TCP SkyLine:数据中心网络高吞吐率传输

王 娟,夏 羽
(四川师范大学计算机科学学院, 四川成都 610101)

摘 要: 针对数据中心网络在“多对一”并发流量模式下,TCP(Transmission Control Protocol)及其现有改进方案在单轮数据传输和多轮数据传输下吞吐率低下问题,提出了一种通过数据包标记实现丢包快速发现和快速重传并动态调整拥塞窗口初始值的策略,称为 TSL(TCP SkyLine). TSL 同时解决了传统 TCP Incast 问题和多轮数据传输下由遗留窗口引发的 TCP Incast 问题. 实验表明,TSL 在单轮数据传输和多轮数据传输下均能获得 90% 以上的带宽利用率. 在 10Gbps 网络中,其支持的并发连接数与传统 TCP 和 DCTCP 相比分别提升了 5 倍和 1 倍,有效吞吐率分别提升了 18 倍和 8.6 倍;在 1Gbps 网路中,支持的并发连接数较传统 TCP 和 DCTCP 分别提升了 5.8 倍和 1 倍.

关键词: 数据中心; 传输协议; TCP Incast; 延迟 ACK; 超时重传; 多轮数据传输
中图分类号: TP393 **文献标识码:** A **文章编号:** 0372-2112 (2020)12-2425-09
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2020.12.019

TCP Skyline:A High-Throughput Transport for Data Center Networks

WANG Juan, XIA Yu
(*Department of Computer Science, Sichuan Normal University, Chengdu, Sichuan* 610101, *China*)

Abstract: In the “many-to-one” concurrent traffic pattern of the data center networks, TCP (Transmission Control Protocol) and its existing improvements have low throughput in single-round and multi-round of data transmission scenarios. Hence, a strategy called TSL (TCP SkyLine) is proposed for fast discovering and retransmitting the lost packets through packet labelling and adjusting the initial size of the congestion window dynamically. TSL solves the traditional TCP Incast problem and the TCP Incast problem caused by the legacy window in multi-round of data transmission. Through extensive experiments we show that TSL can achieve more than 90% goodput regardless of single or multiple rounds of data transmission. In a 10Gbps network, the number of concurrent connections supported by TSL has increased by 5 times and once respectively compared to the traditional TCP and DCTCP, and the goodput has increased by 18 times and 8.6 times respectively. In a 1Gbps network, the number of concurrent connections supported by TSL has increased by 5.8 times and once respectively compared to the traditional TCP and DCTCP.

Key words: data center; transmission protocol; TCP Incast; delay ACK; retransmission timeout; multi-turn transmission

1 引言

数据中心的运行效率决定了云计算、人工智能、移动计算等计算密集型业务的服务质量. 在数据中心中,对复杂的计算任务,通常采用“划分/汇总”的计算模型进行处理,如 MapReduce[1,2] 等分布式计算框架. 在该模型下,工作节点向任务发起节点汇总数据时会产生“多对一”流量传输模式. 当工作节点较多时,交换机瓶颈链路会出现严重拥塞. 目前许多数据中心出于成本考虑而使用浅缓存的交换机,高并发的流量易导致大量丢包. 当工作节点无法通过 N 个(N 为触发快速重传的 ACK 个数,默认值为 3)重复 ACK 发现数据包丢失而等待超时重传(Retransmission TimeOut, RTO)时,由于传统 TCP 协议的 $RTO_{min}^{[3]}$ (minimum Retransmission TimeOut, Linux 系统中默认为 200ms)与数据中心网络的 RTT(Round-Trip Time,通常在微秒级别)严重不匹配,将使网络吞吐率出现灾难性坍塌,严重时可能导致任务错过截止完成时间而违反 SLA[4,5] (Service-Level A-

收稿日期:2019-08-15;修回日期:2020-05-27;责任编辑:覃怀银

greement),该问题被称为 TCP Incast 问题.

此外,由于服务器的数据处理能力有限,"划分/汇总"的计算模型通常分多轮进行.主节点每轮向各工作节点请求指定大小的数据,称为 SRU[6,7](Server Request Unit).在多轮数据请求下,上轮数据传输完成时遗留的拥塞窗口可能加剧新一轮数据传输的 TCP Incast 问题,使网络支持的并发连接总数(可同时参与传输的工作节点总数,以下简称并发数)较单轮数据传输时更少.

近年来,研究人员针对 TCP Incast 问题提出了大量解决方案[8,9].DCTCP[10] 使用 ECN(Explicit Congestion Notification)机制来保持交换机队列长度尽可能短.E-DCTCP[11] 在 DCTCP 的基础上,把即将丢弃的数据包回传给发送端以及时发现丢包并降低发送窗口.类似的方案还有 PDN[12] 和 RPPLN[13].此类方案需更换定制的网络设备,无法在现有网络中使用.TCP Plato[14] 通过对部分数据包进行标记保证有足够的 ACK 触发快速重传.该方案和本文提出的方案类似,但由于其数据包标记频繁,导致其支持的并发数有限.T-Racks[15] 为每条流定义计时器,并在计时器超时后让接收端发送"伪 ACK"触发快速重传以避免 RTO.该方案需维护一张巨大的流表且计时器可能退避到 RTO_{min}.

综合分析已有的研究成果,目前针对 TCP Incast 问题的优化方案大多基于单轮数据传输场景进行改进,而忽略了在多轮数据请求时由遗留窗口引发的 TCP Incast 问题.因此,本文提出了 TSL(TCP SkyLine)方案.TSL 通过标记数据包来加快丢包发现和数据重传过程,缓解单轮数据传输时的 TCP Incast 问题;并使用拥塞自适应的窗口调整策略缓解 TSL 因延迟 ACK 机制引起的吞吐率与支持并发数之间的矛盾.此外,TSL 还通过动态调整每轮数据传输的拥塞窗口初始值,缓解多轮数据请求场景下的 TCP Incast 问题.该方案仅需更新服务器中少量的 TCP 代码,无需更换现有网络设备,适合在现有数据中心部署.

