第3章

基于带宽利用率估计的TCP延迟更新模块研究

针对网络中增加传输时延的路由过度缓存问题，本章设计了一个 TCP 延迟更新模块，该模块可与其他 TCP 结合使用，具有较强的适应性。通过估计带宽利用率和测量时延抖动共同预测拥塞的临界点，增强了拥塞预测的准确性。在接近拥塞临界点前，适当减小拥塞窗口，或暂停窗口更新。仿真实验在 Cubic 中实现了 TCP 延迟更新模块，结果显示该模块能够在保证全网传输效率的同时，减少丢包，降低排队时延。

3.1 引言

路由过度缓存问题是由于路由缓存过大和 TCP 基于丢包的拥塞控制机制共同造成的。路由中过多的缓存数据会显著增加网络时延，很难保证对时延敏感的网络应用的需求。

正如第 2 章中综述的，解决过度缓存的方法主要有两类，一是采用 AQM 技术，通过与 FIFO 队列不同的调度和丢包方式，降低排队时延；二是采用端到端的方法，即使用优先级低于尽力而为服务的拥塞控制机制。AQM 技术需要中间路由的支持，短期内无法应用于互联网。而现有的低优先级拥塞控制机制由于公平性问题，也不适用于当前的网络中。本章选择改进 TCP 以缓解过度缓存问题，这是因为端到端的解决方法相比基于路由的方法，更加方便可行。

为了能有效缓解网络中的过度缓存问题，同时不影响网络整体的传输性能，本章设计并基于 Cubic 实现了一个 TCP 延迟更新模块，该模块通过引入带宽利用率和时延抖动共同预测拥塞的临界点。在即将发生丢包前，适当减小拥塞窗口，或延迟窗口更新，以避免加剧拥塞，减少不必要的丢包和排队时延。而网络不拥塞时则依照原有协议进行窗口更新。模块只有在探测到拥塞时才被启用，并不影响正常的窗口更新，更不会因为时延估计的误差导致公平性问题。仿真实验结果表明，延迟更新模块与原有协议共同实现拥塞控制，能够在保证整体传输效率的同时，有效减少丢包，降低排队时延，同时也具备较好的公平性和 TCP 友好性。

3.2 TCP 延迟更新概述

本节主要阐述 TCP 延迟更新方法的基本原理。由于 TCP 流量具有自相似性，即 TCP

流在开始时会采用激进的方式争夺带宽，当大量流进入后，瓶颈链路开始出现拥塞，链路上的所有流会几乎同时感知到丢包，但此时感知到丢包至少已滞后一个 RTT 的时间，其间的分组可能已丢失。如果能够提前感知到网络中即将丢包，然后适当减小拥塞窗口或者保持当前拥塞窗口大小，便可缓解拥塞，减少丢包，同时保证一定的带宽利用率。

本节设计了一个 TCP 延迟更新模块，可与现有的拥塞控制算法相结合，周期性地对网络拥塞进行预测。在探测到严重拥塞，即将丢包时，直接减小拥塞窗口；轻度拥塞时，保持当前拥塞窗口；而网络不拥塞时，使用原有的拥塞控制方法进行窗口更新。图 3.1 为使用 TCP 延迟更新模块后的拥塞窗口演变模型（以 Cubic 为例）。新协议根据估计的网络带宽利用率和时延抖动开始探测可用带宽，当探测到有轻度拥塞时，将启用 TCP 延迟更新模块，暂停更新窗口，即保持当前窗口大小一段时间，然后继续更新窗口、探测拥塞。当出现丢包并进行快速重传和恢复后，TCP 延迟更新模块重置相应的变量，并重新开始探测网络拥塞。在探测到网络严重拥塞时，TCP 延迟更新模块会将窗口适当减小，并保持一段时间，直到探测网络不拥塞时，恢复窗口更新。

