等功能模块实现[1]. 例如, 拥塞控制决定报文传输的快慢, 直接影响传输控制的性能; 丢包恢复决定报文传输的鲁棒性, 间接影响传输控制的性能[2−7].

在传输控制中, 不论是拥塞控制还是丢包恢复, 都依赖于数据收发的双方之间同步必要的信息. 这种通过目的主机 (即数据接收方)(数据报文的传送方向为从源主机到目的主机, ACK 报文的传送方向为目的主机到源主机. 本文中, 源主机也称为 (数据) 发送方, 目的主机也称为 (数据) 接收方. 因此, ACK 报文的传送方向是从 (数据) 接收方到 (数据) 发送方) 向源主机 (即数据发送方) 反馈信息的方式, 则称为确认机制 (acknowledgment mechanism). 在过去, 确认机制作为传输控制的反馈模块, 受到的关注过少, 20 世纪 90 年代后很少有大的改动. 这是因为, 在传输控制的过程中, 业务数据往往在源主机到目的主机的路径 (即正向路径) 上传输, 故人们通常关注正向路径的数据报文传输性能, 却很少关心反向路径 (即目的主机到源主机) 上确认 (acknowledgment, ACK) 报文的传输.

然而, 现代新型业务如高性能计算、分布式 AI、实时音视频通话、4K/8K 无线投屏、直播、云虚拟现实 (cloud VR)、云游戏等持续涌现, 网络数据爆炸性增长, 在多样化的业务需求和动态变化的网络条件下, 传输控制高吞吐、低时延的目标能否达成, 往往与确认机制如何设计息息相关. 例如, 发生拥塞时, 拥塞控制模块依赖 ACK 及时、准确地反馈连接状态, 从而精准地调整发送速率; 发生丢包时, 丢包恢复模块依赖 ACK 及时、全面地通知包到达信息, 从而高效地进行丢包重传.

本文是业界首个针对数据传输协议确认机制进行系统性研究和分析的工作. 本文将系统地梳理传输控制中确认机制的发展历程, 对比分析现有的确认机制的特点, 并结合下一代传输控制的演进思路, 提出按需确认机制的系统性设计理论. 最后, 总结并讨论确认机制的未来研究方向.

1 问题与挑战

多样化的业务需求和动态变化的网络条件, 对主机到主机之间的数据传输提出了更高的要求. 在这种情况下, 传输控制中的确认机制设计将面临以下挑战.

(1) 带宽适应性

从卫星通信网、蜂窝网、无线局域网、云际网到数据中心网络, 数据传输的链路带宽 (bandwidth) 可能从几十 Kb/s, 增长到几十 Gb/s 乃至 100 Gb/s[8,9]. 若数据接收方 (即目的主机) 每接收到固定数目的数据报文后, 回复一个 ACK 报文, 则 ACK 报文的数目将随着带宽线性增长, 可能导致较大的计算开销和通信开销. 因此, 确认机制的设计应该考虑其带宽适应性.

(2) 时延适应性

数据中心网络的往返时延 (round-trip time, *RTT*) 可能低至微秒级, 云际网的往返时延可以达到百毫秒级, 而卫星通信网络的往返时延可能高达秒级[10−12]. 若数据接收方每隔固定的时间周期 (记为 α) 回复一个 ACK 报文, 则可能导致反馈严重滞后 (例如 $\alpha \gg RTT$), 导致传输控制效率低下. 因此, 确认机制的设计应该考虑其时延适应性.

(3) 抖动适应性

抖动包括带宽抖动和时延抖动. 不论是有线网络还是无线网络, 抖动都是客观存在的. 以无线网络为例, 在数据传输过程中, 无线空口的可用带宽抖动剧烈, 每一毫秒都在变化, 主要原因是无线信号易受同频和临频信号的干扰[13]; 同时, 无线网络传输端到端时延抖动也很剧烈. 主要原因是无线网络链路层会重传因信号干扰丢失的报文, 链路层的重传在上层的传输层看来, 结果就是时延倍增和抖动. 带宽和时延的抖动, 将会直接影响 ACK 反馈信息

的时效性和精准性, 因此, 确认机制的设计应该考虑其抖动适应性.

(4) 反向丢包

确认机制中, 通过 ACK 报文对接收的数据报文进行确认, 初衷是为了应对正向路径上的丢包问题. 然而, 反向路径上的 ACK 报文本身也会丢失. 如果正向路径没有丢包, 只有反向路径存在丢包, 传输控制的性能影响可能较小. 然而, 真实网络的链路往往同时存在正向丢包和反向丢包. 因此, 如何应对反向丢包问题, 也是确认机制设计中需要考虑的因素.

(5) 反向拥塞

一方面, ACK 报文所在的连接可能与其他的连接共享瓶颈链路, 因此反向路径也可能产生拥塞. 另一方面, 卫星通信网和蜂窝网等通常会部署非对称链路技术, 即下行链路的带宽是上行链路带宽的数倍. 当上行链路被 ACK 报文充满时, 其下行链路的数据报文的吞吐将受到限制. 在这些情况下, 确认机制的设计应该考虑减少反向路径上的流量, 以应对反向拥塞带来的不良影响.

(6) 内部干扰

以上挑战可以归类为来自连接外部的干扰, 简称为“外部干扰”. 然而, 对于基于 IEEE 802.11 标准的无线网络而言, 除了外部干扰, 还存在内部干扰. 内部干扰是指同一连接内的 ACK 报文竞争数据报文的频谱资源而产生的干扰. 产生内部干扰的原因是数据报文所在的正向路径和 ACK 报文所在的反向路径共享有限的无线频谱资源, 并且发送一个报文需要的额外开销几乎与报文大小无关——即使一个 ACK 报文通常比一个数据报文小很多, 但发送一个 ACK 报文将消耗与数据报文几乎相同的频谱资源[14]. 因此, 在 ACK 报文的开销不可忽略的场景, 确认机制的设计还应该考虑 ACK 报文造成的内部干扰.

2 确认机制的定义与内涵

2.1 传输控制技术背景

2.1.1 传输控制的定义

计算机之间的数据传输网络, 本质上是一个控制系统. 传输控制是指通过共同遵循的某种协议, 将数据从一个网络节点转移到另一个网络节点的过程. 在计算机网络 OSI 模型中, 传输层是位于网络层之上, 应用层之下的中间层, 传输控制主要完成第 4 层传输层所指定的功能.

传输控制是通过传输层协议实现的. TCP (transmission control protocol) 和 UDP (user datagram protocol) 是两种最常见的传输层协议. TCP 通常需要保证可靠传输, 并配套相应的拥塞控制手段. 而 UDP 本身是“尽最大努力”的, 无需保证可靠, 也不配套相应的拥塞控制手段. 但是, 近年来, 基于 UDP 的可靠或部分可靠的传输控制 (例如 DCCP[14]、QUIC (quick UDP Internet connection)[15]、VOXEL[16] 等) 已经成为业界关注的热点. 具体地, 在 UDP 传输的基础上, 灵活根据应用的需求, 在应用层和 UDP 传输层之间添加一个适配层, 通过在适配层中实现可靠传输和拥塞控制, 实现类似 TCP 的传输效果.

2.1.2 传输控制的功能模块

传输控制的功能模块关系如后文图 1 所示. 数据收发的基本流程是: (1) 发送方从上层应用读取数据, 放在发送缓存中, 发送模块将数据封装成报文后, 通过正向路径上的网络链路将数据报文传输到接收方, 随后放入接收缓存中; (2) 在接收数据报文的过程中, 接收方对每个报文的到达情况、接收缓存剩余量等进行状态监测, 同时根据确认机制, 生成 ACK 报文后, 通过反向路径上的网络链路将 ACK 报文传输到 (数据) 发送方, 供反馈处理模块进行解析; (3) 发送方根据解析的 ACK 反馈信息, 进一步完成丢包恢复、状态监测和速率控制等.

从上述流程可以看出, 不论是状态监测、丢包恢复、还是速率控制, 这些功能模块都需要与确认机制进行交互.

2.2 确认机制的演进历史

传输控制中的确认机制, 是指数据接收方通过确认报文 (ACK) 对数据发送方发送过来的数据报文的传输结果进行确认的机制. 本文研究的确认机制, 对 OSI 模型对应的传输层和应用层均适用. 针对本身自带 ACK 的传输

层协议, 如 TCP, 确认机制工作在传输层; 针对基于 UDP 的应用层传输协议, 如 QUIC, 确认机制工作在应用层, 其主要作用是在 UDP 的数据传输能力基础上, 增强应用系统数据传输的可靠性, 或者帮助应用系统数据传输进行拥塞控制.

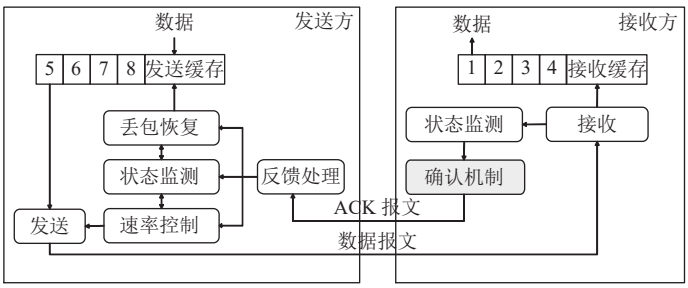


图 1 传输控制的功能模块关系

自 1974 年 Cerf 等人[17]提出 TCP 以来, TCP 经历了 30 年的持续优化, 并在 2000 年左右确定了基本框架. 基本框架确定过程中, 有一些关键算法模块也有其各自的演进路线. 其中, 最典型的就是拥塞控制算法和确认机制. 拥塞控制是网络领域经久不衰的、最经典的研究课题之一. 如图 2 所示, 2000 年以后, 随着网络条件和应用需求的不断演进, 新型的拥塞控制算法如雨后春笋般持续涌现, 在不同的场景和目标中各显神通. 例如, Tan 等人提出 Compound TCP[18] 是 Windows 系统使用的拥塞控制优化算法; Winstein 等人提出适应蜂窝网络带宽抖动的 Sprout[19] 以及基于机器学习的拥塞控制算法 Remy[20] 和 Indigo[21] 等. Cardwell 等人提出 BBR[22], 被认为是继 CUBIC[23] 以后拥塞控制研究的一个新的里程碑.

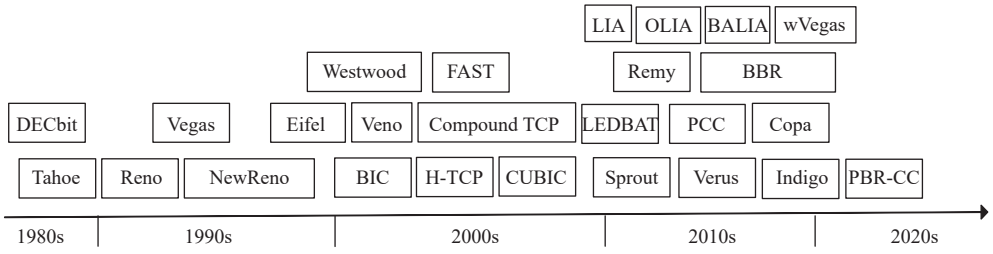


图 2 拥塞控制算法演进

相比拥塞控制, 确认机制相关的工作较少. 如图 3 所示, 确认机制作为传输控制的反馈模块, 20 世纪 90 年代后很少有大的改动. 这是因为, 人们通常关注正向路径的数据报文传输性能, 却很少关心反向路径上 ACK 报文的传输. 而且, 确认机制与传输控制中的拥塞控制、丢包恢复和状态监测等模块紧密耦合, 贸然修改确认机制容易导致副作用. 然而, 在动态变化的网络条件下, 设计更为灵活的、自适应的确认机制, 支撑现代新型业务的多样性, 已经成为业界迫切的需求[24].

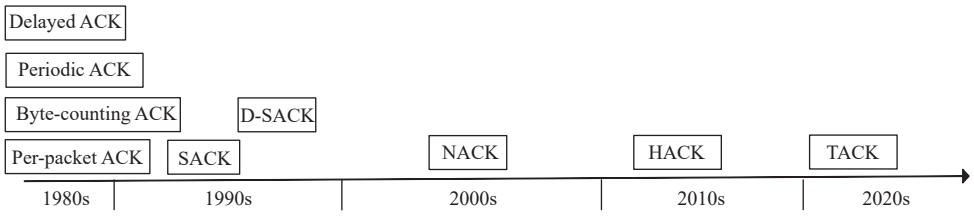


图 3 确认机制演进

针对协议的修改, 其鲁棒性严重依赖于充分的单元测试和端到端测试. 在过去, 人们默认使用内核态协议栈 (例如内核 TCP). 由于内存受限、内核接口受限等因素影响, 加上确认机制涉及协议的多个模块的修改, 针对确认

机制的大幅度的修改显得非常困难. 幸运的是, 用户态协议栈 (例如 QUIC) 的出现, 为人们提供了一个灵活、广阔的“罗马广场”. 用户态协议栈允许大规模的日志跟踪和调试, 针对协议的修改和部署非常方便, 为确认机制的修改乃至重新设计提供了基本前提.

2.3 确认机制三要素

确认机制通常包含类型、触发条件和信息 3 大要素. 类型是指 ACK 报文的类型, 不同类型的 ACK 报文, 可能有不同的触发条件, 也可能携带不同的反馈信息. 例如 SACK 与 NACK (或 NAK) 就是两种不同的 ACK 报文类型, 两者触发条件分别为接收方检测到发生乱序事件和接收方检测到发生丢包事件, 两者携带的反馈信息也正好相反. 下面, 我们将对确认机制三要素进行详细介绍.

2.3.1 ACK 触发条件

ACK 触发条件决定 ACK 生成并发送的时机. 存在以下 3 种基本的触发条件.

基本条件 1: 如图 4(a) 所示, 接收方接收到指定数目的数据报文后, 立即回复一个 ACK 报文.

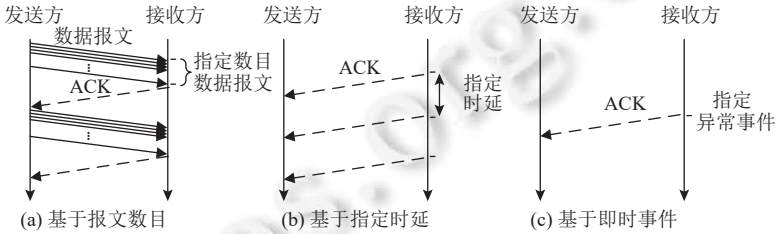


图 4 触发 ACK 的 3 个基本条件

基本条件 2: 如图 4(b) 所示, 不管有没有数据报文到达, 接收方等待指定时延后, 回复一个 ACK 报文.

基本条件 3: 如图 4(c) 所示, 当发生指定即时 (instant) 事件时, 接收方回复一个 ACK 报文. 即时事件是指网络状态异常或者在短时间内发生突变, 例如时延或带宽变化达到一个阈值, 或者丢包率、丢包数目或乱序程度达到一个阈值, 又或者接收缓存剩余容量达到一个阈值等.

其他的 ACK 触发条件, 均是以上 3 种基本条件的一种或者多种的组合. 例如, 在经典 TCP 的延迟确认机制中, 同时使用了上述 3 种基本条件. (1) 每收到两个数据报文后, 必须回复一个 ACK 报文, 采用了基本条件 1; (2) 如果没收到两个数据报文, 但是等待了超过 500 ms (实际的系统实现不一定是 500 ms, 它与系统版本相关, 例如 Linux 系统中是 200 ms, Windows 系统中是 100 ms), 必须回复一个 ACK 报文, 这里采用了基本条件 2; (3) 当发生乱序时, 每收到一个数据报文, 回复一个 ACK 报文, 这里乱序视为即时的异常事件, 表示采用了基本条件 3, 同时每收到一个数据报文回复一个 ACK 报文, 表示采用了基本条件 1.