2 数据中心 TCP 吞吐率优化问题

2.1 问题描述

2.1.1 单轮数据传输的 TCP Incast 问题

本文在如图 1 所示的网络环境(本文所有实验均基于该拓扑)下,让多个服务器(模拟工作节点)向同一服务器(模拟主节点)汇聚数据.链路带宽为 10Gbps,链路每跳时延为 2μs,即总基础 RTT 为 8μs,交换机采用固定缓存分配模式,每个端口分配 128KB 缓存,后文所述的交换机缓存均指每个端口的缓存大小.由主节点向所有工作节点发起数据请求,SRU 为 256KB.

首先考虑单轮数据请求的情况.设置 DCTCP 的 K 值分别为 20(缓存为 128KB 时)和 40(缓存为 256KB 时),g 为 0.0625[10].通过改变并发工作节点的数量和交换机缓存大小,观察传统 TCP 和 DCTCP 的有效吞吐率(goodput),其计算方式为:$G = d_{total}/t_{all}$,G 表示有效吞吐率,d_{total} 为主节点应用层收到的所有数据量,t_{all} 表示从连接建立成功开始,主节点接收 d_{total} 所用的时间.实验结果如图 2 所示.在交换机缓存为 128KB 时,当并发数增加到 17 以上,传统 TCP 的有效吞吐率崩溃到 1Gbps 以下并始终维持在较低水平,带宽利用率不到 10%.DCTCP 由于启用了 ECN 机制,故可支持的并发数较 TCP 有较大提升,但依然有限.通过将交换机缓存扩大到 256KB,可在一定程度上缓解该问题,但效果不明显.且扩大交换机缓存需更换现有网络设备,代价较高.

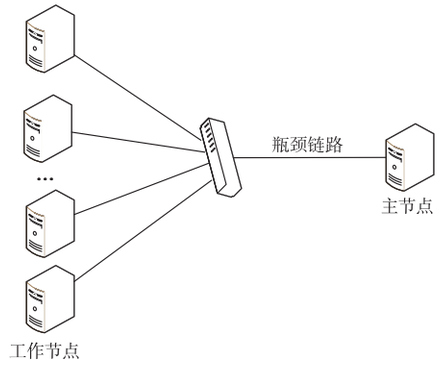


图1 实验网络拓扑结构图

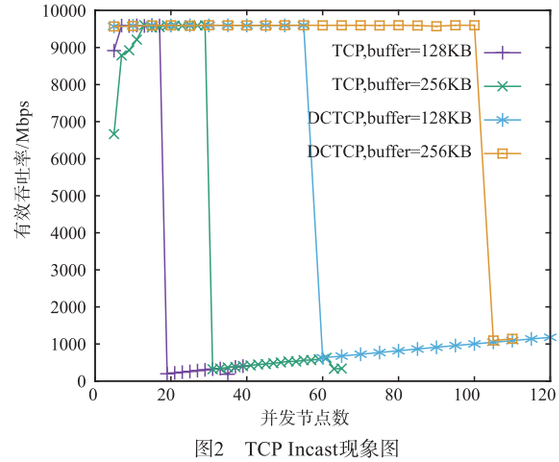
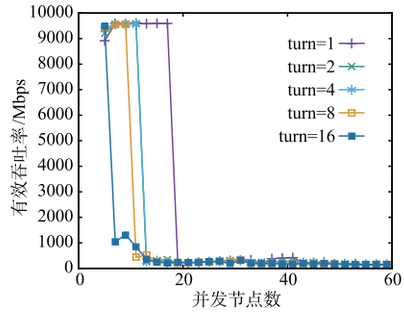


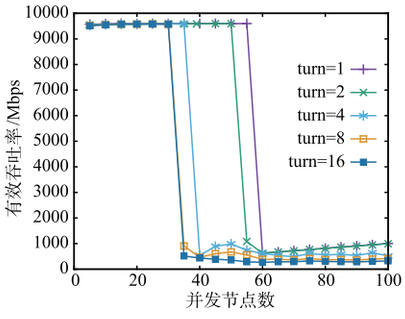
图2 TCP Incast现象图

2.1.2 多轮传输下并发数下降问题

在与 2.1.1 小节相同的实验环境下,设置交换机缓存为 128KB,分别使用传统 TCP 和 DCTCP 协议,让主节点依次向工作节点发起 1、2、4、8、16 轮数据请求.要求主节点只有在收到所有工作节点在该轮发送的 SRU 后才可发起新一轮 SRU 请求.实验结果如图 3(a)和图 3(b)所示,在多轮数据请求下,无论是传统 TCP 还是 DCTCP,可支持的最大并发数较单轮数据请求时均更少.且数据请求的轮数越多,可支持的并发数越少.



(*a*) 传统 TCP 的多轮传输



(*b*) DCTCP的多轮传输

图3　多轮传输的影响

2.2　问题分析

RTO 是引发 TCP Incast 问题的主要因素. 文献(如[14,16]等)中曾指出引发 RTO 的场景有:(1)拥塞窗口(或所有消息)尾部的 N 个数据包中任意一个或多个丢失;(2)发送端整个发送窗口全部丢失;(3)重传包丢失. 出于优化需要,本文将引发 RTO 所丢失的数据包重新分为两大类:场景(1)和场景(2)中触发快速重传所必需的尾部数据包(简称触发包)和场景(3)中的重传包. 触发包和重传包一旦丢失,必然引起 RTO. 在高带宽低延迟的数据中心中,由于大部分流都为短流,RTO 极易引发 TCP Incast 问题.

2.2.1　触发包丢失

传统 TCP 在发送端收到 N 个重复 ACK 时,则认为数据包发生丢失,会立即进入快速重传对丢失的数据包进行恢复[15]. 然而,在没有后续数据包可传输的情况下,若触发包丢失,接收端将无法回复足够的重复 ACK 触发快速重传,只能等待 RTO. 而触发包大多位于拥塞窗口尾部[11]. 因此,拥塞窗口末尾的数据包比其他位置的数据包更加关键.

2.2.2　重传包丢失

在传统 TCP 中,若重传的数据包再次丢失,由于发送端无法准确区分重复 ACK 是由重传之前还是重传之后的数据包产生的,故只能等待超时重传[14,17]. 该做法也是为了避免 TCP 循环进入快速恢复阶段,使发送端反复重传已发送的数据包,降低带宽利用率. 据本文实验统计,“多对一”流量模式下,重传包丢失引发的 RTO 次数占比较高. 因此,重传包丢失将大大增加 TCP Incast 问题产生的概率.