在传输控制中, 存在一种特殊的 ACK 触发方式, 称为 ACK Piggybacking (捎带). ACK Piggybacking 同样可以视为上述 3 个基本条件的组合. 具体地, 接收方收到数据报文后 (基本条件 1), 等待指定时间 (基本条件 2), 如果有数据要发送 (基本条件 3), 则采用 Piggybacking 的方式发送数据报文, 其中数据报文中携带 ACK 报文需要反馈的信息; 如果没有数据要发送, 直接发送 ACK 报文. 在这个例子中, 我们假设了基本条件 1 和基本条件 2 的优先级高于基本条件 3, 实际的协议实现可以根据需求自由定义 3 个基本条件之间的优先关系, 从而实现灵活、多样化的确认机制.

2.3.2 ACK 携带的信息

ACK 携带的信息, 用于 (数据) 接收方向 (数据) 发送方反馈传输状态. 反馈的传输状态信息包括但不限于以下几种.

(1) 时延相关的信息

时延相关的信息包括往返时延、单向时延、排队时延、截止时间等其中的一种或多种.

(2) 带宽相关的信息

带宽相关的信息包括发送速率、接收速率、网络带宽等其中的一种或多种.

(3) 抖动相关的信息

抖动相关的信息包括时延抖动、带宽抖动等其中的一种或多种.

(4) 控制信息

用于发送方进行决策的控制参数可以是一个阈值参数, 也可能是一个指示某种状态的标记.

(5) 丢包信息

携带丢包信息的 ACK 分为携带最新丢包信息的 ACK 和携带冗余历史丢包信息的 ACK. 丢包信息是指接收方接收缓存中不连续的数据段序列之间的间隔. 例如, 接收端缓存中有序列号分别为 (2, 5, 6, 8, 9) 的 5 个报文, 则最新丢包信息为 7 这个报文的序列号, 历史丢包信息为 1, 3 和 4 这 3 个报文的序列号.

(6) 其他网络状态信息

其他网络状态信息包括接收端缓存中已接收但未提交到上层应用的数据量、接收端缓存剩余大小、乱序程度、用于发送方进行决策的控制参数等其中的一种或多种.

2.3.3 ACK 的类型

ACK 的类型通常可以按触发条件进行分类, 或者按报文格式进行分类.

(1) 按触发条件分类

触发条件不同的 ACK 能否视为不同种类的 ACK, 取决于 ACK 的功能是否有差异. 例如, 周期性发送 ACK, 和根据即时事件回复 ACK, 由于前者保证反馈的鲁棒性, 后者保证反馈的实时性, 因此可以视为两种不同类型的 ACK. 又比如, 每收到 A 个报文触发的 ACK 和每收到 B ($A \neq B$) 个报文触发的 ACK, 如果只是 ACK 频率上的区别, ACK 功能上没有差异, 一般地, 可以视为同一种 ACK 类型.

(2) 按报文格式分类

一般地, 先后生成的 ACK 携带的内容是有差异的, 例如确认的序列号可能不同. 但如果差异只是体现在信息的内容上, 则可以视为同一种 ACK 类型. 例如, TCP 协议中带 SACK 选项的 ACK 和不带 SACK 选项的 ACK, 可以视为同一种 ACK 类型, 原因是 ACK 的报文格式没有发生变化, 不带 SACK 选项时, 对应字段的内容为空而已. 然而, 如果 ACK 报文格式发生了变化, 即 ACK 中的相应字段表示的含义不同, 则应该视为不同种类型的 ACK. 例如, 仅携带时延信息的 ACK 和仅携带丢包信息的 ACK, 是两种不同的 ACK 类型.

2.4 确认机制与其他模块的耦合关系

从图 1 所示传输控制的功能模块关系可以看出, 不论是状态监测、丢包恢复、还是速率控制, 这些功能模块都需要与确认机制进行交互. 下面, 我们将讨论传输控制协议中确认机制与上述功能模块之间的耦合关系.

2.4.1 确认机制与状态监测

状态监测可以发生在发送方, 也可以发生在接收方. 在接收方, 状态监测为确认机制提供信息输入, 用于决策 ACK 的触发时机和 ACK 应该携带的反馈信息等; 在发送方, 确认机制决定以何种方式解析 ACK 报文, 解析结果将用于传输过程中的状态监测.

确认机制中的 ACK 触发条件和携带的信息, 会直接影响状态监测的精准性. 如图 5 所示, 假设每个 ACK 报文默认只携带最近接收到的数据报文的发送时间戳 t_1、最近接收到的数据报文的接收时间戳 t_3 和该 ACK 报文的发送时间戳 t_4, 则发送方可以根据每一个 ACK 报文的到达时间戳 t_2, 计算一个 RTT 采样值: $RTT = t_1 - t_2 - (t_4 - t_3)$.

考虑不同 ACK 数目下的 RTT 评估. 假设发送方共发送 6 个数据报文, 每个数据报文对应的实际 RTT 分别为 18 ms, 20 ms, 16 ms, 22 ms, 15 ms 和 30 ms, 最小 RTT 实际值为 15 ms. 如图 6(a) 所示, 若每收到 1 个数据报文回复 1 个 ACK 报文, 则发送方的 RTT 采样值集合为{18, 20, 16, 22, 15, 30}, 则最小 RTT 测量值为 15 ms, 与实际值相等; 如图 6(b) 所示, 若每收到 2 个数据报文回复 1 个 ACK 报文, 则发送方的 RTT 采样值集合为{20, 22, 30}, 则最小 RTT 测量值为 20 ms, 与实际值偏差 30%; 如图 6(c) 所示, 若每收到 6 个数据报文回复 1 个 ACK 报文, 则发送方的 RTT 采样值集合为{30}, 则最小 RTT 测量值为 30 ms, 与实际值偏差 1 倍.

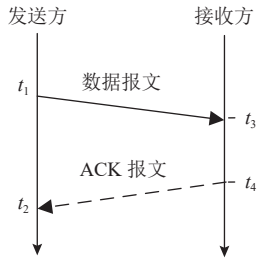


图 5 *RTT* 评估方式示例

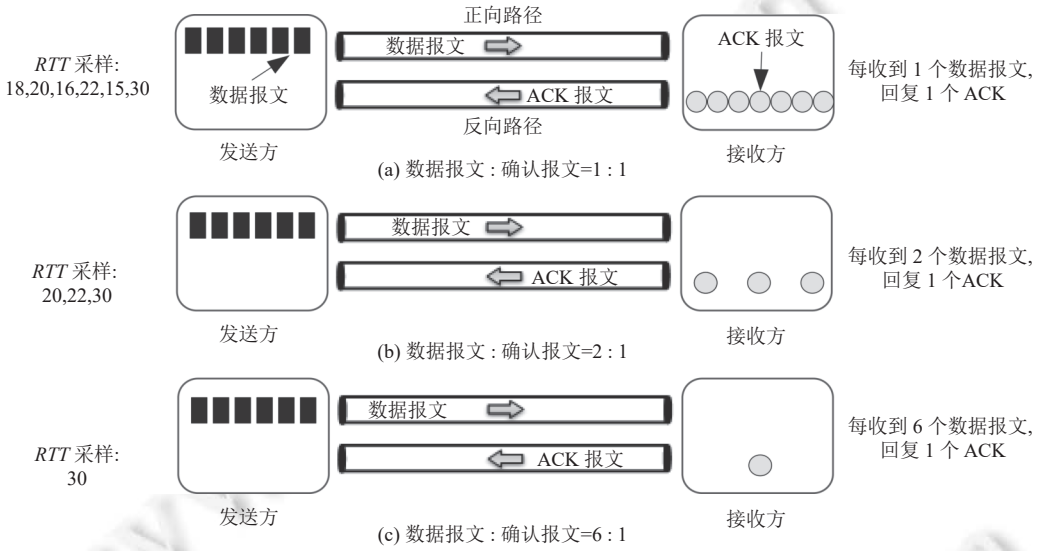


图 6 不同 ACK 数目下的 *RTT* 评估

从上面的例子可以得到, 确认机制的修改, 应该避免显著影响传输状态监测的精准性. 对于按需确认机制而言, ACK 触发条件的设计, 以及 ACK 携带的信息, 需要满足精准的传输状态监测的需求.

2.4.2 确认机制与丢包恢复

丢包恢复包括丢包检测和数据重传两个关键步骤. 丢包检测是丢包恢复的前提, 而 ACK 的触发时机和 ACK 携带的反馈信息, 则直接影响丢包检测的性能. 因此, 确认机制直接影响丢包恢复的效率.

对于需要保障可靠和有序的传输控制而言, 行头阻塞 (head-of-line blocking, HoLB)(详细分析见第 4.1.4 节) 导致报文组装延迟, 从而影响传输性能. 后文图 7 和图 8 对比了两种不同的 ACK 触发方式对报文组装延迟的影响. 在图 7 中, 每收到 1 个数据报文回复 1 个 ACK 报文, 假设丢包检测采用经典 TCP 默认的 DupACK, 即“3 次重复 ACK”算法[25], 则数据报文 3 从丢失到恢复所花费时间长短, 对于报文重组装具有直接影响. 具体地, 报文 3 的丢失阻塞了后续队列, 使得后续的 8 个数据报文无法及时提交到上层应用. 在图 8 中, 每收到 100 个数据报文回复 1 个 ACK 报文, 假设理想情况下发送方可以通过该 ACK 报文推测丢包事件, 则报文 3 的丢失阻塞了后续队列, 使得后续的 98 个数据报文无法及时提交到上层应用. 因此, ACK 触发方式直接影响丢包恢复效率, 一般地, ACK 触发时间间隔越长, 则丢包恢复的效率越低.

从上面的例子可以看出, 确认机制的设计, 应该考虑其是否会显著影响丢包恢复的效率. 具体而言, ACK 触发条件的设计, 以及 ACK 携带的信息, 需要满足高效的丢包恢复的需求.

2.4.3 确认机制与速率控制

速率控制包括拥塞控制和流量控制. 拥塞控制和流量控制的区别, 在于拥塞控制的目标是优化整网性能, 而流

量控制的目标是优化收发双方之间的连接性能. 拥塞控制依赖于 ACK 报文反馈的传输时延、速率和丢包等信息, 评估可用的网络瓶颈带宽, 从而调整发送速率; 流量控制依赖于 ACK 报文反馈接收方的接收缓存占用情况, 从而避免因发送方的发送速率过高导致接收方接收缓存满的情况.

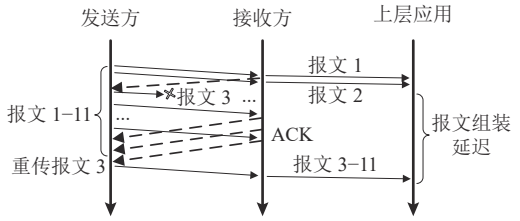


图 7 丢包恢复: 每收到 1 个数据报文回复 1 个 ACK

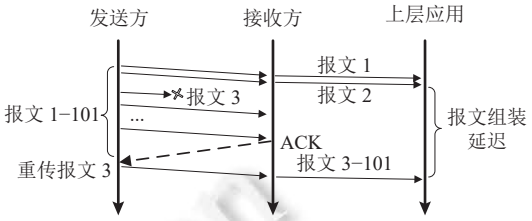


图 8 每收到 100 个数据报文回复 1 个 ACK

确认机制的设计, 直接影响拥塞控制的效果. 发送方进行拥塞控制时, 需要根据接收到的 ACK 报文的到达情况及其反馈的信息, 对网络的传输管道进行评估, 从而确定最大可发送的数据量 (即拥塞窗口) 或者最大发送速率. 因此, ACK 的触发时机影响拥塞控制的更新粒度和数据发送过程中的流量模式. 而 ACK 携带的反馈信息, 则直接影响拥塞控制的精准性. 图 9 从发送方的视角, 对比了不同 ACK 触发间隔时间对流量模式的影响. 如图 9(a) 所示, 当 ACK 间隔时间短时, 数据的发送速率相对恒定; 然而, 如图 9(b) 所示, 当 ACK 间隔时间较长时, 流量呈现突发的特征. 众所周知, 突发流量可能导致瓶颈链路缓存满导致丢包, 从而严重地影响整网的传输效率.

发送方进行流量控制时, 需要根据 ACK 反馈的接收窗口大小, 相应地调整发送速率. 具体地, 发送速率取决于发送窗口, 而发送窗口通常取拥塞窗口和接收窗口的较小值. 也就是说, 速率控制是由拥塞控制中的拥塞窗口和流量控制中的接收窗口共同决定的, 类似于“木桶效应”, 处于瓶颈的一方起到控制作用. 图 10 给出了接收方接收缓存结构示意图. 其中, 接收窗口大小等于接收方接收缓存中的剩余缓存量, 由 ACK 报文反馈给发送方. 已缓存的数据, 其序列号空间连续的部分可以提交至上层应用, 不连续的部分则造成行头阻塞现象, 必须继续保留在接收缓存中, 等待行头的报文成功到达后, 才能提交至上层应用.

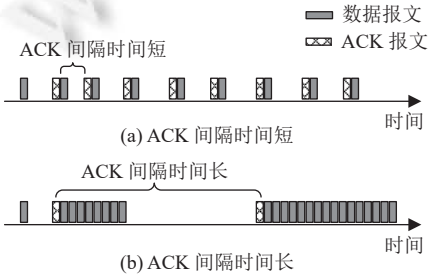


图 9 流量模式与 ACK 间隔时间的关系

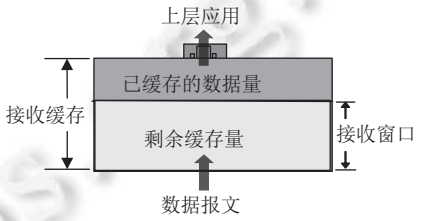


图 10 接收方接收缓存

另一方面, 确认机制的设计, 直接影响流量控制的效果. 如图 11 所示, 当 ACK 触发时间间隔较大时, 可能造成反馈延迟, 进而导致带宽利用率低下. 举例来说, 设 ACK 触发时间间隔为 50 ms, 当部分数据报文丢失导致行头阻塞时, 接收缓存会呈现即被填满的状态. 此时, 接收方发送 ACK1 通告一个小接收窗口 (例如接近 0). 当发送方接收到 ACK1 报文后, 将减少或停止数据发送. 然而, 如果此时由于数据重传成功, 接收方行头阻塞恢复, 即使已占用的接收缓存得到释放, 发送方也无法继续发送数据. 在这种情况下, 发送方需要等接收方 50 ms 后触发 ACK2, ACK2 通告一个大于 0 的接收窗口后, 发送方才能恢复发送数据. 在这一过程中, 原本发送方可以发送数据的机会, 可能被白白浪费掉了. 因此, ACK 延迟触发会降低传输效率.

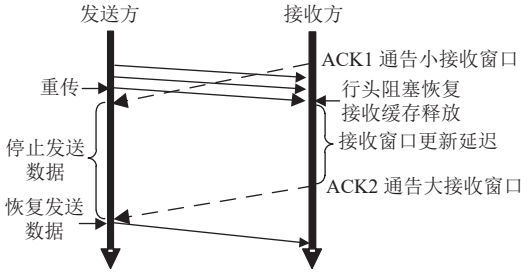


图 11 ACK 延迟触发降低传输效率

2.5 按需确认机制设计原则

为了适应异构网络环境的动态变化以及不同业务的多样化需求, 解决带宽适应性、时延适应性、抖动适应性、反向丢包、反向拥塞和内部干扰等挑战, 业界提出按需确认的新理念 (acknowledgment on demand, AOD). 按需确认机制试图重新思考和设计反馈信号, 以寻求最优的确认机制, 支撑高效的数据传输. “按需确认”, 本质的含义是, 生成的 ACK 报文必须是传输控制所必需的、并且是足够的, 二者缺一不可. 如图 3 所示的确认机制演进历程中的 TACK 机制, 则是按需确认机制的一种实现, 我们将在后面的章节对其进行详细讨论.

根据上述的讨论, 结合确认机制“类型-触发条件-信息”三要素的定义, 按需确认机制的基本设计原则, 可以概括为以下几点.

(1) 按需确认机制, 要求 ACK 是根据传输状态按需触发的. 按需触发, 就是指要求根据传输的状态, 选取 3 个基本条件的最优组合和参数配置.