2.2.3　遗留窗口问题

在多轮数据传输场景下,主节点必须收到所有工作节点的 SRU 后才发起下一轮数据请求的工作方式,使得原本通过带宽竞争已经错开传输的数据流,再次形成流量并发.

此外,各工作节点在上一轮数据传输的遗留窗口可视为下一轮数据传输的起始拥塞窗口(Initial Congestion Window Size,ICWS). 而较大的 ICWS 易让发送端在发送第一个窗口数据时就引起交换机缓存溢出. 本文在与 2.1.1 小节相同的环境下设置交换机缓存为 128KB,ICWS 分别为 1MSS、2MSS、4MSS 和 Google 曾建议的 10MSS[18],使用 DCTCP 让主节点向工作节点请求 256KB 的数据. 如图 4 所示的实验结果显示,ICWS 越大,可支持的并发数越少. 因为 ICWS 越大,在高并发下产生的突发数据包越多,越容易造成交换机缓存溢出. 因此,遗留窗口是多轮传输下并发数恶化的重要原因. 要在多轮数据传输下支持更多的并发数,应考虑为每轮数据传输选择合理的 ICWS.

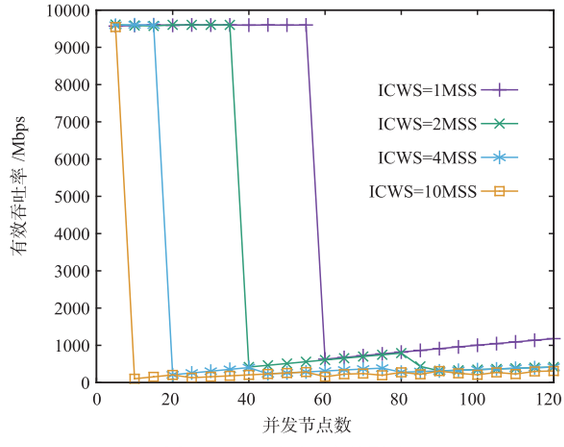


图4　ICWS对吞吐率的影响

综上所述,触发包和重传包的丢失是单轮数据传输产生 TCP Incast 问题的主要原因,而遗留窗口则是导致多轮数据传输下 TCP Incast 问题提前发生的重要因素.

3　TSL 设计

3.1　丢包发现与恢复

据前文分析,TSL 对滑动窗口和所有消息可发送的最后一个数据包进行特殊标记,并称其为“信号包”(Signal Packet,SP). 在不考虑多路径的网络中,接收端可根据 SP 的到达判断该窗口内是否有数据包丢失. 当

接收端收到 SP 时,若在 SP 之前发送的数据包尚未到达,则认为数据包可能发生了丢失. 此时 TSL 让接收端立即向发送端回复一个携带特殊标记的 ACK,称为“信号 ACK”(Signal ACK,SA),以请求丢失的数据. 为了让连接保持丢包检测能力,TSL 要求交换机不丢弃带有特殊标记的包,确保一个窗口内至少有一个数据包顺利到达接收端,并让对应的 SA 回到发送端.

为了避免在多路径网络中把滞留在网络中的错序数据包误判为丢包,TSL 收到 SA 后,若发现重复 ACK 计数未达到 N 且发送端又无法发送新数据,则发送带特殊标记的空数据包,称为“空信号包”(Dummy Signal Packet,DSP). 若接收端在收到 DSP 时仍有数据缺失,则继续回复 SA. 该过程一直持续到产生足够的重复 ACK 触发快速重传(如图 5)或接收端收到疑似丢失的数据并开始请求新的数据为止.

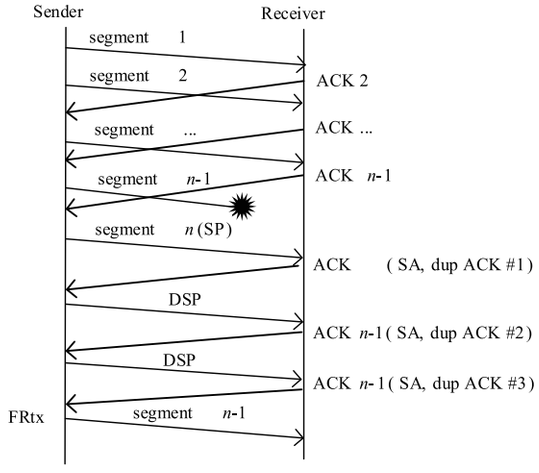


图5 通过DSP发现和恢复丢失数据

然而,在网络较拥塞的情况下,若坚持使用 3 个重复 ACK 触发快速重传,可能导致网络中的 DSP 过多,加剧网络拥塞. 因此,TSL 根据当前滑动窗口的大小 W 调整进入快速重传的条件. 定义滑动窗口阈值为 S,当 $W>S$ 时,说明网络拥塞程度较小,发送端坚持收到 3 个重复 ACK后才进入快速重传,避免因误判丢包而降低发送窗口;当 $W\leqslant S$ 时,说明网络较为拥塞,发送过多的 DSP 会加剧网络拥塞. 此时,发送端若收到 SA,即判断为丢包并立即进入快速重传而不再等待 3 个重复 ACK. 实验中,设置 $S=3$MSS,因为此时滑动窗口较小,网络中滞留数据包的概率较小,即便丢包误判引起窗口降低和数据重传,对性能影响也不大.

3.2 防止重传包丢失

文献[17]曾提出使用 SACK 结合 OR(Opportunity Retransmission)对重传包丢失问题进行优化. 该方案依赖于接收窗口的最后一个数据包,若该数据包丢失,该方案将失效. 因此,TSL 对所有重传包进行标记,让交换机不丢失重传包,可有效解决由重传包丢失引起的 RTO.