(2) 按需确认机制, 要求 ACK 根据传输的状态按需携带必要的信息. 按需要携带必要的信息, 就是要 (数据) 接收方在上述描述的信息中, 选取必要的、足够的信息, 通过 ACK 报文反馈给 (数据) 发送方.

(3) 按需确认机制, 要求 ACK 根据传输的状态按需使用最合适的 ACK 类型. ACK 的类型是多样的, 不同类型的 ACK 触发条件不同或者携带的信息不同. 按需使用最合适的 ACK 类型, 就是要 (数据) 接收方根据传输状态, 综合 ACK 触发条件和 ACK 携带的信息, 在最佳的时机, 通过 ACK 报文反馈最必要的、足量的信息给 (数据) 发送方.

(4) 确认机制必须支撑传输控制协议其他模块的有序工作, 保证低开销和高性能.

3 基于三要素的现有机制分析与比较

3.1 基本的确认机制分类

3.1.1 Per-packet ACK 机制

Per-packet ACK 机制是指接收方每收到一个数据报文, 回复一个 ACK 报文. 用 f 表示 ACK 的发送频率, 即每秒发送的 ACK 报文数目, 以赫兹 (Hz) 为单位. 则 Per-packet ACK 机制的 ACK 发送频率可以表示为:

$$f_p = \frac{bw}{PKT\_SIZE} \tag{1}$$

其中, bw 表示发送方到接收方之间的吞吐量; PKT_SIZE 表示数据报文大小. 其中, $PKT\_SIZE \leqslant MSS$, MSS 表示最大段长度 (maximum segment size).

Per-packet ACK 机制是一种以数据报文到达事件驱动的反馈机制. 这种机制可能的问题在于: (a) 当存在大量的小数据报文时 ($PKT\_SIZE \leqslant MSS$), ACK 报文开销将不可忽略; (b) 当发送方不发数据时, 发送方无法得到 ACK 反馈. 因此, 发生尾包丢失事件时, 发送方无数据可发, 接收方无法反馈丢包信息, 则可能引发超时问题.

3.1.2 Byte-counting ACK 机制

当存在大量的小数据报文时, 一种改进的方法就是计算累计的数据量, 如果达到指定阈值则回复 ACK 报文. 这种机制被称为 Byte-counting ACK 机制. Byte-counting ACK 机制的 ACK 发送频率可以表示为:

$$f_b = \frac{bw}{L \cdot MSS} \tag{2}$$

即, 每累计收到 L 个 MSS 大小的数据报文后回复 1 个 ACK 报文. L 表示接收方在回复 ACK 之前收到的数据报文数目. L 越大, ACK 频率越小. 然而, 指定 L 的取值后, f_b 与 bw 正相关. 这意味着当 bw 极大时, ACK 频率可能仍然处于较高水平. 换句话说, ACK 发送频率在带宽变化时无法保证收敛到较低的值.

与 Per-packet ACK 机制一样, Byte-counting ACK 机制也是一种以数据报文到达事件驱动的反馈机制. 它解决了小数据报文开销问题, 但是同样面临着发送方不发数据时无 ACK 反馈的问题.

3.1.3 Periodic ACK 机制

Per-packet ACK 和 Byte-counting ACK 机制将 ACK 报文的发送和数据报文的到达紧密耦合, 导致大带宽传输下仍然存在较大的 ACK 报文开销. 一种解决思路是, 不管有没有数据包到达, 接收方周期性地回复 ACK 报文, 在接收方和发送方之间同步信息, 这种机制被称为 Periodic ACK 机制. 该机制的 ACK 发送频率可以表示为:

$$f_{\text{pack}} = \frac{1}{\alpha} \tag{3}$$

其中, α 为 ACK 发送周期, 表示两个 ACK 报文之间的时间间隔.

Periodic ACK 机制可以保证在大带宽传输下, 保持一个相对恒定的 ACK 频率. 然而, 当带宽极小时, ACK 频率仍然与大带宽情况保持一致, 显然会浪费资源. 因此, Periodic ACK 机制也无法适应带宽的变化.

3.1.4 Delayed ACK 机制

Delayed ACK 机制是现代传输控制协议如 TCP 和 QUIC 所采用的默认确认机制[26–28]. 具体地, Delayed ACK 机制规定接收方超时或者每收到 L 个数据报文 (报文大小等于 MSS) 时, 回复 ACK 报文. 该机制的 ACK 发送频率可以表示为:

$$f_{\text{delayed}} = \max\left(\frac{bw}{L \cdot MSS}, \frac{1}{\alpha}\right) \tag{4}$$

根据 RFC 1122 和 RFC 5681 规定, $L \leqslant 2$, 且 α 的取值为几十到几百毫秒, 与具体的系统版本相关. 例如, 最新版本的 Ubuntu 中 $\alpha = 200$ ms.

Delayed ACK 机制综合了 Byte-counting ACK 和 Periodic ACK 机制, 但是限定了 L 和 α 的取值, 使得 ACK 频率无法达到最小化.

当然, 即使允许 L>2, Delayed ACK 也不是一种最优的最小化 ACK 频率的机制. 当 bw 满足 $\frac{bw}{L \cdot MSS} > \frac{1}{\alpha}$ 时, 则 $f_{\text{delayed}} = \frac{bw}{L \cdot MSS}$. 在这种情况下, bw 越大, ACK 频率越高. 另一方面, 当 bw 满足 $\frac{bw}{L \cdot MSS} < \frac{1}{\alpha}$ 时, $\frac{bw}{L \cdot MSS} = \frac{1}{\alpha}$. 在这种情况下, ACK 频率不能随 bw 的减小而降低. 这意味着 Delayed ACK 机制也无法适应带宽的变化, 即 ACK 发送频率在带宽变化时无法保证收敛到较低的值.

3.1.5 Bounded ACK 机制

Bounded ACK 机制, 它的目标是在网络条件动态变化下, 使得 ACK 的发送频率是有界的 (bounded). 其基本原理就是汲取 Byte-counting ACK 和 Periodic ACK 两种机制的优点, 在不同的场景下自适应地选择不同的方式. 具体地, 由公式 (4) 可知, Delayed ACK 机制采用了 Byte-counting ACK 和 Periodic ACK 两者中的 ACK 频率的较大值. Bounded ACK 机制与 Delayed ACK 机制正好相反——ACK 发送频率采用 Byte-counting ACK 和 Periodic ACK 两者中的 ACK 频率的较小值, 即:

$$f_{\text{bounded}} = \min\left(\frac{bw}{L \cdot MSS}, \frac{1}{\alpha}\right) \tag{5}$$

因此, 当 bw 较大时, Bounded ACK 机制的 ACK 频率保持在一个相对恒定的水平 (即 Periodic ACK); 当 bw 较小时, Bounded ACK 机制回退到 Byte-counting ACK, 此时, ACK 频率随着 bw 等比例减小. 因此, Bounded ACK 机

制能够使得 ACK 的频率在不同的带宽场景下保持较低水平, 具有较强的带宽适应性.

3.1.6 Tame ACK 机制

Tame ACK 机制也称为 TACK 机制[29,30], 它的目标是在网络条件动态变化下, 最小化 ACK 的发送频率. TACK 机制是针对 Bounded ACK 机制的进一步改进, 不仅具有带宽适应性, 还具有时延适用性. 根据公式 (5), 具体实现时, TACK 机制将 α 的取值设置成 RTT 的因数, 即 $\alpha=\frac{RTT_{\min}}{\beta}$. $RTT_{\min}$ 表示一段时间内的最小 RTT, β 表示每个 $RTT_{\min}$ 回复的 ACK 的数目. 由于 bw 表示实时的吞吐量, 在实际实现中无法及时更新, 因此 TACK 机制采用了其统计值, 即 $bw_{\max}$, 表示一段时间内的 bw 最大值, 则 TACK 机制的 ACK 频率 f_{tack} 可以表示为:

$$f_{tack}=\min\left(\frac{bw_{\max}}{L\cdot MSS},\frac{\beta}{RTT_{\min}}\right) \tag{6}$$

定义带宽时延积 (bandwidth and delay product, BDP) 为 $BDP=RTT_{\min}\times bw_{\max}$. 则 TACK 机制频率更新方法如下: (1) 当 BDP 大于或等于 $\rho\times L\times MSS$ 时, 该 ACK 的频率更新为: 每经过 $RTT_{\min}$ 时间发送 β 个 ACK; (2) 当 BDP 小于 $\rho\times L\times MSS$ 时, 该 ACK 的频率更新为: 每收到 $L\times MSS$ 字节的数据报文后, 发送 1 个 ACK.

因此, 当 BDP 较大时, TACK 机制的 ACK 频率保持在一个相对恒定的水平 (即 Periodic ACK), 同时与 $RTT_{\min}$ 成比例进行变化, 具有较强的时延适应性; 当 BDP 较小时, TACK 机制回退到 Byte-counting ACK, 此时, ACK 频率随着 bw 等比例减小. 因此, TACK 机制能够最小化 ACK 的频率, 具有较强的带宽适应性.

3.1.7 不同 ACK 机制对比

表 1 总结了不同 ACK 机制的优缺点对比分析. 其中, 最简单的 ACK 机制为 Per-packet ACK, 但其面临 ACK 开销大的问题. 减少 ACK 开销主要有两类方法. 一类是收到更多数据报文后再回 ACK 报文, 如 Byte-counting ACK; 一类是经过更长的时间后再回 ACK 报文, 如 Periodic ACK.

表 1 不同 ACK 机制对比

机制	ACK频率	触发条件	优/缺点
Per-packet ACK	$\frac{bw}{PKT\_SIZE}$	收到1个数据报文	开销大、尾包丢失超时问题
Byte-counting ACK	$\frac{bw}{L\cdot MSS}$	收到L个MSS大小的数据报文	解决开销问题, 但带宽较大时适应性差、尾包丢失超时问题
Periodic ACK	$\frac{1}{\alpha}$	周期性	解决尾包丢失超时问题, 但带宽较小时适应性差
Delayed ACK	$\max\left(\frac{bw}{L\cdot MSS},\frac{1}{\alpha}\right)$ $L=2$	Byte-counting ACK和Periodic ACK两者频率较大值	带宽适应性差, 无法最小化ACK频率
Bounded ACK	$\min\left(\frac{bw}{L\cdot MSS},\frac{1}{\alpha}\right)$	Byte-counting ACK和Periodic ACK两者频率较小值	带宽适应性强, ACK频率有界
TACK	$\min\left(\frac{bw_{\max}}{L\cdot MSS},\frac{\beta}{RTT_{\min}}\right)$	Byte-counting ACK和Periodic ACK两者频率较小值	带宽适应性强, 时延适应性强, 可以最小化ACK频率

Byte-counting ACK 的 ACK 频率与带宽成正比, 因此带宽较大时 ACK 开销仍然较大 (即带宽适应性差). 同时, 由于 ACK 触发条件必须是接收到数据报文, 因此, 当发生尾包丢失时, 易导致超时问题.

Periodic ACK 不以接收数据报文为触发条件, 因此, 即使发生尾包丢失, 发送方也可以收到反馈, 从而避免超时. 然而, Periodic ACK 的 ACK 频率与带宽不相关, 当带宽较小时, ACK 频率仍然可能在较高的水平, 从而导致资源浪费 (即带宽适应性差).

Delayed ACK 综合了 Byte-counting ACK 和 Periodic ACK, 然而并没有解决带宽适应性的问题. 具体地, 当带宽较大时, Delayed ACK 采用 Byte-counting ACK 的方式, 并且规定 L=2, 在这种情况下, ACK 开销仍然较大; 当带宽较小时, Delayed ACK 回退到 Periodic ACK, 但在这种情况下, ACK 频率无法随带宽成比例减小.

如前面所述, Bounded ACK 机制和 TACK 机制相比于 Delayed ACK 机制的区别, 在于 ACK 发送频率控制逻辑正好相反 (max 和 min). Delayed ACK 机制中, 触发发送 ACK 的两个条件是“或”的关系, 而 Bounded ACK 机制和 TACK 机制使用的是“与”的逻辑. 因此, 在不同的网络条件下, Bounded ACK 机制、TACK 机制和 Delayed ACK 机制的 ACK 发送频率大小截然不同. 例如, 当带宽很大的时候, 在 ACK 数目方面, Delayed ACK 可能会比 Bounded ACK 机制和 TACK 机制高出若干个数量级[30]. 进一步地, TACK 机制相比 Bounded ACK 机制, 除了同样具有良好的带宽适应性以外, 还增加了对时延的适应性, 因此 TACK 机制能够更好地适应时延和带宽同时动态变化的场景.

3.2 主流传输协议中的确认机制

3.2.1 TCP 的确认机制

(1) TCP 报文和选项

TCP 报文格式如图 12 所示. 从上到下的源端口字段到选项区域, 属于报文头. 报文头后续部分为数据区域, 携带来自上层 (如应用层) 的数据. 下面, 以 ACK 标记位、PSH 标记位和选项区域等与确认机制密切相关的字段为例, 对 TCP 报文格式进行介绍.

图 12 TCP 报文结构

针对 ACK 标记位, 如果值为 1, 则表示该报文是一个 ACK 报文, 否则是一个数据报文. 更进一步, 如果 ACK 标记位为 1 并且数据区域不为空, 则表示处于 ACK Piggybacking 模式——如果 (数据) 接收方在接收数据的同时也有数据要发送给 (数据) 发送方, 那么可以在数据报文中带上 ACK 需要携带的反馈信息, 从而避免大量的 ACK 以一个单独的报文发送, 减少网络资源请求的次数, 降低通信开销.

发送方在发送数据时可以设置 PSH 标记位, 用于通知接收方立即进行响应. 默认情况下, 接收方的响应是将缓冲区的数据立即提交到上层应用. 然而, 如果与确认机制相结合, 则可以重新定义接收方的响应方式, 例如, 无需满足 Delayed ACK 机制下的触发条件, 就可以立即回复一个 ACK 报文. 值得一提的是, 即使不修改已有标记位的含义, TCP 报文头中的保留字段, 也为上述确认机制的扩展提供了条件.

TCP 选项区域可包含多个选项 (Option) 字段, 采用经典的 TLV (type-length-value) 结构来表示. 选项字段都以 1 字节的“Type”开头, 指明 Option 的类型, 1 字节的 Length 指明选项字段的总长度 (包括 Type, Length 和 Value 的总长度). TCP 定义了丰富的选项字段, 例如 SACK 选项、Timestamp 选项、window scale (WSCALE 或 WSopt) 选项和 maximum segment size (*MSS*) 选项等. 下面, 针对 SACK 选项的设计进行详细讨论.

(2) SACK

SACK (selective acknowledgment) 选项扩展了传统 ACK 的反馈信息量, 由 RFC 2018 定义[31]. 具体地, 它允许接收方单独确认非连续的片段, 用于告知发送方哪些报文收到了. SACK 选项可以帮助发送端确认真正丢失的报文, 从而只重传丢失的片段. SACK 选项可以减少不必要的重传, 提升传输效率.

要使用 SACK, 收发双方必须同时支持 SACK 选项. 因此双发双方需要在建立连接的时候使用 SACK Permitted 选项进行协调. 如果双方都支持 SACK, 则后续的传输过程中 ACK 报文可以携带 SACK 选项. SACK 选项由数据接收方发送, 以通知数据发送方已经接收、并且在接收方接收缓存中排队的非连续数据块 (本文称

Range). 除去最新的触发 SACK 的第 1 个 Range, 其他 Range 用历史的 Range 进行填充. 因此, SACK 机制中 ACK 构造时遵循以下 3 个原则: (a) 第 1 个 Range 需要指出是哪一个片段触发了 SACK 选项; (b) 尽可能多地把所有的 Range 填满; (c) SACK 要报告最近接收的不连续的 Range.

SACK 选项中, Type=5 表示这是选择确认 (SACK), 该字段占用一个字节. Length 表示 TCP 选项长度, 占用一个字节, 左边界和右边界各占用 4 字节, 总共用掉了 10 字节. 每新增加一个 Range, 将占用 8 字节. 如图 12 所示头部长度字段长 4 位, 所能表示的最大十进制为 15, 15×32/8 = 60, 故报文首部最大长度为 60 字节, 从而 TCP 选项字段最大长度为 40 字节. 因此, SACK 选项最多携带 4 组 Range. 实际上, TCP 默认开启一些重要的 TCP 扩展选项 (如 Timestamp 选项, 占用 10 字节), 所以 SACK 通常最多携带 3 组 Range.