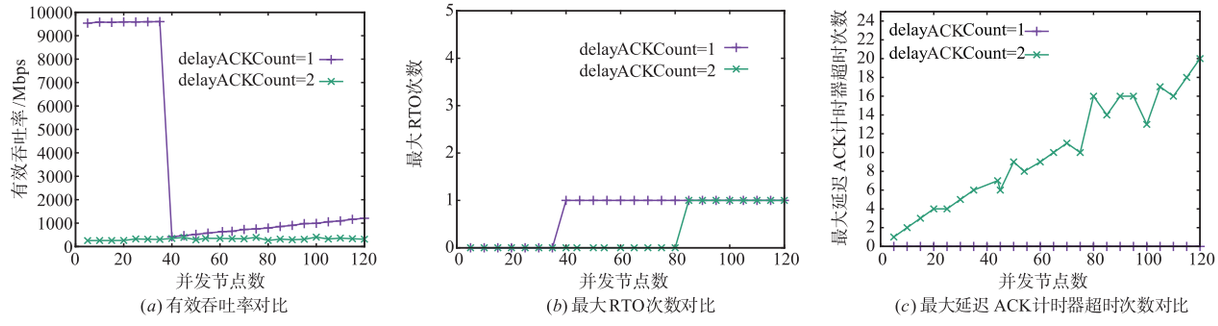
考虑到在高并发下可能出现大量丢包导致重传包较多,存在交换机缓存被重传包占满的风险. 故 TSL 结合了 ECN 机制,在网络即将拥塞但未发生丢包时显式地通知发送端降低拥塞窗口,减少重传包的数量.

3.3 拥塞自适应的窗口增长策略

3.3.1 延迟 ACK 对 TSL 的影响

基于 3.1 和 3.2 的方案能较好地缓解传统 TCP 中的 RTO. 但本文在实验过程中发现 TSL 性能对延迟 ACK[19]的启用与否较为敏感. 该机制指在未发生数据包丢失时,接收端只有在延迟 ACK 计数器达到阈值(Linux 系统默认值为 2)或延迟 ACK 计时器达到预定超时时间(Linux 系统默认为 40ms)时才回复一个 ACK.

在交换机缓存为 128KB 的环境下,设置 SRU 为 256KB,并基于 2.2.3 节的结论令 ICWS 为 1MSS,分别对 TSL 启用和关闭延迟 ACK 时的有效吞吐率、最大 RTO 次数和最大延迟 ACK 计时器超时次数进行统计,结果分别如图 6(a)~(c)所示. 据实验结果可知,启用延迟 ACK 时,由于延迟回复 ACK 可减缓发送端拥塞窗口的增长速度,故支持的并发数较高;但在网络极度拥塞时,一些流的拥塞窗口可能降低到 1MSS,此时延迟 ACK 计数器无法达到阈值,只能等待延迟 ACK 计时器超时. 且超时时间还会累计,由此产生的链路空闲可能比 RTO 的影响更为严重. 而关闭延迟 ACK 时则情况相反.



(a) 有效吞吐率对比 (b) 最大RTO次数对比 (c) 最大延迟 ACK计时器超时次数对比

图6 延迟ACK对TSL性能的影响

由此可见,TSL 存在由延迟 ACK 带来的吞吐率与支持并发数互相矛盾的问题. 为缓解该问题,TSL 进一步提出了拥塞自适应的窗口调整(Congestion Adaptive Window Adjustment,CAWA)策略.

3.3.2　CAWA 策略

启用延迟 ACK 时,延迟 ACK 计时器超时主要是在网络重度拥塞(或 ICWS = 1MSS)时,发送端的滑动窗口只可发送一个(或更少)数据包导致的. 在 TSL 中,当发送端的拥塞窗口只可发送一个数据包时,该数据包自然成为窗口最后一个包,一定被标记为 SP. CAWA 的核心思想在于让接收端根据网络拥塞状况选择何时对 SP 回复 ACK,以策略性地控制发送端拥塞窗口的增长.

接收端收到 SP 时,CAWA 的工作状况可分以下几种情况讨论:

(1)若无数据缺失且此时网络不拥塞,则立即回复普通 ACK 以增大发送端滑动窗口,其中拥塞状况主要依靠 ECN 机制进行判断;

(2)若无数据缺失,但在该 RTT 内收到携带拥塞信息的数据包,则判断网络处于拥塞状态,此时通过预估发送端的窗口大小(记为$\tilde{W}_{avail}$)来选择如何回复 ACK. $\tilde{W}_{avail}$主要通过相邻 SP 的序列号之差来估计(此处暂不考虑 SP 是 SRU 最后一个数据包的情况),如图 7 所示. TSL 根据$\tilde{W}_{avail}$的大小选择如何回复 ACK:

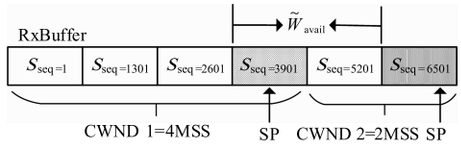


图7　拥塞窗口估算示意图

(*a*)若$\tilde{W}_{avail} > 1MSS$,说明发送端可发送新数据让延迟 ACK 计数器达到阈值. 此时,按正常延迟 ACK 机制执行;

(*b*)若$\tilde{W}_{avail} \leqslant 1MSS$,说明发送端由于窗口限制已无法再发送新数据. 对实验结果的分析表明,若此时立即回复 ACK,会出现发送窗口增长过快而导致交换机队列溢出的问题. 因此,TSL 将延迟回复 ACK. 与传统 TCP 固定超时时间不同,TSL 设置的延迟时间与该 SP 的传输时间有关. 定义接收端收到上一个 SP 的时间为 t_{prev},收到当前 SP 的时间为 t_{curr},则当前 SP 的传输时间粗略估计为 $t_{sp} = t_{curr} - t_{prev}$. 考虑到服务器计时器的精度,延迟回复 ACK 的时间 t_{del} 由当前 SP 的传输时间 t_{sp}、延迟 ACK 计时器超时时间 t_{DTO} 和当前主机可支持的最小时钟精度 t_{minAcc} 共同决定:

$$t_{del} = \min((\max(t_{sp}, t_{minAcc}), t_{DTO})$$

由此可见,只有在网络极度拥塞时,t_{del} 才退避到 t_{DTO}. 此时,网络已极度拥塞,t_{DTO} 的延迟对性能影响较小. 且等待 t_{prev} 的时间再回复 ACK,相当于在网络拥塞时等待一个数据包离开网络,即虚拟地将延迟 ACK 计数器增加至 2,可在链路利用率和支持并发数之间取得较好平衡.

(3)若有缺失数据,则无论此时网络是否拥塞,均向发送端回复 SA,帮助发送端快速发现丢包;

若 SP 是 SRU 最后一个数据包,则按 3.4 所述策略处理.

3.4　窗口对齐

多轮传输在一定程度上可看作单轮传输的不断重复. 据图 4 可知,ICWS 越小,网络支持的并发数越多. 因此,TSL 在发送端记录了每轮数据传输过程中拥塞窗口达到的最小值 $CWND_{min}$(不包括 ICWS)和慢启动阈值达到的最小值 $ssThresh_{min}$,并在下轮传输开始时将发送端的拥塞窗口和慢启动阈值分别对齐到 $CWND_{min}$ 和 $ssThresh_{min}$,称为"窗口对齐". 因为 $CWND_{min}$ 反应了上轮数据传输中网络达到的最大拥塞程度,调整慢启动阈值则可降低丢包发生的概率. 此处不将窗口直接对齐到 1MSS,主要考虑当各工作节点回复的数据大小不同时,部分流会提前完成所有数据传输,此时,下一轮的最优启动窗口大小会有所变化. 当只剩少量的流还在传输数据时,若窗口仍从 1MSS 开始,将无法充分利用带宽.

窗口对齐的关键在于发送端可正确判断一轮数据传输的结束. 在 TSL 中,每轮传输结束时发送的最后一个数据包一定为 SP. TSL 对该 SP 使用额外标记,并称其为"尾部信号包"(Tail Signal Packet,TSP). 接收端收到 TSP 后,若无数据包丢失,则立即回复特殊标记的 ACK,称为"尾部信号 ACK"(Tail Signal ACK,TSA). 反之则回复 SA,并在收到该轮传输的所有数据后向发送端回复 TSA. 发送端在收到 TSA 后得知本轮传输结束,将调整拥塞窗口和慢启动阈值.

窗口对齐技术将多轮数据传输的吞吐率优化问题简化为单轮数据传输的吞吐率优化问题. 无论发生多少轮数据传输,均能支持较高的并发数.

4　TSL 实现

TSL 的实现主要涉及数据包标记和防止标记包丢失两大方面. 对数据包的标记本文采用了 IP 包头中已有的 DSCP[20](Differentiated Services Code Point)字段,通过为不同类型的标记包设置不同的 DSCP 值,可让发送端和接收端识别不同类型的数据包.

防止标记包丢失主要通过在交换机上启用 WRED(Weighted Random Early Detection)队列实现. 该队列可对不同的 DSCP 值(区间)设置不同的最小丢包(或 ECN 标记)阈值 minTh、最大丢包阈值 maxTh 和丢包(或 ECN 标记)概率. 目前许多低端交换机也支持 WRED 队列[14].

为了及时反馈网络拥塞状况,TSL 还在交换机上启用了 ECN 机制,并给标记包和非标记包设置了相同的 minTh 值(保证带宽共享的公平性)和不同的 maxTh 值. 记标记包和非标记包的最小标记阈值为 minTh,非标记包和标记包的最大丢包阈值分别为 $\text{maxTh}_{\text{unlabel}}$ 和 $\text{maxTh}_{\text{label}}$,队列的最大长度为 $l_{\max}$. 计算队列长度时使用瞬时队列长度,时刻 i 的队列长度记为 l_i. 为了保证标记包不丢失,应满足 $\text{maxTh}_{\text{unlabel}} < l_{\max}$,$\text{maxTh}_{\text{label}} = l_{\max}$.

如图 8,当 $l_i < \text{minTh}$ 时,所有数据包可公平进入队列. 当 $\text{minTh} < l_i < \text{maxTh}_{\text{unlabel}}$ 或 $\text{minTh} < l_i < \text{maxTh}_{\text{label}}$ 时,对进入队列的数据包按概率标记 CE(Congestion Experienced). 而当 $l_i > \text{maxTh}_{\text{unlabel}}$ 时,交换机将丢弃到达的非标记包,只让标记包进入队列,直到交换机缓存溢出才丢失标记包.

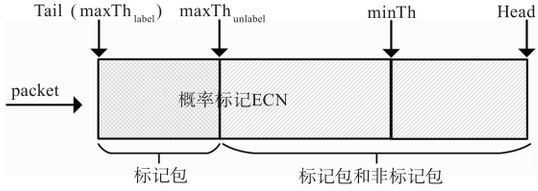


图8 WRED算法示意图

5 实验结果与分析

本文主要从参数选取、TSL 在单轮传输和多轮传输下的性能表现等方面对 TSL 进行了综合评估.

实验在 NS-3 上构建了如图1所示的网络. 所有仿真中,若未特别说明,$\text{RTO}_{\min}$ 为 200ms,基础 RTT 为 8μs. 对 TSL,设置延迟 ACK 计数器阈值为 2,延迟 ACK 超时时间为 40ms. 其余算法则禁用延迟 ACK. WRED 队列的 minTh 和 maxTh 参数值按占队列总长度的比例进行设置. 标记包的 $\text{maxTh}_{\text{unlabel}}$ 值始终取 1(队列最大长度).

5.1 WRED 参数选取

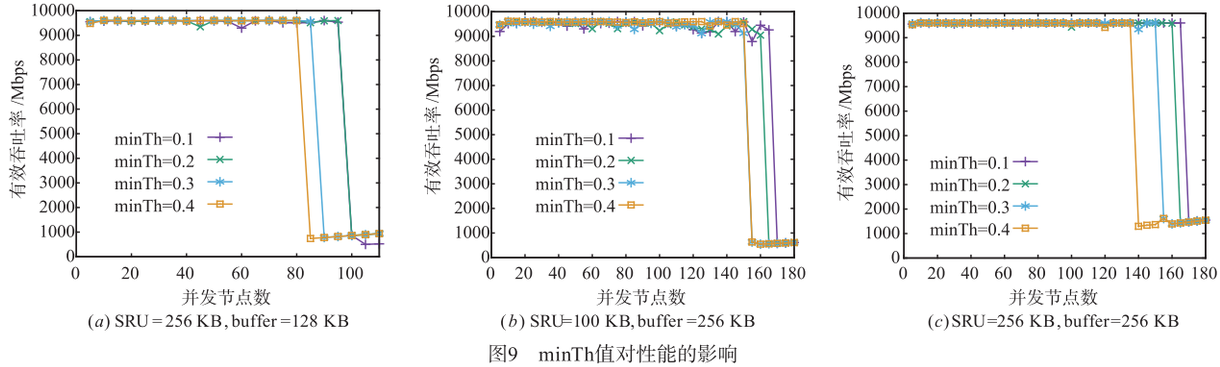
WRED 队列的参数选择,目前没有行之有效的解决方案,主要通过实际测试来确定. 通过大量实验仿真,发现在不同场景下,minTh 和 $\text{maxTh}_{\text{unlabel}}$ 参数的最优值不同但又有所规律. 总结为以下几点:

(1)在合理范围内,更小的 minTh 值可让 TSL 获得更好的性能.