由于 TCP 选项字段的长度限制, 导致 SACK 携带的 Range 数量有限, 在极端网络丢包和抖动场景下, 收益有限. 例如, 面对反向丢包时, 选择重传无法精确表达, 不必要的重传无法避免, 导致带宽浪费.

(3) D-SACK

RFC 2883 定义的 D-SACK (Duplicate SACK) 是对 SACK 选项的进一步扩展[32]. D-SACK 规定接收方收到重复数据报文时, SACK 选项的第 1 个 Range 用来传递最近接收到的重复报文的数据序号. 也就是说, 发送方根据 SACK 信息可得知哪些数据丢失了; 发送方根据 D-SACK 信息可得知哪些数据被重复接收了. 发送方根据 D-SACK 推测不必要的重传, 进而指导后续的丢包检测和重传等操作.

(4) Delayed ACK

TCP ACK, SACK, D-SACK 规定了 TCP 的确认机制中的类型和信息两大要素. 确认机制中的第 3 大要素——触发条件, 则遵循 Delayed ACK 机制. Delayed ACK 最早由 RFC 1122 提出[26], 后面在 RFC 5681 中进行了更新[27]. Delayed ACK 机制规定满足以下任意一个条件时, 接收端需要发送一个 ACK 报文: (a) 每收到 2 个的数据报文 (报文大小等于 *MSS*); (b) 每经过一个时钟周期, 且没有后续报文到达; (c) 出现乱序.

注意, 当出现乱序时, TCP 的 ACK 报文将携带 SACK 选项, 即 SACK 报文. 前面已对 Delayed ACK 进行了详细地分析和讨论, 此处不再赘述.

3.2.2 QUIC 的确认机制

(1) QUIC 报文和帧

QUIC 报文 (packet) 由一个或多个帧 (frame) 组成[33]. 报文是 QUIC 协议实现可靠传输的基本单位, 换句话说, QUIC 报文丢失后, 其包含的所有帧都需要在后续的传输过程中重传. 为了实现多路复用, QUIC 规定同一个报文中的多个帧可以来自不同的流 (stream), 且数据帧和控制帧 (如 ACK Frame) 可以包含在同一个报文中.

(2) QUIC ACK Frame

在 QUIC 中, 类型值为 0x02–0x03 的帧为 ACK Frame. QUIC 协议的确认机制基于 ACK Frame 进行实现. ACK Frame 格式如下所示[33].

```
ACK Frame {
  Type (i) = 0x02..0x03,
  Largest Acknowledged (i),
  ACK Delay (i),
  ACK Range Count (i),
  First ACK Range (i),
  ACK Range (..) …,
  [ECN Counts (..)],
}
```

其中, Type = 0x03 表示 ACK Frame 携带显式拥塞通知 (explicit congestion notification, ECN) 信息; Largest Acknowledged 表示接收方确认的最大报文序号 (在 QUIC 中称为 packet number, 在本文中称为 PKT.SEQ); ACK

Delay 表示接收方收到报文到接收方发出该 ACK Frame 所等待的时间; ACK Range Count 表示该 ACK Frame 中包含的 ACK Range 数目; ACK Range 指示已接收的报文序号范围或者未接收的报文序号范围. 其中, (i) 表示这个字段是整型, (..) 表示这个字段是可变长度的整型 (variable-length integer).

3.2.3 RTP/RTCP 的确认机制

RTP/RTCP 是实时传输协议 (real-time transport protocol, RTP) 和实时传输控制协议 (real-time control protocol, RTCP) 的总称, 广泛使用于视频通话等实时流媒体应用中[34]. RTP 工作在 UDP 之上, 其确认机制工作在应用层. RTP 为端到端的实时传输提供时间信息和流同步, 但并不保证服务质量. 服务质量由 RTCP 来提供, 因此, RTCP 协议实现了确认机制的功能. 下面, 我们将通过确认机制三要素对 RTCP 进行简要介绍.

(1) RTCP 报文的触发条件

RTCP 报文本身等价于一种特殊的 ACK 报文. RTCP 报文发送频率与会话参与者的数量成比例变小, 是一种带宽适应性的体现. RTCP 本质上属于 Bounded ACK 机制, 因此可以根据公式 (5) 计算发送频率 f_{bounded} . RFC 3550 给出了 RTCP 报文的发送频率计算中的参数设定方法. 具体地, L 的取值与数据发送者的数目和会话成员总数目相关; 根据是否发送过 RTCP 报文, α 的取值分别为 α=5 s 或者 α=2.5 s.

根据 RFC 3550 定义, 实际的 RTCP 报文发送频率 f_{rtcp} 需要在 f_{bounded} 的基础上进行一些随机化处理, 以解决多个会话参与者之间的意外同步问题. 例如, 规定 $f_{\text{rtcp}} = \lambda \cdot f_{\text{bounded}}$, 其中 λ 为服从均匀分布的系数.

(2) RTCP 报文的类型

RTCP 报文作为控制报文, 不仅可以由数据发送方发出, 还可以由数据接收方发出. 根据功能的不同, RFC3550 将 RTCP 报文的类型主要分为 SR (源报告报文)、RR (接收方报告报文)、SEDS (源描述报文)、BYE (离开申明报文) 和 APP (特殊应用报文) 这 5 类. 其中, SR (sender report) 用来使发送方以多播方式向所有接收方报告发送情况; RR (receiver report) 用于接收方向发送方报告接收情况; SEDS (source description) 用于报告和站点相关的信息, 包括 CNAME; BYE (goodbye) 用于表示会话参与者离开系统的报告; APP (application) 用于开发新应用和新特征的实验.

(3) RTCP 报文携带的信息

不同类型的 RTCP 报文, 根据其功能的不同, 携带的信息也有差异. 每个 RTCP 报文携带的信息并非音视频数据本身, 而是收发双方的统计信息. 统计信息主要包括实时数据的数据报文数目、传输过程中丢失的数据报文数目、数据报文的抖动和往返时延等.

RTCP 规范没有指定上层应用如何使用 RTCP 报文中反馈的统计信息, 因此, 如何根据这些统计信息进行相应措施, 完全由上层应用自行决定. 例如, 发送端可根据 RTCP 报文中的信息来修改视频输出码率、丢包是否重传或者进行拥塞控制, 接收端可以根据 RTCP 报文中的信息进行故障诊断, 网络管理员也可以根据 RTCP 报文中的信息来评估实时音视频传输的总体性能.

3.2.4 不同协议的确认机制对比

(1) QUIC 与 TCP 的确认机制对比

与经典 TCP 类似, QUIC 协议默认采用 Delayed ACK 机制, 但是赋予了 ACK 更多的灵活性. 具体表现在以下 3 个方面.

第一, ACK 携带的内容更多. 一方面, QUIC 协议的 ACK Frame 相比 TCP ACK 报文, 可以携带更多的 Range (超过 4 个), 从而比 TCP 具有更好的反馈鲁棒性. 另一方面, QUIC ACK 携带 ACK Delay 字段, 在 ACK 延迟发送的场景下, 能够使每个报文对应的 *RTT* 采样值更加准确, 这也是 QUIC 能够支持自定义的 ACK 频率的基本前提.

第二, 支持自定义 ACK 的发送频率. 一方面, QUIC 支持收发双方进行传输参数 (transport parameter) 协商, 其中传输参数 max_ack_delay 用于指示发送 ACK 之前最长等待的时间 (即 α). 通过在建连开始时协商 max_ack_delay, QUIC 支持每个连接采用不同的 ACK 发送频率. 另一方面, Iyengar 等人[35] 建议扩展 QUIC 协议, 新增加 ACK-FREQUENCY Frame, 使得发送方能够按需地通知接收方更新 ACK 频率, 具体而言, ACK-FREQUENCY Frame 支持发送方动态地修改公式 (4) 中的 L 和 α . 显然, 这种扩展仍然属于 Delayed ACK 的范畴.

第三, QUIC 协议的 Delayed ACK 机制运行在应用层 (用户态), 而 TCP 协议的 Delayed ACK 机制运行在传输层 (内核态). 由于用户态相比内核态天然易于调试, 因此可以预见未来 QUIC 协议上的 ACK 机制将会更有机会得到进一步的演进和优化.

(2) RTCP 与 TCP 的确认机制对比

RTCP 协议主要用于实时音视频传输应用中, 而 TCP 协议是通用传输协议, 两者应用场景的差异导致其在 ACK 报文的类型和携带的信息的设计上截然不同.

ACK 报文的发送频率方面, RTCP 协议的确认机制是 Bounded ACK 机制的具体实现. 因此, 相比 TCP 的 Delayed ACK 机制, RTCP 协议的确认机制具有更强的带宽适应性. 具体地, RTCP 协议的确认机制可以根据会话中发送方的数目进行适配, 如果发送方超过总成员的 25%, 则 ACK 占用的带宽平均分配给所有发送方和接收方; 否则把至少 25% 的 ACK 带宽分配给发送方, 其余分配给接收方.

RTCP 协议的确认机制是继 TCP 协议的确认机制之后的一种全新的尝试, 为后续的按需确认机制的提出奠定了良好的基础.

4 按需确认机制设计与评价

4.1 设计实践: 针对无线局域网的 TACK 机制

按需确认机制的实现, 依赖于对收发双方节点的修改 (通常不需要修改中间的转发节点). 即使基于用户态协议栈, 实现一个全新的确认机制也依然存在双端部署难题. 因此, 设计一个按需确认机制, 使其能够运行在通用网络中显然不太现实. 真正可行的方式, 是针对某种特定应用环境设计相应的确认机制. 下面, 以无线局域网抗干扰的应用场景为例, 介绍如何根据应用场景的需求, 设计可行的按需确认机制.

4.1.1 动机: 减少 ACK 数目

传统传输协议 TCP 因为要保证可靠, 在发送数据报文的同时, 难以避免地频繁发送 ACK 报文. 另一方面, 无线局域网 (如 Wi-Fi) 的半双工特性和冲突避免机制, 使 ACK 报文和数据报文产生直接的资源竞争, 形成“内部干扰”. 在传统 TCP 的 ACK 机制下, ACK 报文占用大量的可用频谱资源. 而且, 带宽越大, “内部干扰”越严重. 在这种情况下, 减少 ACK 报文的数目从而降低“内部干扰”, 不仅可以提高无线传输的有效带宽利用率, 而且还可以在弱网下将有限的宝贵传输资源留给数据报文.

TACK (Tame ACK) 机制[29,30] 是一种按需确认机制, 其设计目标是最小化 ACK 的数目, 从而减少内部干扰, 并且需要能够支撑其他协议功能模块实现低开销和高性能的基本目标. 如前面所述, TACK 机制的 ACK 频率 f_{tack} 可以表示为:

$$f_{tack=\min}\left(\frac{bw_{\max}}{L\cdot MSS},\frac{\beta}{RTT_{\min}}\right) \tag{7}$$

其中, $bw_{\max}$ 表示一段时间内的最大带宽, MSS 表示最大段长度, $RTT_{\min}$ 表示一段时间内的最小 RTT. L 和 β 是可调参数. $\frac{bw_{\max}}{L\cdot MSS}$ 表示每累计收到 L 个 MSS 大小的数据报文后回复 1 个 ACK 报文. β 用于控制每个 RTT 回复的 TACK 的数目, $\frac{\beta}{RTT_{\min}}$ 表示每经过 $RTT_{\min}$ 回复 β 个 ACK.

要最小化 ACK 数目, 则需要确定 β 的下界和 L 的上界. Li 等人[29] 通过理论推导和分析, 证明了当瓶颈链路的缓存至少为 1 倍 BDP 时, 有 β=2, 即 β 的下界为 2. 同时, L 的上界可以表示为 $\frac{Q}{\rho\cdot\rho'}$, 其中 Q 表示 ACK 携带的信息量, ρ 表示正向路径上的丢包率, ρ' 表示反向路径上的丢包率.

为了增加 ACK 报文的反馈鲁棒性, Li 等人[29] 通过进一步的分析, 结合实际经验, 建议取 $\beta=4$.

4.1.2 挑战: TACK 引入的副作用

(1) 丢包恢复延迟大

行头阻塞导致报文组装延迟, 从而严重影响传输性能. 应用 TACK 将进一步增大由于行头阻塞引入的报文组

装延迟. 由公式 (1) 可知, 当 $RTT_{\min}$ 较大时, TACK 报文可能会被过度延迟, 从而影响丢失检测, 导致代价昂贵的超时重传. TACK 报文的丢失将进一步加剧这个问题. 例如, 假设 $RTT_{\min}=200$ ms, $bw_{\max}=10$ Mb/s, $L=2$, 则 $f_{tack}=20$ Hz. 与经典 TCP 的确认机制相比, 发生丢包事件时, TACK 会导致反馈延迟高达 50 ms. 如果 TACK 报文丢失或重传报文再次丢失, 则这个延迟将会翻倍.

(2) 时延评估偏差

图 6 所示的用例分析已经证明, 减少 ACK 的数目将导致时延评估产生偏差. 由于 TACK 的目标是最小化 ACK 的数目, 所以两个 TACK 报文之间的间隔时间可能较长, 期间接收方可能接收到多个数据报文, 即数据报文和 ACK 是多对一的关系. 进行 *RTT* 评估时, 如果在多个数据报文中随机选择一个报文进行采样, 则可能引入评估偏差. 具体地, 最小 *RTT* 可能偏大, 最大 *RTT* 可能偏小. 一种容易想到的方案是, 在一个 TACK 报文中携带两个 TACK 报文间隔时间期间的所有收到的数据报文的相应时间戳信息, 以便于发送方能够对每个报文进行采样, 实现类似于 Per-packet ACK 机制的效果. 然而, 这种方法存在两个问题: (1) 带宽资源和内存资源等开销大; (2) 当数据报文数目较多时, ACK 报文长度有限, 可能不足以携带时延评估所必需的完整信息.

(3) 流量突发

从图 9 的用例分析可知, 当 ACK 间隔时间较长时, 流量呈现突发的特征. TACK 报文之间的间隔时间通常是比较长的, 因此一个 TACK 报文可能会确认大量的数据报文. 在这种情况下, 流量突发造成的影响将不可忽略. 如果处理不当, 则可能导致瓶颈链路缓存不足、丢包率增高和排队时延增大等问题.

(4) 接收窗口通告延迟

图 11 所示的用例分析已经证明, 当 ACK 触发时间间隔较大时, 可能造成反馈延迟, 进而导致带宽利用率低下. 如果 TACK 报文丢失, 则这一问题造成的后果将进一步恶化.

4.1.3 方案: 基于 TACK 的按需确认机制

(1) 总体策略

TACK 报文的触发频率设计, 能够最小化 ACK 的数目. 然而, 如果只是简单地应用 TACK 报文, 则可能导致传输控制效率低下, 原因是减少 ACK 数目将带来丢包恢复延迟大、时延评估偏差、流量突发和接收窗口通告延迟等副作用. 因此, 传输控制真正需要的, 是一个基于 TACK 的按需确认机制 (简称为 TACK 机制). TACK 机制的目标是在最小化 ACK 数目的基础上, 增加少量的必要的 ACK 数目, 消除上述副作用, 从而真正体现“按需确认”的理念.

具体地, 相比经典 TCP 的确认机制, TACK 机制赋予了 ACK 报文更多的智能和灵活性. 其设计原理, 可以简单地理解成 3 点: 1) 更多 ACK 种类适应不同场景需求; 2) ACK 按需携带更多必要的信息; 从而实现: 3) 更少但足够的 ACK 数目. 值得注意的是, 需要减少 ACK 数目的场景, 如无线局域网等, 在 ACK 报文中携带更多的信息, 只是增大报文长度, 并不会造成显著的频谱资源开销.

TACK 机制按照触发条件来分类, 包括两种 ACK 类型, 分别为 IACK (Instant ACK) 和 TACK (Tame ACK). 具体来说, IACK 报文加快传输控制对即时事件 (例如丢包) 的反馈和响应, 同时, TACK 报文保证反馈的鲁棒性和可靠性. IACK 基于即时事件驱动. 例如, 当接收方缓存满时, 接收方可以发送一个 IACK 报文, 通知发送方停止发送数据. 发生丢包事件时, 接收方可以发送一个 IACK 报文, 通知发送方进行重传. TACK 报文是以指定频率触发的, 其数量不会跟吞吐成比例, 也就不会造成“内部干扰”随着吞吐增大而增大.

值得注意的是, 通常情况下, IACK 报文的引入, 并不会明显地增加 ACK 报文数目. 假设正向路径上的丢包率为 ρ, 则最坏情况下的 IACK 数目为 $\rho\frac{bw_{\max}}{L\cdot MSS}$. 首先, IACK 的数目实际会远小于这个上界; 其次, 真实场景中的丢包率通常较小 (例如 $\rho<10\%$).

TACK 报文和 IACK 报文相辅相成、互相补充和高效协同, 加上基于 TACK 机制的丢包恢复、时延探测和速率控制算法, 可以保证丢包恢复的鲁棒性、时延探测的准确性以及速率控制高效性. Li 等人[29] 详细阐述了如何基于 TACK 机制设计丢包恢复算法、时延探测算法和速率控制算法. 此处不再赘述.

(2) TACK 报文携带的信息

TACK 报文除了对最新收到的数据报文进行确认以外, 还可以根据需要反馈丢包信息、带宽信息、窗口信息、

时延信息、接收缓存中接收但未提交到上层应用的数据量等. 作为扩展, 只要整个 TACK 报文不超过 *MSS*, TACK 还可以携带历史丢包信息等. 这些历史丢包信息并不是必选的, 可以按需携带. 具体地, Li 等人[29]给出了反向路径上的丢包率 ρ' 满足以下条件时, TACK 报文需要携带更多的历史丢包信息:

$$\rho' > \begin{cases} \dfrac{Q \cdot MSS}{\rho \cdot BDP}, & BDP \geqslant \beta \cdot L \cdot MSS \\ \dfrac{Q}{\rho \cdot L}, & BDP < \beta \cdot L \cdot MSS \end{cases},$$

其中, Q 表示 ACK 携带的信息量, ρ 表示正向路径上的丢包率, ρ' 表示反向路径上的丢包率. 更进一步, 当 $BDP \geqslant \beta \cdot L \cdot MSS$ 时, 需要增加的信息量为 $\dfrac{\rho \cdot \rho' \cdot BDP}{MSS} - Q$; 当 $BDP < \beta \cdot L \cdot MSS$ 时, 需要增加的信息量为 $\rho \cdot \rho' \cdot L - Q$.

(3) 丢包驱动的 IACK 报文

接收方通过丢包驱动的 IACK, 可以主动拉取丢失的数据包. 数据包丢失后会重传, 但重传包也可能会丢失. 为了在接收端能够成功地检测丢失的重传包数目, TACK 机制要求每个报文除了携带一个与 TCP 相同定义的数据序号 (用 DATA.SEQ 表示) 以外, 还需要额外引入一个严格递增的报文序号 (用 PKT.SEQ 表示), 作为每个报文的一个属性. 也就是说, 先发的报文 PKT.SEQ 小, 后发的报文 PKT.SEQ 大. 对于一个原始报文和其对应的重传报文, 两者的 DATA.SEQ 相同, 但是 PKT.SEQ 不同. 结合 PKT.SEQ 这一新的数据报文属性, 发送端在收到丢包驱动的 IACK 时, 就可以明确地知道具体哪个数据包 (包括是否是重传包) 已丢失, 从而提高丢包恢复效率. 值得注意的是, DATA.SEQ 是必要的, 因为它保证了报文的有序组装.

另一方面, PKT.SEQ 也是必要的. 下面给一个例子说明: 发送端发送 5 个数据报文, 其字节范围为 [0–5999] (*MSS*=1500 byte). 假设 PKT.SEQ = 2 的报文 [1500–2999] 丢失, 后续 PKT.SEQ = 3 的报文 [3000–4499] 到达接收方, 接收方判断丢包事件后通知发送端, 发送端重传 PKT.SEQ = 4 的报文 [1500–2999]. 假设 PKT.SEQ = 4 报文再次丢失, 当其后续 PKT.SEQ = 5 的报文 [4500–5999] 到达接收方时, 接收方仍然可以通过乱序的 PKT.SEQ, 检测丢包事件. 然而, 如果没有 PKT.SEQ, 接收端很难检测到底有多少个重传报文又丢失了.

如果接收端检测到前后两个数据报文的 PKT.SEQ 差值大于 1, 会进行丢包事件检测, 如果检测到发生了丢包事件, 就发送 IACK 报文. 该 IACK 报文携带一个 PKT.SEQ 丢失区间, 该区间指示接收缓存中最大的 PKT.SEQ 和第 2 大的 PKT.SEQ. 期间的差值就表明哪些数据报文在网络中丢失. 发送端解析此区间, 可以准确找到对应的报文并重传.

4.1.4 讨论: 滑动窗口与行头阻塞

(1) 滑动窗口机制概述

为了保证数据报文的可靠交付, 如果采用发送方每发出一个数据报文后, 必须等待收到确认该报文的 ACK 报文后开始发送下一个数据报文, 就变成了一种“停止-等待”协议. 由于上一个报文会阻塞下一个报文的发送, 这种协议通常效率低下. 因此, TCP 采用滑动窗口机制来提高数据传输效率. 滑动窗口允许发送方一次性发送不超过该窗口大小的数据, 窗口范围内的后序报文无需等待前序报文的 ACK, 这个窗口的最大值就是发送窗口 (SWND). 通常发送窗口取拥塞窗口 (CWND) 和接收窗口 (RWND) 两者的较小值. 当可发送数据的窗口消耗殆尽时, 就需要等待; 此时如果收到 ACK 确认应答后, 当前窗口又会向前滑动, 为发送下一批数据报文腾出窗口.

假设某个数据报文丢失, 迟迟未收到相应 ACK, 在该报文对应的重传没有成功之前, 滑动窗口都无法滑动, 这样就阻塞了后续所有数据报文的发送. 这就是 TCP 滑动窗口机制中的行头阻塞 (head-of-line blocking, HoLB) 问题.

(2)TACK 机制如何解决行头阻塞问题?

造成队头阻塞的根本原因, 是由滑动窗口的定义决定的. 换句话说, 队头阻塞的根本原因是经典 TCP 的滑动窗口往前滑动的必要条件是发送窗口中 DATA.SEQ 最小的报文 (即滑动窗口的左边界) 得到确认. 因此, 即使 TCP SACK 选项能够确认发送窗口中非连续的后序报文, 但这只能减少不必要的重传, 对滑动窗口的滑动也没有任何实质的帮助.

基于 TACK 机制的传输协议, 原则上可以摒弃传统的滑动窗口机制, 而替代以一种无食言 (No Reneging) 机制. No Reneging 机制是一种缓存管理机制, 它要求接收方不能将接收缓存中的已成功接收的数据报文丢弃. 那么, 为什么 TCP 要采用滑动窗口机制而不采用缓存管理机制呢? 本质的原因就在于 TCP 报文被成功接收后, 接收方是可以将不连续的数据报文丢弃的, 也就是说, TCP 是可食言的. 滑动窗口机制要求左边界的报文确认后才能滑动, 而 No Reneging 缓存管理机制只要有新的已接收数据报文得到确认, 就可以继续发送报文. 根据发送窗口的定义, 发送窗口大小被限制为不超过拥塞窗口和接收窗口的较小值. 其中, 接收窗口通过 ACK 报文由接收方反馈给发送方. No Reneging 机制下剩余可发送数据量的更新策略为: 发送窗口 – 在接收缓存中的已确认数据量 .

值得一提的是, 当原始数据包 K 丢失后, 发送方会将需要重传的数据报文放到待发送队列, 重新编号 (比如数据报文 $K+i$) 后重新发送给接收方. 由于使用了单调递增的 PKT.SEQ, 传输控制对重传报文的处理跟原始报文的处理完全相同. 因此, 基于 TACK 机制的传输协议, 在数据报文的处理逻辑方面比 TCP 相对简单.

(3) QUIC 相关机制

QUIC 协议同样采用了 No Reneging 缓存管理机制, 因此, QUIC 也采用了区别于传统滑动窗口的机制, 从而克服 HoLB 问题. 并且, QUIC 协议同样采用了与 TACK 机制类似的双 SEQ 机制. 具体地, QUIC 使用单调递增的 packet number 来记录每个报文的发送顺序, 同样根据 packet number 的乱序来检测丢包事件. TACK 机制使用 DATA.SEQ 来确认重传报文和原始报文的内容是否一致, 而在 QUIC 协议中, 使用数据流标识 (Stream ID) 作为数据报文多路复用传输到接收方后能正常组装的依据. 重传报文和原始报文单靠 Stream ID 的比对无法判断两个数据报文内容一致. 因此, QUIC 协议在每个数据报文中增加 stream offset 字段, 标识当前数据报文在当前 Stream ID 中的字节偏移量, 从而指导数据的有序组装. 换句话说, QUIC 协议中的 Packet Number 等价于 TACK 机制中的 PKT.SEQ, QUIC 协议中的 stream offset 等价于 TACK 机制中的 DATA.SEQ.

4.2 机制对比和分析

4.2.1 TACK 与 SACK

TCP 的 ACK 报文可以携带 SACK 选项, 也称 SACK 报文. SACK 选项由数据接收方发送, 以通知数据发送方已经接收、并且在接收方接收缓存中排队的非连续数据区间 (Range). 这些区间信息可以帮助发送方识别已经传输成功的数据, 避免不必要的重传. 由于 TCP 选项字段的长度限制, 导致 SACK 携带的区间数量有限, 面对反向丢包时, 选择重传无法精确表达, 不必要的重传无法避免, 导致带宽浪费.

SACK 选项字段长度受限的原因, 是因为 TCP 需要支持 ACK Piggybacking 机制, 所以 SACK 选项字段只能放在有限的报文头部. 如前面介绍的, ACK Piggybacking 能够间接地减少 ACK 报文的数目. 然而, 当应用 TACK 机制时, ACK 数目通常已经足够小. 因此, 基于 TACK 机制的传输控制, 可以考虑不支持 ACK Piggybacking 机制. 在这种情况下, 除了在报文头部可以携带区间信息以外, 还可以让 ACK 报文自带数据部分, 该数据部分并不是真正的数据, 而是用于反馈控制的 Range 信息.

4.2.2 TACK 与 HACK

为了提高 IEEE 802.11 无线链路的传输性能, Salameh 等人提出了 HACK (hierarchical ACK)[36]. HACK 通过改变 Wi-Fi 介质访问控制 (MAC), 在链路层 ACK 报文中携带 TCP ACK 原本需要反馈的信息, 去掉了 TCP ACK 对频谱资源的需求, 从而提高 TCP 的性能. 下面从 3 个方面来讨论 TACK 和 HACK 的区别.

第一, TACK 减少了端到端的 ACK 报文数目, 而 HACK 只减少了无线链路上的 ACK 报文数目. 因此, TACK 在减少 ACK 开销这方面更为通用, 可用于解决非对称网络中 ACK 路径拥塞的问题[37].

第二, HACK 要求修改网卡, 但不需要修改 TCP; TACK 要求修改 TCP, 但不需要修改网卡.

第三, 由于链路层 ACK 和传输层 ACK 的触发时间通常是异步的, 所以 HACK 很可能会导致 ACK 延迟. 因此, HACK 不能解决传输方面的挑战, 如丢包恢复延迟大、时延评估偏差、流量突发和接收窗口通告延迟等.

4.2.3 TACK 与 RACK

TACK 是为了减少 ACK 数目, 但是又不影响协议性能的一种传输协议确认机制. 确认机制需要支撑的协议功

能不仅是丢包检测, 还要其他功能比如拥塞控制和传输状态监控等. RACK (recent ACK) 于 2021 年正式成为 RFC 标准 (RFC 8985[38]). RACK 是一个丢包检测算法, 它依赖的确认机制还是经典 TCP 的基于 SACK 选项的 Delayed ACK 机制. 虽然两者命名类似, 但是, RACK 和 TACK 本质上是两个完全不同范畴的概念.

然而, 两者之间也具有一定的联系. 如果一个协议使用 TACK 机制, ACK 数量就会急剧下降, 那么这个协议不太容易实现仅发送端修改的 RACK. 原因是 ACK 报文间隔时间过长, 最新收到的报文时间戳更新缓慢, 无法支撑 RACK 及时在发送端检测丢包. 这也是 TACK 机制推荐使用基于接收端的丢包恢复策略的原因.

4.2.4 IACK 与 FACK

FACK (Forward ACK)[39], 准确来说并非某一种具体的 ACK 类型, 而是一个丢包检测和拥塞控制的综合方案. 这里主要讨论 FACK 丢包检测算法的部分.

经典 TCP 默认采用 DupACK[25], 也称为“3 次重复 ACK”检测丢包. 如图 13 所示, 其原理是“事不过三”, 即如果一个报文其后续的 3 个报文都到了, 这个报文还没有到达, 则这个报文被认为是丢失报文. 如果在 3 次重复 ACK 到达之前, 已经丢失了多个数据报文, DupACK 算法可能导致不必要的延迟.

如图 14 所示, FACK 算法规定当接收方缓存中的不连续队列长度大于 3 时, 表示发生了丢包事件. 换句话说, DupACK 算法检测丢包事件的决策条件是 (dupacks == 3), 而 FACK 算法检测丢包事件的决策条件是 ((最大序号 – 最小序号) > 3 || (dupacks == 3)). 其中, dupacks 表示重复 ACK 报文的数目, 最大序号表示接收方缓存中已接收到的报文最大数据序号, 最小序号表示发送方还没有收到确认的数据序号. 显然, 如果丢失的报文恰好只有 1 个, 则 FACK 算法和 DupACK 算法的效果是一样的. 可以认为, FACK 算法是 DupACK 算法的泛化, 用于解决一个窗口内多个报文丢包的问题.

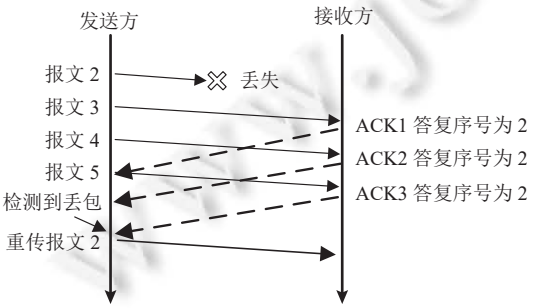


图 13 经典 TCP DupACK 丢包检测

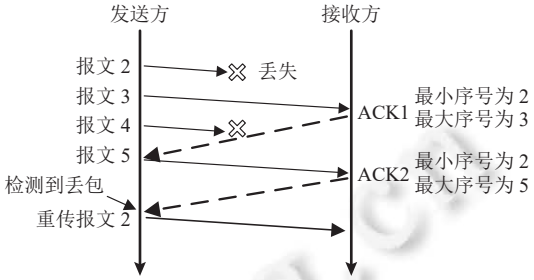


图 14 FACK 丢包检测

网络中的乱序可能是由于丢包造成的, 也有可能是由于多路径负载均衡等策略引起的. 当网络中存在大量由非丢包原因引起的乱序时, FACK 算法将会导致大量的不必要的重传. TACK 机制通过在接收方通过乱序来检测丢包事件, 为了应对非丢包原因引起的乱序, 接收方发送 IACK 之前需要等待一段时间, 称为 IACK 延迟, 类似于 RACK 算法[39]中定义的乱序窗口 (通常取 *RTT*/4). 我们将在后面的章节对 IACK 展开讨论.

4.2.5 IACK 与 RACK

前面提到, RACK 是一个丢包检测算法, 而非某一种具体的 ACK 类型.RACK 检测丢包的原理是, 当发送方认为某个报文缺失的时间足够久后, 该报文被认为是丢失报文. 这与经典 TCP 采用 DupACK 进行丢包检测的方式具有本质区别.