设置交换机缓存分别为 128KB 和 256KB,固定 $\text{maxTh}_{\text{unlabel}}$ 为 0.7,SRU 分别设置为 100KB 和 256KB,对 WRED 设置不同的 minTh 值进行仿真,实验结果如图9(*a*)~(*c*)所示. 在合理范围内,越小的 minTh 值可获得更好的性能. 因为在高并发下,更小的 minTh 值可及时反馈网络拥塞,减缓发送端发送速率从而减少丢包. 大量实验仿真结果表明,minTh 值设置为 0.1 为大多数场景下的最优值.

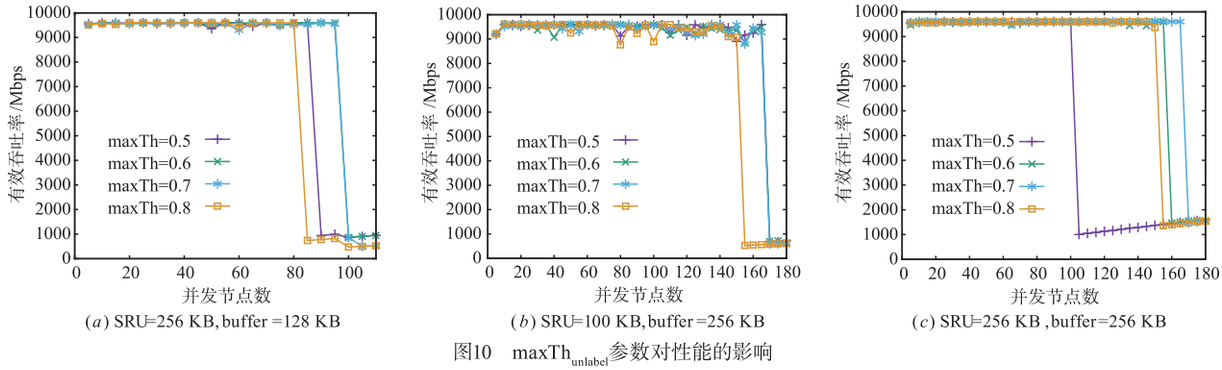
(2)在不同交换机缓存和 SRU 下,$\text{maxTh}_{\text{unlabel}}$ 均呈现出在合理范围内越大,TSL 性能越好的规律.

固定 minTh 值为 0.1,重复与(1)相同的实验. 如图10(*a*)~(*c*)的实验结果显示,不同 SRU 和不同交换机缓存下,$\text{maxTh}_{\text{unlabel}}$ 在合理范围内越大,TSL 性能越好. 因为过小的 $\text{maxTh}_{\text{unlabel}}$ 值在高并发下易造成非标记包



(*a*) SRU=256 KB, buffer=128 KB (*b*) SRU=100 KB, buffer=256 KB (*c*) SRU=256 KB, buffer=256 KB

图9 minTh值对性能的影响



(*a*) SRU=256 KB, buffer=128 KB (*b*) SRU=100 KB, buffer=256 KB (*c*) SRU=256 KB, buffer=256 KB

图10 $\text{maxTh}_{\text{unlabel}}$ 参数对性能的影响

大量丢失,丢失的非标记包转化为重传包后易使交换机缓存被快速填满. 而过大的 $\text{maxTh}_{\text{unlabel}}$ 值会导致预留给标记包的空间不足,降低网络的丢包检测能力. 通过大量仿真发现,在大部分情况下,$\text{maxTh}_{\text{unlabel}}$ 值为 0.7 时,TSL 可获得较好性能.

5.2　对主机计时器的精度要求

主机计时器的精度主要影响 TSL 在网络未崩溃前的有效吞吐率. 在交换机缓存为 256KB 的环境下,设置 SRU 为 256KB,让主节点请求 256KB 的数据. 如图 11 的结果显示,主机可支持的计时器精度越高,TSL 性能越好. 且只要服务器可支持 5ms 以上的时钟精度,TSL 即可获得较高性能. 数据中心的大部分主机均支持该精度.

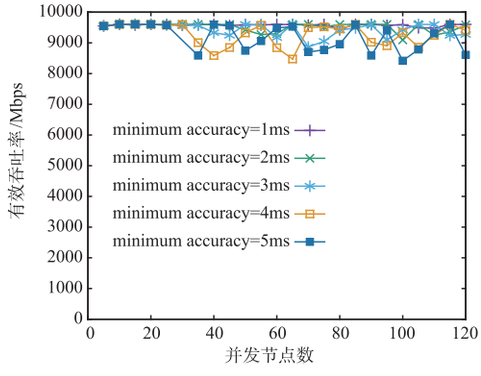


图11　时钟精度对性能的影响

5.3　CAWA 策略的有效性

本节实验验证 CAWA 策略对 TSL 中由延迟 ACK 引起的吞吐率和并发数之间矛盾的有效性. 在交换机缓存为 256KB 的环境下让主节点请求 256KB 的数据,SRU 为 256KB. 比较 TSL 启用延迟 ACK 时,使用 CAWA 策略、不使用 CAWA 策略和 TSL 禁用延迟 ACK 三种场景下的有效吞吐率(图 12(*a*))和最大 RTO 次数(图 12(*b*)). 据实验结果可知,TSL 在不使用 CAWA 的情况下,会产生网络吞吐率与支持并发数的矛盾,这点与 3.3.1 节分析一致. 而同时启用延迟 ACK 和 CAWA 策略后,由于接收端可根据网络状况选择正确的时机回复 ACK,因此 TSL 可在保持高吞吐率的同时支持较高的并发数.