在尾包丢失和重传报文丢失的情况下, RACK 相比 DupACK 具有显著优势. 以重传报文丢失为例. DupACK 算法基于计数进行丢包检测, 在重传报文丢失后, 由于发送方无法准确区分重传报文与原始报文, 发送方无法明确地检测到重传报文丢失事件, 因此可能导致开销较大的超时重传. 而 RACK 基于计时进行丢包检测, 由于时间是一个严格递增的量, 因此可以根据时间戳的不同 (如图 15 所示的 t_1 和 t_3) 来区分重传报文与原始报文. 如图 15 所示, RACK 可以准确地识别重传报文的丢失情况.

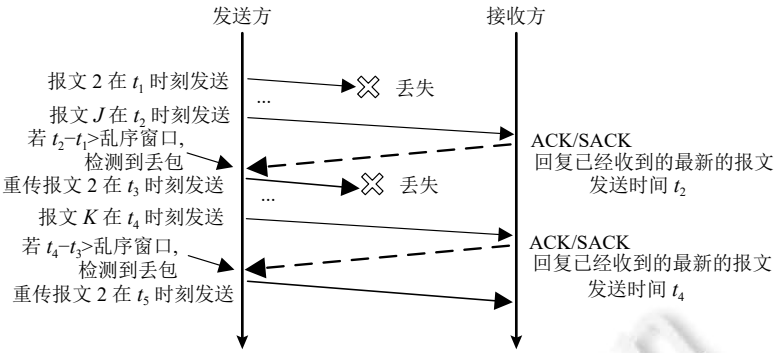


图 15 RACK 丢包检测

在 TACK 机制中, 丢包检测在接收方完成. 具体地, 接收方根据报文 PKT.SEQ 的乱序情况检测丢包事件, 通过丢包驱动的 IACK, 反馈丢包信息到发送方. 基于 IACK 的丢包检测算法, 其本质上就是接收端版本的 RACK 算法. 与 RACK 的区别, 体现在 RACK 使用 (严格递增的) 时间戳, 而 TACK 机制使用一个严格递增的报文序号 (PKT.SEQ), 本质上时间戳和 PKT.SEQ 具有相同的作用. IACK 基于报文序号 PKT.SEQ 进行丢包检测, 由于 PKT.SEQ 是一个严格递增的量, 因此可以根据 PKT.SEQ 的不同 (如图 16 所示的 PKT.SEQ=2 和 PKT.SEQ=5) 来区分重传报文与原始报文. 如图 16 所示, IACK 也可以准确地识别重传报文的丢失情况.

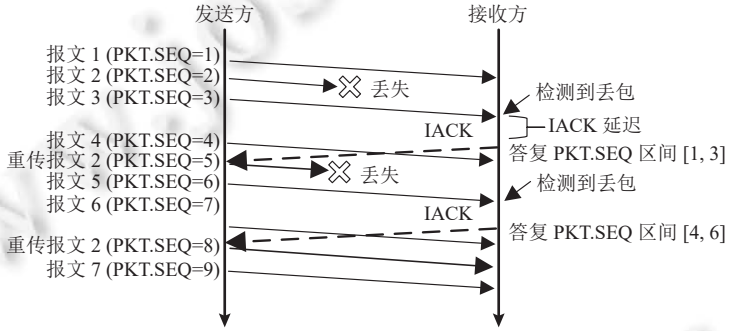


图 16 IACK 丢包检测

值得一提的是, 如果 TCP 不开启时间戳选项, 当发送方触发重传数据报文后, 接收方即使收到了该数据报文, 发送方也无法判断确认接收的数据报文是来自原始请求的响应, 还是来自重传请求的响应, 这就带来了重传二义性 (retransmission ambiguity). 重传二义性影响采样 *RTT* 测量值的准确性, 进而影响重发超时阈值计算的准确度, 可能会增大数据报文超时重传的概率. 时间戳和报文序号都能将重复报文和原始报文区分出来, 因此, 两者都能够提高 *RTT* 测量的准确性.

4.2.6 IACK 与 NACK

不论是否携带 SACK 选项, TCP 的 ACK 均是一种“正”反馈, 即反馈哪些报文成功接收. 容易想到, 还存在与其相反的“负”反馈, 称为 NACK/NAK (Negative ACK). ACK 告诉发送方哪些报文收到了, 而 NACK 告诉发送方哪些报文丢失了. 传输同样多的数据报文, NACK 的数量通常少于 ACK 的数量, 然而采用 NACK 意味着发送方必须被动地等待丢包信息, 没有丢包信息则表明报文成功接收, 但需等待多久则无法明确表示. NACK 作为一种错误控制机制, 已经被广泛地应用在 WebRTC[40], UDT[41], RBUDP[42], NACK Option[43]和 NORM[44]等传输解决方案.

丢包驱动的 IACK 与 NACK 具有相同的思想. 准确地说, IACK 是一个比 NACK 更一般化的概念, 它的触发条件是发生即时事件. 这些即时事件不仅可以是丢包事件, 还可以是缓冲区耗尽事件、*RTT* 更新请求事件等. 因此, 可以认为 IACK 是 NACK 的扩展.

4.2.7 不同机制和算法对比

业界往往容易混淆 ACK 相关的机制和算法. ACK 机制是指定义了 ACK 类型、各类型对应的触发条件、各类型对应携带的信息的一个方法的总称; 而 ACK 相关的算法, 例如 RACK 和 FACK, 这些并不是 ACK 机制, 而是基于某种 ACK 机制的丢包检测算法. 因此, 我们在表 2 中对比了上述的 ACK 相关机制和算法, 以便更明确地阐述他们之间的区别和联系.

表 2 ACK 相关机制和算法对比

机制/算法	类别	ACK频率	携带的信息
TCP ACK	ACK报文	$\frac{bw}{L\cdot MSS}$	后续需要的报文信息ACK number等. 参考TCP标准报文头部字段定义
SACK	ACK报文	乱序事件触发	已接收的报文信息. 除标准报文头外, 选项区域携带不超过4个Range
HACK	ACK机制	随MAC层ACK发送	传输层ACK不携带信息
RACK	丢包检测算法	—	SACK选项
FACK	丢包检测算法	—	SACK选项
NACK	ACK报文	乱序事件触发	丢包信息
TACK	ACK机制/ACK报文	$\min\left(\frac{bw_{\max}}{L\cdot MSS},\frac{\beta}{RTT_{\min}}\right)$	带宽信息、时延信息、丢包信息等
IACK	ACK报文	乱序等即时事件触发	丢包信息

4.3 TACK 机制的实现方案

TACK 机制在协议栈中的可以有如图 17 所示的 4 种形态: 依次是基于内核 UDP 的用户态实现、基于内核 TCP 的改进、基于用户态 IP 的 TCP 实现和基于用户态协议 QUIC 的实现.

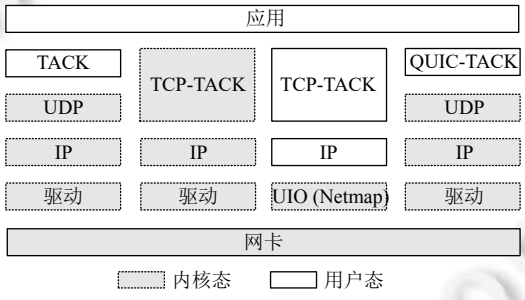


图 17 TACK 机制在协议栈中的实现方式

4.3.1 基于 UDP 的实现

基于 UDP 的 TACK 机制实现, 就是基于 TACK 机制实现可靠的、有连接、有序的传输控制. 下面从报文格式和接口设计两个角度来举例说明具体的实现要点.

报文主要分为数据报文和 ACK 报文, ACK 报文则主要分为 IACK 报文和 TACK 报文两种. 数据报文的格式的设计可参考如图 18 所示. 首先, 不论是数据报文还是 ACK 报文, 都应该携带一个自定义的、公共的、基于 TACK 的协议报文头, 其中类型字段指示报文的类型, 长度字段指示报文长度, 其他字段可根据报文类型灵活定义. 数据报文中, 由于要保证有序的数据组装, 因此数据报文需携带一个类似于 TCP 协议中定义的数据序号 (DATA.SEQ). 数据序号只与数据的内容相关, 因此重传报文与其对应的原始报文的数据序号是相同的. 同时, TACK 机制要求数据报文携带一个报文序号 (PKT.SEQ), 报文序号只与报文的发送顺序相关, 与数据的内容无关, 因此重传报文与其对应的原始报文的报文序号是不同的. 图 18 中给出了这种双序号设计的样例.

基于 UDP 实现的 IACK 报文格式如图 19 所示. 如果接收方检测到前后两个数据报文的 PKT.SEQ 差值大于

1, 接收方进行丢包事件检测, 如果检测到发生了丢包事件, 就发送 IACK. 该 IACK 可以携带报文序号 (PKT.SEQ) 丢失区间, 该区间指示接收缓存中最大的 PKT.SEQ 和次大 (第 2 大) 的 PKT.SEQ, 最大的 PKT.SEQ 和次大的 PKT.SEQ 两者的差值可以指示哪些数据报文发生丢失. 发送方解析最大的 PKT.SEQ 和次大的 PKT.SEQ, 可以准确找到对应的报文并重传.

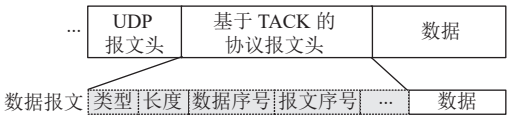


图 18 基于 UDP 实现的数据报文格式

图 19 基于 UDP 实现的 IACK 报文格式

基于 UDP 实现的 TACK 报文格式如图 20 所示. 对于一个 TACK 报文, 其报文头可参照经典 TCP ACK 报文包含一个确认号 SEQ, 指示当前收到的连续的数据序号的最大值, 含义是告知发送方该数据序号之前的数据报文都已经接收到, 发送方可以释放这些数据占用的缓存. TACK 报文相比 TCP ACK 报文, 将携带更多的反馈信息. 例如, 丢包信息, 包含丢包率和 HOL.SEQ 字段.HOL.SEQ 可指示接收缓存中最小的数据序号. TACK 报文头中的确认号 SEQ 和丢包信息中的 HOL.SEQ 字段, 共同指示了接收缓存中的第一个接收缓存空洞 (receive-buffer-bubble, RBB), 即最早丢失的数据报文的字节范围 (byte range).

TACK 除了指示丢包信息以外, 其报文数据部分还可以指示带宽信息 (如 delivery rate)、窗口信息 (如 congestion window)、时延信息 (如 ACK Delay) 等. 作为扩展, 只要整个 TACK 报文不超过 *MSS*, TACK 还可以进行扩展, 携带更多额外 RBB (Additional RBB) 用来指示更多的历史丢包信息等, 每个 Additional RBB 都由左边界 (left edge of additional RBB) 和右边界 (right edge of additional RBB) 共同表示.

基于 UDP 实现 TACK 机制, 可以参考经典 TCP 协议, 提供类 POSIX 套接字接口. 采用客户端-服务端模式, 则接口定义及其调用流程可参考如图 21 所示.

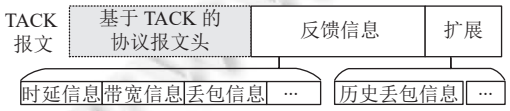


图 20 基于 UDP 实现的 TACK 报文格式

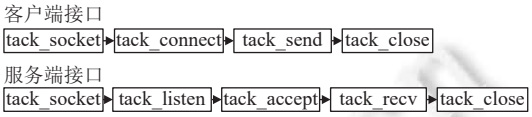


图 21 基于 UDP 实现的类 POSIX 套接字接口

4.3.2 基于 TCP 的实现

基于 TCP 实现 TACK 机制, 如图 17 所示的两种基本的形态: 基于内核 TCP 的改进和基于用户态 IP 的 TCP 实现. 其中, 用户态 IP 的 TCP 实现基于 Netmap 或者 DPDK 等 UIO 实现与网卡之间的数据交互. 不论采用哪种形态, 基于 TCP 实现 TACK 机制需要的协议修改基本相同.

经典 TCP 通常使用报文头中的 40 字节选项 (Option) 字段区域对 TCP 进行扩展, 例如 Timestamp 选项、SACK 选项和 *MSS* 选项等. 理论上, TCP-TACK 可利用 TCP 选项字段定义更多的 ACK 类型, 扩展报文以携带更多的反馈信息. 然而, 由于选项字段区域限制为 40 字节, TACK 报文可能无法使用这些有限的长度, 来携带传输所需要的所有反馈信息. 例如, 在丢包严重的情况下, 基于选项进行扩展的 TACK 报文无法携带超过 4 个 RBB.

在这种情况下, TCP-TACK 可通过扩展 TCP 选项区域定义更多 ACK 类型, 然后通过扩展 TCP 的数据区域 (data field) 以使 ACK 可携带更多的反馈信息. 这种设计来源于以下考虑: (1) TCP 选项的扩展可以充分地使用报文头部剩余空间; (2) 数据区域的扩展, 只增加 ACK 报文大小, 不会增加报文数目, 不会违背 TACK 机制的设计初衷; (3) 如前面所述, 基于 TACK 机制的传输控制, 可以不支持 ACK Piggybacking 机制. 扩展 ACK 报文的数据区域以用于反馈控制信息具有可行性.

具体的 TCP 选项扩展和 TCP 数据区域扩展方案, 可以参考 Li 等人在文献 [30] 中示例. 此处不再赘述. 针对 TCP 进行选项字段扩展, 其报文可能无法通过网络路径中广泛存在的中间盒 (middlebox) 的过滤, 即某些中间盒可

能会将无法识别的选项字段丢弃或者二次处理. 因此, 基于 TCP 实现 TACK 机制, 其应用场景受到极大限制. TCP-TACK 通常只能用于相对封闭的场景中, 例如无线局域网. 如果要实现广泛的部署, 则需要设计一些逃生机制, 以适应 TCP 扩展字段被修改或者移除的场景[30].

4.3.3 基于 QUIC 的实现

QUIC 是一个灵活的传输协议框架, 使用 UDP 作为底层抽象, 避免修改内核操作系统; QUIC 报文头以加密形式进行传输, 避免受到中间盒的修改和移除[2]. 因此, TACK 机制在 QUIC 中的实现, 本质上是一种特殊的基于 UDP 的实现, 区别在于 QUIC-TACK 的实现可以最大程度地复用现有 QUIC 框架的原始设计, 既可减少开发工作量, 又可提升兼容性.

一个 QUIC 报文包含一个或者多个帧 (frame). QUIC 收发双方通常通过传输参数 (transport parameter) 协商各自支持的版本和特性. 由于 TACK 机制是一种 QUIC 框架的扩展, 因此, 可以在 QUIC 传输参数中增加一个标记, 例如 tack-support. 当某个通信节点设置 tack-support 参数时, 表明其支持 TACK 机制, 如果对端不支持 TACK 机制, QUIC-TACK 需要回退到 QUIC 默认的确认机制.

当收发双方都支持 TACK 机制时, QUIC-TACK 将生成 TACK 帧和 IACK 帧, 用于收发双方之间的信息同步和反馈; 同时, 根据网络条件和传输状态, 发送方按需生成 ACK-FREQUENCY 帧, 用于指示接收方动态调整 TACK 发送频率. 具体的 TACK 帧、IACK 帧和 ACK-FREQUENCY 帧的设计, 可以参考 Li 等人在文献 [24] 中的示例. 此处不再赘述.

4.3.4 不同实现方案对比

由于协议底座的差异, 不同的 TACK 机制的实现方案之间在开发难度、适用范围和运行开销等方面差别较大. 如表 3 所示, 由于用户态协议栈相比内核态协议栈具有更大的灵活性, 方便调试和修改, 因此基于内核 TCP 的实现方案开发难度最大; 在 QUIC 协议框架中, TACK 机制的实现可以最大程度地复用如 packet number (对应本文中的 PKT.SEQ)、ACK Gap (对应本文中的 RBB) 等机制, 加上 QUIC 框架本身是用户态实现方式, 因此, 基于 QUIC 的实现方案开发难度最低. 适用范围方面, 根据部署和修改难度, 以及中间盒的适应性, 可以推断基于 QUIC 的实现方案适应各种应用场景的能力最强, 基于 UDP 的实现次之, 基于内核 TCP 的实现最小. 运行开销方面, 尤其内核 TCP 实现了 TSO/GRO/LRO 等优化, 在带宽相同的情况下, 基于内核 TCP 的实现方案运行开销最小; 由于 QUIC 除了实现基本的传输控制以外, 还引入了多路复用和 stream 等机制, 框架的复杂性不可避免地引入了一些额外开销, 因此基于 QUIC 的实现方案运行开销最大.