5.4　单轮数据传输下的吞吐率提升

为了观察 TSL 在单轮数据传输下的提升,本文在缓存为 256KB 的交换机上,让主节点向工作节点请求 1 轮数据,SRU 为 256KB. 设置 DCTCP 的 K 值为 40,g 为 0.0625,Plato 算法的 minTh 和 maxTh 参数采用原论文中的最优值 0.7,设置 TSL 的 minTh 值为 0.1,$\text{maxTh}_{\text{unlabel}}$ 值为 0.7. 各算法的实验结果如图 13(*a*)所示,DCTCP 启用了 ECN,故所支持的并发数较传统 TCP 有所提升,

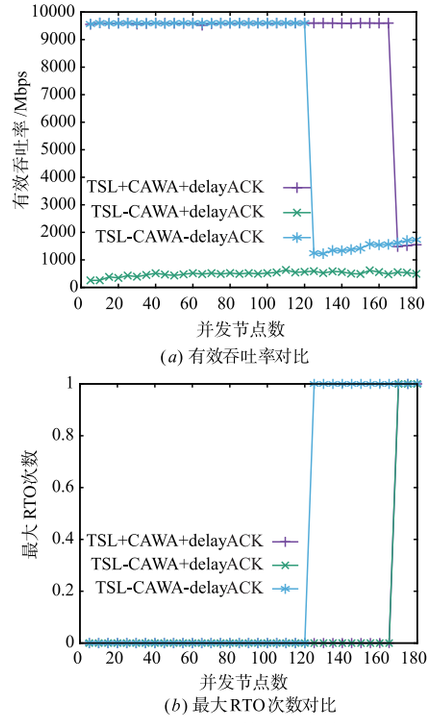


图12　CAWA策略的有效性

但由于缺乏有效的丢包检测机制,效果有限;Plato 每隔一个 RTT 就标记一个数据包,在 RTT 较小的环境下,网络中的标记包较多,造成非标记包大量丢失,致使产生的“空包”也较多,故其有效吞吐率较其余算法更低. 且其收到标记包时,不考虑拥塞状况就立即回复 ACK,易加重网络拥塞,导致其支持的并发数有限. 而 TSL 既能支持高并发的数据传输,又能实现 95% 以上的带宽利用率.

5.5　多轮数据传输下的吞吐率提升

将 5.4 中的单轮数据传输改为多轮数据传输场景进行实验,让主节点向工作节点发起 4 轮数据请求. 对比图 13(*a*)和图 13(*b*)的实验结果,由于遗留窗口问题,DCTCP 和 TCP Plato 在多轮数据传输下的性能较单轮数据传输时均有所下降,传统 TCP 因为其支持的并发数本身较少,性能下降不明显. 而 TSL 采用窗口对齐策略,在单轮和多轮数据传输下均可支持较高的并发连接.

5.6　异构数据中心网络环境下的性能

在某些增量部署的混合数据中心(同时有高速和低速网络设备)中,同一个 TCP 协议必须适应高速低时延、高速高时延等多种网络场景. 故本文分别在 1Gbps(RTT = 100μs)的低速高时延和 10Gbps(RTT = 100μs)的高速高时延网络环境中重复 5.4 节实验,结果如图 13(*c*)和图 13(*d*)所示. DCTCP 在高速高时延的环境下由于拥塞信息反馈不够及时,支持的并发数下降较多. Plato 在低速高时延环境下吞吐率波动也较大. 而TSL

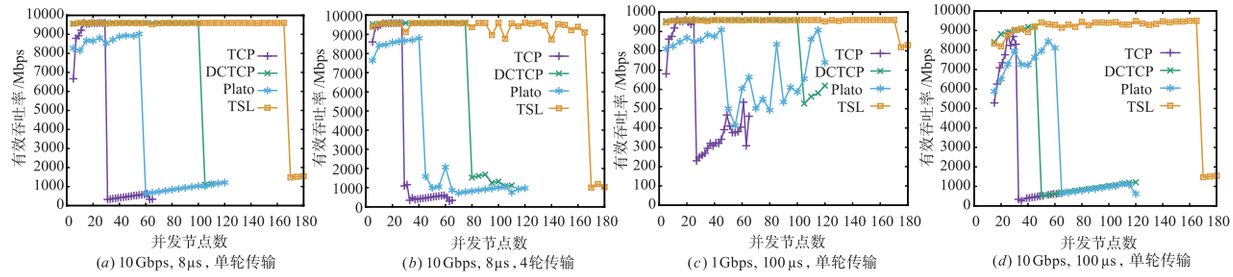


图13 各算法性能比较

则能在高速低时延、高速高时延及低速高时延等环境下均保持90%以上的吞吐率,同时支持160台左右的并发数. 适合在混合数据中心环境部署.

5.7 多条流共享带宽的公平性

固定交换机缓存为256KB,在网络中运行5条数据流,共享10Gbps带宽,每条流按最大速率持续发送数据直至离开网络,所有节点均运行TSL算法. 仿真持续45s,每间隔5s启动一条数据流,待所有流启动完成后持续运行5s. 之后再让网络中的数据流按5s的时间间隔依次离开网络,观察每条数据流实时获取网络带宽的情况. 通过图14的实验结果可知,当网络中有新的数据流加入或离开时,网络中现有的数据流均可在短时间内实现带宽公平共享. 故TSL对TCP的修改并不影响多条数据流共享网络带宽的公平性.

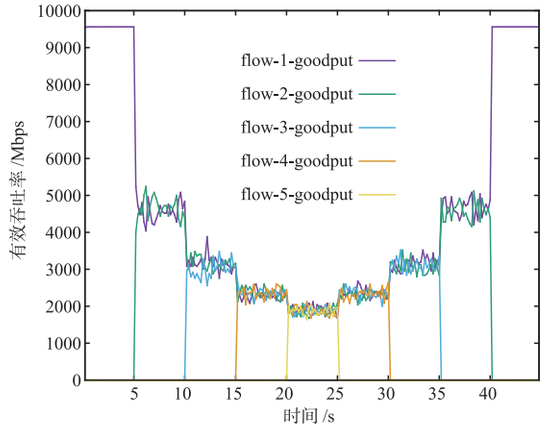


图14 共享带宽的公平性示意图

6 结束语

本文针对数据中心网络中“多对一”流量模型下网络传输效率过低的问题,提出了可快速发现丢包并快速恢复、动态调整拥塞窗口及拥塞窗口自适应调整的策略TSL,只需对TCP协议进行简单修改便可在单轮数据传输和多轮数据传输的应用场景下获得较高的有效吞吐率,对数据中心的有效运行具有重大意义.

由于WRED协议和RED协议类似,对参数的设置比较敏感,文中使用大量实验的方式寻找了不同场景下的最佳阈值,并得出了一些结论. 在后续的研究中,我们将进一步研究自适应的阈值设置方式,用于在不同的场景下自动计算最佳阈值.

参考文献

[1] 宋杰,孙宗哲,等. MapReduce大数据处理平台与算法研究进展[J]. 软件学报,2017,28(3):514-543.
SONG Jie,SUN Zong-zhe,et al. Research advance on MapReduce based big data processing platforms and algorithms[J]. Journal of Software,2017,28(3):514-543.

[2] 梁俊杰,李凤华,等. MapReduce框架下的优化高维索引与KNN查询[J]. 电子学报,2016,44(8):1873-1880.
LIANG Jun-jie,LI Feng-hua,et al. Optimized high-dimensional index and KNN query in MapReduce[J]. Acta Electronica Sinica,2016,44(8):1873-1880. (in Chinese)

[3] Vasudevan V,Phanishayee A,Shah H,et al. Safe and effective fine-grained TCP retransmissions for data center communication[J]. ACM SIGCOMM Computer Communication Review,2009,39(4):303-314.

[4] Yadav G H K,Madhavi K. Response time-based resource allocation according to service level agreements in cloud computing[J]. International Journal of Internet Technology and Secured Transactions,2019,9(4):537-546.

[5] 叶进,李陶深,葛志辉. 数据中心网络基于优先级拥塞控制的最后期限可感知TCP协议[J]. 软件学报,2015,26(S2):61-70.
YEJin,LI Tao-shen,Ge Zhi-hui. Priority-based congestion control scheme for deadline-aware TCP in data center networks[J]. Journal of Software,2015,26(S2):61-70. (in Chinese)

[6] 郭丽娜. 数据中心网络TCP Incast仿真分析[J]. 计算机工程与设计,2015,36(01):60-64.
GUO Li-na. Simulation analysis of TCP Incast in data center networks[J]. Computer Engineering and Design,2015,36(01):60-64. (in Chinese)

[7] 黄家玮,徐文茜,等. 数据中心网络中一种基于ECN的TCP慢启动拥塞控制策略[J]. 电子科技大学学报,2018,47(02):169-177.

HUANG Jia-wei, XU Wen-xi, et al. An ECN-based slow-start of TCP congestion control in data center networks [J]. Journal of University of Electronic Science and Technology of China, 2018, 47(02): 169 – 177. (in Chinese)

[8] 李丹,陈贵海,任丰原,等. 数据中心网络的研究进展与趋势[J]. 计算机学报,2014,37(2):259 – 274.
Li Dan, Chen Gui-hai, et al. Data center network research progress and trends [J]. Chinese Journal of Computers, 2014, 37(2): 259 – 274. (in Chinese)

[9] Alipio M, Tiglao N M, et al. TCP Incast solutions in data center networks: A classification and survey[J]. Journal of Network and Computer Applications, 2019, 146: 102421.

[10] Alizadeh M, Greenberg A, Maltz D A, et al. Data center TCP (DCTCP)[A]. ACM SIGCOMM 2010 Conference [C]. New York: Association for Computing Machinery, 2010. 63 – 74.

[11] Huang Y, Hu B. Enhanced DCTCP to explicitly inform of packet loss[A]. 2015 IEEE International Conference on Communications (ICC)[C]. London: IEEE, 2015. 5511 – 5516.

[12] Xia Y, Wang T, et al. Preventing passive TCP timeouts in data center networks with packet drop notification[A]. 2014 IEEE 3rd International Conference on Cloud Networking (CloudNet)[C]. Luxembourg: IEEE, 2014. 173 – 178.

[13] Cheng P, Ren F, et al. Catch the whole lot in an action: Rapid precise packet loss notification in data center[A]. 11th {MSENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 14)[C]. Seattle: {USENIX} Association, 2014. 17 – 28.

[14] Shukla S, Chan S, Tam A S W, et al. TCP PLATO: Packet labelling to alleviate time-out[J]. IEEE Journal on Selected Areas in Communications, 2013, 32(1): 65 – 76.

[15] Abdelmoniem A M, Bensaou B. Curbing timeouts for TCP-Incast in data centers via a cross-layer faster recovery mechanism[A]. IEEE INFOCOM 2018-IEEE Conference on Computer Communications[C]. Hawaii: IEEE, 2018. 675 – 683.

[16] 胡圣. 基于 SDN 的 TCP Incast 问题解决方案[D]. 武汉:华中科技大学,2015.

[17] 万文凯,汪海涛,等. 移动网络中重传超时问题的研究[J]. 软件,2017,38(12):29 – 36.
WAN Wen-kai, WANG Hai-tao, et al. Research on retransmission timeout over mobile data networks[J]. Computer Engineering & Software, 2017, 38(12): 29 – 36. (in Chinese)

[18] Dukkipati N, Refice T, Cheng Y, et al. An argument for increasing TCP's initial congestion window[J]. ACM SIGCOMM Computer Communication Review, 2010, 40(3): 26 – 33.

[19] Vasudevan V, Phanishayee A, Shah H, et al. A (In) cast of Thousands: Scaling Datacenter TCP to Kiloservers and Gigabits, CMU-PDL-09-101 [R]. Pittsburgh: Carnegie Mellon University, 2009.

[20] Custura A, Secchi R, Fairhurst G. Exploring DSCP modification pathologies in the internet[J]. Computer Communications, 2018, 127: 86 – 94.

作者简介

王　娟　女,1993 年生于四川内江. 四川师范大学计算机科学学院硕士研究生. 主要研究方向为数据中心网络优化.
E-mail:535165469@qq.com

夏　羽(通信作者)　男,1983 年生于云南昆明. 现为四川师范大学计算机科学学院讲师、硕士研究生导师. 主要研究方向:高性能交换、数据中心网络.
E-mail:rainsia@163.com