表 3 TACK 机制实现方案对比

实现方案	开发难度	适用范围	运行开销
基于UDP	小	中	大
基于内核TCP	大	较小	较小
基于用户态TCP	中	小	小
基于QUIC	较小	大	较大

4.4 TACK 机制的实验评估结果

4.4.1 TACK 机制的定量分析

为了简单起见, 便于理解, 我们假设 L=1, 同时假设每个数据报文的大小都等于最大段长 (MSS). 根据 ACK 频率公式 (4) 和公式 (6), TCP 的 Delayed ACK 机制和 TACK 机制的对比如图 22 所示. 图中的拐点指示了 TACK 机制相比传统 TCP 产生收益的边界条件. 从图中容易得到以下 3 个基本结论.

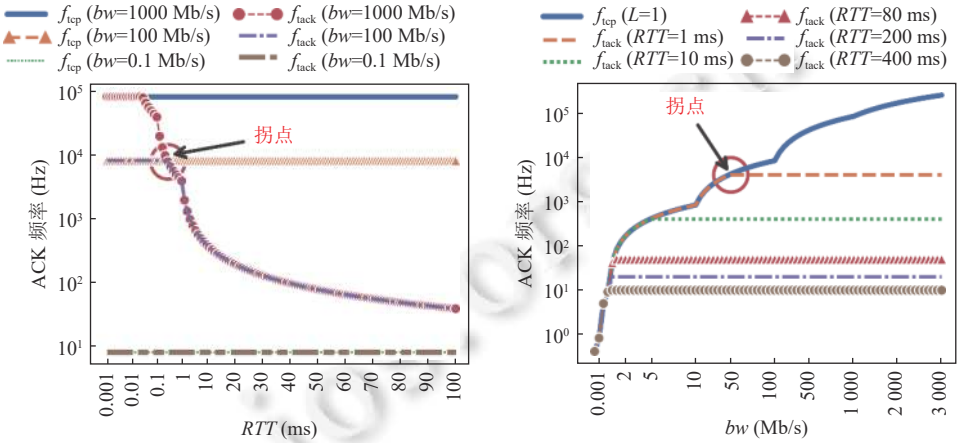
首先, 给定 L 的值, TACK 机制的 ACK 频率总是不超过 TCP 的 Delayed ACK 机制的 ACK 频率, 即 $f_{tack} \leqslant f_{tcp}$. 例如, 当 bw = 48 Mb/s, RTT = 10 ms 时, TACK 机制的 ACK 频率是 TCP 的 1/10.

其次, 吞吐量越大, 应用 TACK 机制减少的 ACK 数目越多, 收益越大. 例如, 如图 22(a) 所示, 当 RTT = 10 ms

时, 如果吞吐量从 48 Mb/s 增长到 200 Mb/s, 则 TACK 机制的 ACK 频率将降低两个数量级 ($f_{tack}=2.4\%\cdot f_{tcp}$), 而 TCP 仅降低了一个数量级. 同时, 如图 22(a) 所示, 吞吐量越大, 拐点对应的 *RTT* 边界值越小.

最后, 往返时延越大, 应用 TACK 机制减少的 ACK 数目越多, 收益越大. 例如, 如图 22(b) 所示, 当 $bw=200$ Mb/s 时, 如果往返时延从 10 ms 增长到 80 ms, 则 TACK 机制的 ACK 频率将降低 3 个数量级 ($f_{tack}=0.3\%\cdot f_{tcp}$), 而 TCP 保持不变. 同时, 如图 22(b) 所示, 往返时延越大, 拐点对应的 *bw* 边界值越小.

总结起来, 在大部分情况下, TACK 机制可以显著地减少 ACK 报文的数目. 同时, 容易证明, 如果数据报文小于 *MSS*, 也可以得到相似的结论.



(a) ACK 频率与往返时延的关系　　(b) ACK 频率与吞吐量的关系

图 22　ACK 频率与往返时延、吞吐量之间的关系

4.4.2　TACK 机制在实际网络中的运行结果

(1) 实验设置

实验运行在基于 IEEE 802.11 b/g/n/ac 的无线链路上, 这些链路的物理速率分别为 11/54/300/866.7 Mb/s. 实验对比对象为应用了 TACK 机制的 TCP 实现, 即 TCP-TACK, 以及采用 Delayed ACK 机制的传统的 TCP. 其中, $L=2$. 实验过程中, 一台 Wi-Fi 主机 (Intel Wireless-AC 8260, 2 × 2) 通过一台无线路由器 (TLWDR7500) 连接另一台有线的主机. 所有设备均放置在一个公共办公室环境中, 经检测, 峰值情况下, 环境中存在超过 10 个其他的 Wi-Fi 路由器和超过 100 个无线终端用户. 通过 Ping 进行测试, 发现 *RTT* 在 4 ms 到 200 ms 之间波动, 且存在少量的突发丢包. 收发双方之间进行打流测试, 测试重复执行了 7×24 h.

(2) 实验结果

在给出实际的性能结果之前, 我们先进行理论数值分析. 如图 23(a) 所示, 针对 IEEE 802.11 b 链路, 当 *RTT* 较小 (*RTT*=10 ms) 时, TACK 与 TCP 具有相同的 ACK 报文数目. 然而, 针对 IEEE 802.11 ac 链路, 当 *RTT* 较小 (*RTT*=10 ms) 时, TACK 的 ACK 报文数目相比 TCP 降低了两个数量级, 当 *RTT* 较大 (*RTT*=80 ms) 时, TACK 的 ACK 报文数目相比 TCP 降低了 3 个数量级. 考虑到当前实际网络很少用到 IEEE 802.11 b 链路, 大部分实际网络的带宽都要高于 IEEE 802.11 b 链路的带宽, 因此, TACK 机制在实际的网络中能够显著地减少 ACK 的数目, 从而有潜力提升网络传输的性能.

如图 23(b) 所示, 在实际的基于 IEEE 802.11 b/g/n/ac 链路的网络中, TCP-TACK 相比传统 TCP 减少了 90% 以上的 ACK 数目, 提升了 20%–28% 的吞吐量. TCP-TACK 获得有益效果的原因, 来源于 TCP-TACK 减少了 ACK 报文的数目, 从而减少了 ACK 报文与数据报文之间的频谱资源竞争, 从而提升了可用带宽.

针对前面讨论的 ACK 机制面临的问题和挑战, 我们进一步探究 TACK 机制在时延抖动和反向丢包场景下的性能. 图 24(a) 给出了时延抖动下的 TACK 机制性能举例. 在这个实验中, 实际最小 *RTT* 为 100 ms, 蓝色虚线表示

基于 TACK 机制测量得到的 *RTT* 的测量值, 橙色虚线表示基于 TACK 机制使用传统 TCP 的在发送端进行采样的最小时延探测算法, 绿色实线表示基于 TACK 机制使用了基于单向时延的最小时延探测算法 (算法详情见文献 [29]). 实验表明, 在应用 TACK 机制的情况下, 如果继续采用传统时延探测算法, 将导致较大的偏差 (8–18 ms), 而如果经过重新精心地设计相应的算法, 则可以避免因应用 TACK 引入的副作用.

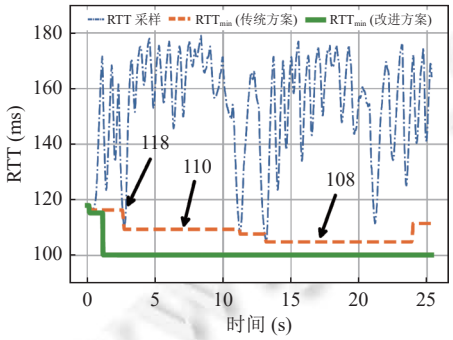
	802.11b		802.11ac	
RTT_{min}	TCP (L=2)	TACK (L=2)	TCP (L=2)	TACK (L=2)
10 ms	294	294	24 777	400
80 ms	294	50	24 777	50
200 ms	294	20	24 777	20

(a) 实际网络中 ACK 报文数目举例[28]

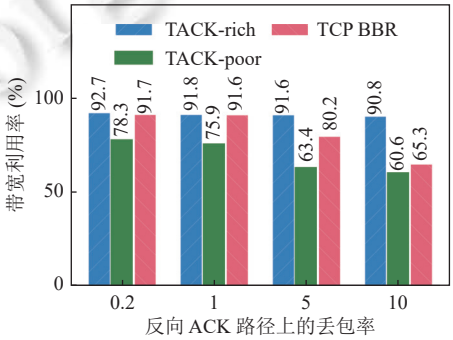
	802.11b	802.11g	802.11n	802.11ac
减少的 ACK 数目	90.5%	95.4%	99.4%	99.8%
提升的吞吐量	20.0%	26.3%	27.7%	28.1%

(b) 实际网络中 TACK 机制的有益效果[28]

图 23 TACK 机制在实际网络中的运行结果



(a) 时延抖动下的 TACK 机制性能[28]



(b) 反向丢包下的 TACK 机制性能[28]

图 24 TACK 机制在时延抖动和反射丢包场景下的性能举例

图 24(b) 给出了反向丢包下的 TACK 机制性能举例. 在这个实验中, *RTT* 固定为 200 ms, 正向数据路径上的丢包率为 1%, 反向 ACK 路径上的丢包率在 0.2%–10% 之间变化. “TACK-poor”表示 TACK 报文仅携带 1 个 RBB, “TACK-rich”表示 TACK 报文携带尽量多的 RBB. 实验表明, 在应用 TACK 机制的情况下, 如果仅携带 1 个 RBB, 反向丢包将导致协议性能恶化, 只有携带更多的信息量, 如“TACK-rich”才可以避免因应用 TACK 引入的副作用. 综上所述, 针对时延抖动和反向丢包场景的实验结果, 进一步证明了频繁的 ACK 数目并不是传输所必须的, 如果对当前传统的传输协议机制 (如时延探测和丢包恢复) 进行针对性地调整和修改, 则可以避免因减少 ACK 带来的副作用.

5 未来研究方向

确认机制是传输控制中的关键模块, 针对确认机制的研究对协议性能优化起到关键作用. 但是, 相比拥塞控制算法, 确认机制与传输控制中的其他模块耦合性更强. 这主要体现在: (1) 确认机制优化方案通常需要收发双端配合. 例如, 新增一种 ACK 类型, 需要数据接收方按 ACK 报文指定条件进行触发和封装携带指定的信息; 同时, 需要数据发送方能够识别和解析该 ACK 类型携带的内容, 并正确进行传输控制操作 (例如增减窗口). (2) 确认机制的变更, 通常需要相应地修改整个协议栈. 例如, 减少 ACK 的数目, 需要修改拥塞控制算法, 将部分原来在发送方完成的逻辑 (例如统计每包时延), 迁移到接收方来完成[30]. (3) 不同的确认机制之间差异较大, 协议强耦合性导致很难在一个统一的协议框架中对不同的机制进行对比. (4) 不存在一种通用的确认机制在所有场景和应用条件下都能达到最优, 本文中描述的 TACK 机制只是按需确认的在无线局域网中的实例, 然而, 针对更多的场景和差异

化应用, 还需要寻求对应的最优的确认机制.

因此, 传输控制中的确认机制值得进一步广泛深入的研究, 未来研究方向可以从以下几个方面展开.

(1) 确认机制的兼容性

不同的网络环境, 可能需要不同的确认机制; 同一个连接内, 不同的时刻也可能需要不同的确认机制. 因此, 需要设计协商机制, 使数据收发双方之间实现兼容. 在经典 TCP 中, 支持通过选项区域扩展, 实现确认机制的协商. 例如, 可以在 TCP 选项区域新增加 TACK-Permitted 选项字段. 在建连过程中, TCP 的 SYN 报文携带 TACK-Permitted 选项用于指明 SYN 报文的发送方支持 TACK 机制[30]. 在 QUIC 中, 通过传输参数在收发双方之间进行参数的协商. 因此, 可以复用传输参数在收发双方之间进行 ACK 机制的协商. 具体地, 定义一个新的传输参数字段 tack-support. 发送方在帧中携带 tack-support, 表明发送方支持 TACK 机制, 接收方接收到这个携带 tack-support 字段的帧后, 如果支持, 则在后续的传输过程中, 采用 TACK 机制.

(2) 确认机制的模块化

根据确认机制“类型-触发条件-信息”三要素, 确认机制的模块化必须实现 ACK 类型识别、ACK 频率更新和 ACK 信息解析等功能的模块化. 在经典 TCP 中, 支持通过选项区域扩展指示 ACK 类型. 例如, 可以在 TCP 选项区域新增加 ACK-Type 选项字段. 当 ACK-Type 值为 0x01 时表示 TACK 报文, 0x02–0xff 表示不同类型的 IACK 报文[30]. 相比 TCP, 在 QUIC 中进行扩展更加容易, 因为 QUIC 天然地支持通过帧的类型直接指示不同的 ACK 类型. 同时, Iyengar 等人[35]提出在 QUIC 中新增 ACK-FREQUENCY 帧类型. 数据发送方可以通过向接收方发送 ACK-FREQUENCY 帧, 通知接收方更新 ACK 频率. 相比 ACK 类型识别和 ACK 频率更新, ACK 信息解析的模块化工作目前进展缓慢, 有待进一步地研究和完善.

(3) 统一的确认机制测试平台

不同的确认机制对传输控制的影响各不相同, 给定一种网络环境和应用需求, 要评价哪种机制最优, 则需要将不同的机制放在一起对比测试. 另一方面, 按需确认追求发送的每个 ACK 报文都正好是传输控制所必需的, 如何评价每个 ACK 报文产生的作用, 也需要设计对应的评判标准. 当前, 业界存在一些传输协议测试平台, 如 Pantheon[45]. 然而, 这些平台仅支持将整个传输协议作为测试对象, 无法通过控制变量的方式将确认机制作为独立的测试对象进行评估. 这种情况下, 传输协议中除确认机制以外的其他所有模块可能不同, 违背了控制变量法, 从而引入评估偏差. 因此, 一个新的研究方向, 是开发一套统一的确认机制测试平台, 研发人员只需要提供修改确认机制三要素中的部分或者全部算法, 则可以进行统一评估, 综合对比各种机制在不同参数下的性能, 从而为不同的网络条件和应用需求寻求最优的确认机制提供依据.

(4) 面向 QoE 的确认机制

当前针对确认机制的研究和设计, 都是建立在网络数据传输的速率、时延、丢包等基本的性能指标上的. 然而, 用户通常愿意为蓝光、4K、VR 体验买单, 但不一定愿意为 10 Mb/s、20 Mb/s 和 100 Mb/s 的数字买单. 用户体验 (QoE) 是主观感受, 并不能简单地用传输速率来评价, 但用户体验也一定是可定义、可衡量、可管理的. 因此, 针对 QoE 设计按需的确认机制, 是未来的一个重要研究方向.

(5) 基于 AI 的确认机制

人工智能 (AI) 具有强大的自适应性和自学习能力, 可以为一些复杂网络条件情况下的决策提供解决方案, 近年来引起了业界广泛关注. 当前 AI 和传输控制相结合, 主要集中在拥塞控制算法层面, 如 Remy[20] 、Indigo[21] 、PCC[46] 和 PCC Vivace[47] 等. 然而, 基于 AI 的确认机制方面的研究则相对较少. 未来的一个研究方向, 可以是将 AI 和确认机制的设计相结合, 需要根据网络条件的动态变化, 自适应地在 ACK 类型、ACK 频率和 ACK 携带的信息等层面进行实时决策, 从而有可能真正意义上地实现传输控制过程中的按需确认.

6 结 论

数据爆炸性增长, 新型应用持续涌现, 分布式系统之间的数据传输控制, 对数据中心网络、广域网和无线局域网等提出了更高的差异化需求, 同时, 多种网络特征的差异性也为设计专用的、高效的传输控制确认机制, 提供了

广阔的发挥空间. 例如, 数据中心网络中超高带宽和超低时延, 要求确认机制能够实现及时反馈, 贸然减少 ACK 数目可能导致性能恶化; 而无线局域网中频谱作为宝贵资源, 应该尽量让给数据报文的传输, 因此减少 ACK 数目将显著提升性能. 另一方面, 数据中心网络和无线局域网, 相比广域网而言, 通常同时支持端节点和中间交换节点的定制化, 这种灵活性为确认机制的设计提供了更多的可能性.

另一方面, 内核协议栈最大的问题在于它的更新迭代太慢, 与操作系统绑定, 调试和问题定位非常复杂, 无法适应网络条件动态变化、新型业务需求多样化的现状. 而用户态协议栈很好地解决了这个问题. 当前最热的用户态协议框架为 QUIC 协议. 各个大厂都在积极布局, 例如, Microsoft 的 MsQuic[48] , Facebook 的 mvfst[49] , 华为的 hQUIC[50], 阿里的 XLINK[51] 等, 可以说是百花齐放, 百家争鸣的状态, QUIC 作为 HTTP 3.0 的协议底座, 未来也拥有无限可能. 可以预见, 依托用户态协议栈这个“罗马广场”, 针对不同的网络环境和差异化的应用需求, 按需确认机制的设计理念, 将有力推动下一代传输协议的发展.

References:

[1] Jacobson V. Congestion avoidance and control. ACM SIGCOMM Computer Communication Review, 1988, 18(4): 314–329. [doi: 10.1145/52325.52356]

[2] Luo WM, Lin C, Yan BP. A survey of congestion control in the Internet. Chinese Journal of Computers, 2001, 24(1): 1–18 (in Chinese with English abstract). [doi: 10.3321/j.issn:0254-4164.2001.01.001]

[3] Wu H, Yu ZH, Cheng G, Hu XY. Encrypted video recognition in large-scale fingerprint database. Ruan Jian Xue Bao/Journal of Software, 2021, 32(10): 3310–3330 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6025.htm [doi: 10.13328/j.cnki.jos.006025]

[4] Wang JX, Gong H, Chen JE. A cooperant congestion control protocol in high bandwidth-delay product networks. Ruan Jian Xue Bao/Journal of Software, 2008, 19(1): 125–135 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/125.htm [doi: 10.3724/SP.J.1001.2008.00125]

[5] Xu CB, Xian YJ, Tang CW, Yang SZ. An active congestion control mechanism for transmission control protocol. Ruan Jian Xue Bao/Journal of Software, 2008, 19(6): 1533–1545 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/1533.htm [doi: 10.3724/SP.J.1001.2008.01533]

[6] Zhang YM, Xu WQ, Huang J, Wang YM, Shu T, Liu LG. Optimal cross-layer power control and congestion control providing energy saving for ad hoc networks. Ruan Jian Xue Bao/Journal of Software, 2013, 24(4): 900–914 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4317.htm [doi: 10.3724/SP.J.1001.2013.04317]

[7] Ren FY, Lin C, Liu WD. Congestion control in IP network. Chinese Journal of Computers, 2003, 26(9): 1025–1034 (in Chinese with English abstract). [doi: 10.3321/j.issn:0254-4164.2003.09.001]

[8] Du XL, Xu K, Li T, Zheng K, Fu ST, Shen M. Traffic control for data center network: State of the art and future research. Chinese Journal of Computers, 2021, 44(7): 1287–1309 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2021.01287]

[9] Li D, Chen GH, Ren FY, Jiang CL, Xu MW. Data center network research progress and trends. Chinese Journal of Computers, 2014, 37(2): 259–274 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2014.00259]

[10] Xu K, Zhu M, Lin C. Internet architecture evaluation models, mechanisms and methods. Chinese Journal of Computers, 2012, 35(10): 1985–2006 [doi: 10.3724/SP.J.1016.2012.01985]

[11] Wan K, Luo XF, Jiang Y, Xu K. The flow-oriented scheduling algorithms in SDN system. Chinese Journal of Computers, 2016, 39(6): 1208–1223 (in Chinese with English abstract). [doi: 10.11897/SP.J.1016.2016.01208]

[12] Zhang YC, Xu K, Wang HY, Li Q, Li T, Cao X. Going fast and fair: Latency optimization for cloud-based service chains. IEEE Network, 2018, 32(2): 138–143. [doi: 10.1109/mnet.2017.1700275]

[13] Li T, Liang J, Ding Y, *et al*. On design and performance of offline finding network. In: Proc. of the 2023 IEEE INFOCOM. 2023. 1–10.

[14] Floyd S, Kohler E. RFC 4341: Profile for datagram congestion control protocol (DCCP) Congestion Control ID 2: TCP-like Congestion Control. 2006. https://www.rfc-editor.org/rfc/rfc4341.html

[15] Langley A, Riddoch A, Wilk A, Vicente A, Krasic C, Zhang D, Yang F, Kouranov F, Swett I, Iyengar J, Bailey J, Dorfman J, Roskind J, Kulik J, Westin P, Tenneti R, Shade R, Hamilton R, Vasiliev V, Chang WT, Shi ZY. The QUIC transport protocol: Design and Internet-scale deployment. In: Proc. of the 2017 Conf. of the ACM Special Interest Group on Data Communication. Los Angeles: ACM, 2017. 183–196. [doi: 10.1145/3098822.3098842]

[16] Palmer M, Appel M, Spiteri K, Chandrasekaran B, Feldmann A, Sitaraman R. The subtle art of not worrying about losses: Optimizing

video streaming with imperfect transmission. In: Proc. of the 2021 ACM CoNEXT. 2021. 1–16.
[17] Cerf V, Kahn R. A protocol for packet network intercommunication. IEEE Trans. on Communications, 1974, 22(5): 637–648.
[18] Tan K, Song J, Zhang Q, Sridharan M. A Compound TCP approach for high-speed and long distance networks. In: Proc. of the 25th IEEE Int'l Conf. on Computer Communications. Barcelona: IEEE, 2006. 1–12. [doi: 10.1109/INFOCOM.2006.188]
[19] Winstein K, Sivaraman A, Balakrishnan H. Stochastic forecasts achieve high throughput and low delay over cellular networks. In: Proc. of the 10th USENIX Conf. on Networked Systems Design and Implementation. Lombard: USENIX Association, 2013. 459–472.
[20] Winstein K, Balakrishnan H. TCP ex machina: Computer-generated congestion control. ACM SIGCOMM Computer Communication Review, 2013, 43(4): 123–134. [doi: 10.1145/2534169.2486020]
[21] Yan FY, Ma J, Hill GD, Raghavan D, Wahby RS, Levis P, Winstein K. Pantheon: The training ground for Internet congestion-control research. In: Proc. of the 2018 USENIX Annual Technical Conf. Boston, 2018. 1–13.
[22] Cardwell N, Cheng YC, Gunn CS, Yeganeh SH, Jacobson V. BBR: Congestion-based congestion control. Queue, 2016, 14(5): 20–53. [doi: 10.1145/3012426.3022184]
[23] Ha S, Rhee I, Xu LS. CUBIC: A new TCP-friendly high-speed TCP variant. ACM SIGOPS Operating Systems Review, 2008, 42(5): 64–74. [doi: 10.1145/1400097.1400105]
[24] Li T, Zheng K, Xu K. Acknowledgment on demand for transport control. IEEE Internet Computing, 2021, 25(2): 109–115. [doi: 10.1109/MIC.2020.3045208]
[25] Blanton E, Allman M, Wang L, Jarvinen I, Kojo M, Nishida Y. RFC 6675: A conservative loss recovery algorithm based on selective acknowledgment (SACK) for TCP. 2012. https://www.rfc-editor.org/rfc/rfc6675.html
[26] Braden R. RFC 1122: Requirements for Internet hosts—Communication layers. 1989. https://www.rfc-editor.org/rfc/rfc1122.html
[27] Allman M, Paxson V, Blanton E. RFC 5681: TCP congestion control. 2009. https://www.rfc-editor.org/rfc/rfc5681.html
[28] Iyengar J, Swett I. RFC 9002: QUIC loss detection and congestion control. 2021. https://www.rfc-editor.org/rfc/rfc9002.html
[29] Li T, Zheng K, Xu K, Jadhav RA, Xiong T, Winstein K, Tan K. TACK: Improving wireless transport performance by taming acknowledgments. In: Proc. of the 2020 Annual Conf. of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication. ACM, 2020. 15–30. [doi: 10.1145/3387514.3405850]
[30] Li T, Zheng K, Xu K, Jadhav RA, Xiong T, Winstein K, Tan K. Revisiting acknowledgment mechanism for transport control: Modeling, Analysis, and Implementation. IEEE/ACM Trans. on Networking, 2021, 29(6): 2678–2692. [doi: 10.1109/TNET.2021.3101011]
[31] Mathis M, Mahdavi J, Floyd S, Romanow A. RFC 2018: TCP selective acknowledgment options. 1996. https://www.rfc-editor.org/rfc/rfc2018.html
[32] Floyd S, Mahdavi J, Mathis M, Podolsky M. RFC 2883: An extension to the selective acknowledgement (SACK) option for TCP. 2000.
[33] Iyengar J, Thomson M. RFC 9000: QUIC: A UDP-based multiplexed and secure transport. Internet Engineering Task Force, 2021.
[34] Schulzrinne H, Casner S, Frederick R, Jacobson V. RFC 3550: RTP: A transport protocol for real-time applications. Internet Engineering Task Force, 2003.
[35] Iyengar J, Swett I. Sender Control of Acknowledgement Delays in QUIC, document draft 02. 2020. https://www.ietf.org/staging/draft-iyengar-quic-delayed-ack-None.html
[36] Salameh L, Zhushi A, Handley M, Jamieson K, Karp B. HACK: Hierarchical ACKs for efficient wireless medium utilization. In: Proc. of the 2014 USENIX Annual Technical Conf. Philadelphia: USENIX Association, 2014. 359–370.
[37] Balakrishnan H, Padmanabhan VN, Fairhurst G, Sooriyabandara M. RFC 3449: TCP performance implications of network path asymmetry. 2002. https://www.rfc-editor.org/rfc/rfc3449.html
[38] Cheng Y, Cardwell N, Dukkipati N, Jha P. RFC 8985: The RACK-TLP loss detection algorithm for TCP. 2021. https://www.rfc-editor.org/rfc/rfc8985.html
[39] Mathis M, Mahdavi J. Forward acknowledgement: Refining TCP congestion control. ACM SIGCOMM Computer Communication Review, 1996, 26(4): 281–291. [doi: 10.1145/248157.248181]
[40] Google Chrome team. WebRTC. 2019. https://webrtc.org/
[41] Gu YH, Grossman RL. UDT: UDP-based data transfer for high-speed wide area networks. Computer Networks, 2007, 51(7): 1777–1799. [doi: 10.1016/j.comnet.2006.11.009]
[42] He E, Leigh J, Yu O, DeFanti TA. Reliable Blast UDP: Predictable high performance bulk data transfer. In: Proc. of the 2002 IEEE Int'l Conf. on Cluster Computing. Chicago: IEEE, 2002. 317–324. [doi: 10.1109/CLUSTR.2002.1137760]
[43] Fox R. RFC 1106: TCP big window and nak options. 1989. https://www.rfc-editor.org/rfc/rfc1106.html
[44] Adamson B, Bormann C, Handley M, Macker J. RFC 5740: Nack-oriented reliable multicast (NORM) transport protocol. 2009. https://www.rfc-editor.org/rfc/rfc5740.html

[45] Pantheon. Pantheon of congestion control. 2018. http://pantheon.stanford.edu/
[46] Dong M, Li QX, Zarchy D, Godfrey PB, Schapira M. PCC: Re-architecting congestion control for consistent high performance. In: Proc. of the 12th USENIX Conf. on Networked Systems Design and Implementation. Oakland: USENIX Association, 2015. 395–408.
[47] Dong M, Meng T, Zarchy D, Arslan E, Gilad Y, Godfrey PB, Schapira M. PCC vivace: Online-learning congestion control. In: Proc. of the 15th USENIX Conf. on Networked Systems Design and Implementation. Renton: USENIX Association, 2018. 343–356.
[48] Microsoft. 2021. https://github.com/microsoft/msquic
[49] Facebook. 2021. https://github.com/facebookincubator/mvfst
[50] Huawei. 2021. https://developer.huawei.com/consumer/en/hms/huawei-hQUIC/
[51] Zheng ZL, Ma YF, Liu YM, Yang FR, Li ZY, Zhang YB, Zhang JH, Shi W, Chen WT, Li D, An Q, Hong H, Liu HH, Zhang M. XLINK: QoE-driven multi-path QUIC transport in large-scale video services. In: Proc. of the 2021 ACM SIGCOMM Conf. ACM, 2021. 418–432. [doi: 10.1145/3452296.3472893]

附中文参考文献:

[2] 罗万明, 林闯, 阎保平. TCP/IP拥塞控制研究. 计算机学报, 2001, 24(1): 1–18. [doi: 10.3321/j.issn:0254-4164.2001.01.001]
[3] 吴桦, 于振华, 程光, 胡晓艳. 大型指纹库场景中加密视频识别方法. 软件学报, 2021, 32(10): 3310–3330. http://www.jos.org.cn/1000-9825/6025.htm [doi: 10.13328/j.cnki.jos.006025]
[4] 王建新, 龚皓, 陈建二. 高带宽延时网络中一种协同式拥塞控制协议. 软件学报, 2008, 19(1): 125–135. http://www.jos.org.cn/1000-9825/19/125.htm [doi: 10.3724/SP.J.1001.2008.00125]
[5] 徐昌彪, 鲜永菊, 唐朝伟, 杨士中. 一种传输控制协议中的主动拥塞控制机制. 软件学报, 2008, 19(6): 1533–1545. http://www.jos.org.cn/1000-9825/19/1533.htm [doi: 10.3724/SP.J.1001.2008.01533]
[6] 张永敏, 徐伟强, 黄炯, 汪亚明, 舒挺, 刘良桂. Ad Hoc网络节能型功率控制与拥塞控制的跨层优化. 软件学报, 2013, 24(4): 900–914. http://www.jos.org.cn/1000-9825/4317.htm [doi: 10.3724/SP.J.1001.2013.04317]
[7] 任丰原, 林闯, 刘卫东. IP网络中的拥塞控制. 计算机学报, 2003, 26(9): 1025–1034. [doi: 10.3321/j.issn:0254-4164.2003.09.001]
[8] 杜鑫乐, 徐恪, 李彤, 郑凯, 付松涛, 沈蒙. 数据中心网络的流量控制: 研究现状与趋势. 计算机学报, 2021, 44(7): 1287–1309. [doi: 10.11897/SP.J.1016.2021.01287]
[9] 李丹, 陈贵海, 任丰原, 蒋长林, 徐明伟. 数据中心网络的研究进展与趋势. 计算机学报, 2014, 37(2): 259–274. [doi: 10.3724/SP.J.1016.2014.00259]
[10] 徐恪, 朱敏, 林闯. 互联网体系结构评估模型、机制及方法研究综述. 计算机学报, 2012, 35(10): 1985–2006. [doi: 10.3724/SP.J.1016.2012.01985]
[11] 宛考, 罗雪峰, 江勇, 徐恪. 软件定义网络系统中面向流的调度算法. 计算机学报, 2016, 39(6): 1208–1223. [doi: 10.11897/SP.J.1016.2016.01208]

李彤(1989－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为新一代互联网体系结构, 分布式系统, 大数据.

徐恪(1974－), 男, 博士, 教授, 博士生导师, CCF 杰出会员, 主要研究领域为新一代互联网体系结构, 网络空间安全, 区块链系统.

郑凯(1978－), 男, 博士, 主要研究领域为数据中心网络高性能通信, 广域网弱确定性通信, 终端设备间近场极简通信.