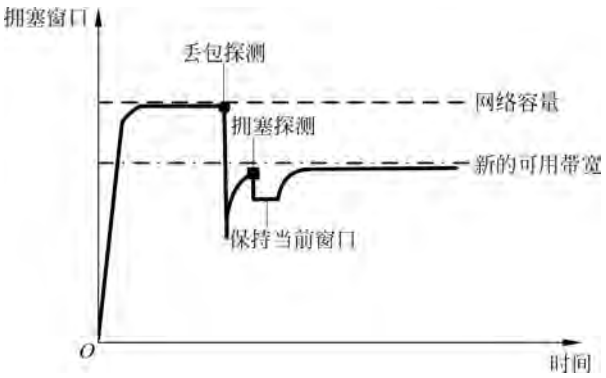

图 3.1　拥塞窗口演化模型（以 Cubic 为例）

3.3　TCP 延迟更新模块

本节将详细阐述 TCP 延迟更新模块的原理。模块主要包括两部分：网络拥塞的预测和窗口的更新控制。如图 3.2 所示，延迟更新模块周期性地预测网络拥塞情况，若预测到网络拥塞，则根据拥塞程度对窗口进行控制；若未发现网络拥塞，则进行正常的窗口更新。也就是说，模块只在预测到网络发生拥塞时会被调用，而在其他情况下并不影响原有协议的正常运行。

图 3.2　TCP 延迟更新模块的框架

3.3.1　基于带宽利用率的网络拥塞预测算法

与现有的拥塞控制方法不同，为了能够更准确地预测网络拥塞，模块采用多位控制信号进行预测，即带宽利用率和时延抖动，通过这两个关键参数估计可能遭遇的拥塞，提早控制速率，避免大量丢包，降低排队时延。模块每隔一段时间（记为 period）进行一次估计，并计算上一次估计至今的带宽利用率和相邻两次估计之间的时延抖动。带宽利用率和时延抖动的计算方法如下：

1. 带宽利用率 U

基于时延的拥塞控制方法（如 TCP Vegas）通过对短期排队时延的估计来探测网络拥塞（即直接使用当前 RTT 以及链路上的最小 RTT 参与窗口的更新），能够及早感知网络拥塞，然而由于时延的估计存在误差，不能很好地反映网络拥塞情况。延迟更新模块并不根据排队时延的短期变化来指示拥塞，而是通过计算带宽利用率 U 来探测排队时延在一段时间内的总体变化趋势，于是可以降低个别误差值对拥塞估计的影响。模块通过式(3.1)来估计一定周期内瓶颈链路上的带宽利用率。

$$U = 1 - \frac{\text{noncongested\_num}}{\text{total\_num}} \tag{3.1}$$

式中，noncongested_num 表示周期内经历的 RTT 等于链路上的最小 RTT（min_RTT）的分组数，total_num 表示收到的分组总数。从式(3.1)可看出，只要收到的分组数足够多，那么少量时延估计的误差并不会对带宽利用率的估算产生太大的影响，因此能够保证拥塞估计的准确性。

2. 时延抖动 jitter

时延抖动反映了链路中时延的变化程度。模块对每次探测周期中所有传输的分组所经历的 RTT 求平均值。将相邻两次探测周期的平均 RTT 分别记为 ave_RTT 和 last_ave_RTT。模块利用相邻两次探测过程中计算的平均 RTT 之差作为时延抖动，即

$$\text{jitter} = \text{ave\_RTT} - \text{last\_ave\_RTT} \tag{3.2}$$

3.3.2　窗口更新控制方法

模块利用带宽利用率和时延抖动共同估计网络的状态，预测网络拥塞程度。根据网络拥塞程度调用不同的窗口更新方法。当 $0.99 < U < 1$ 且 $\text{jitter} > 0$ 时，认为网络拥塞，于是暂时中断原有拥塞控制方法的窗口更新策略，保持当前的拥塞窗口大小；当 $U = 1$ 时，模块认为网络严重拥塞，将拥塞窗口适当减小，并启动定时器 suspend_timer，保持这个窗口一段时间，直到定时器超时，之后恢复原有的窗口更新策略。当 $U \leqslant 0.99$ 或 $\text{jitter} \leqslant 0$ 时，模块认为网络已不拥塞，此时恢复原有协议的窗口更新策略。

3.3.3　TCP 延迟更新模块的具体方案

图 3.3 给出了 TCP 延迟更新模块的具体流程。模块以 period 为周期对网络拥塞进行预测，使用 suspend_flag 和 flag 标志来同时标示当前网络是否拥塞。其中 suspend_flag 的值由 3.3.1 节中定义的带宽利用率和时延抖动共同决定。当网络负载较重，即利用率 $U > 0.99$ 且 $\text{jitter} > 0$ 时，设置 suspend_flag 为 1，尤其是当 $U = 1$ 且 $\text{jitter} > 0$ 时，说明网络可能面临严重的拥

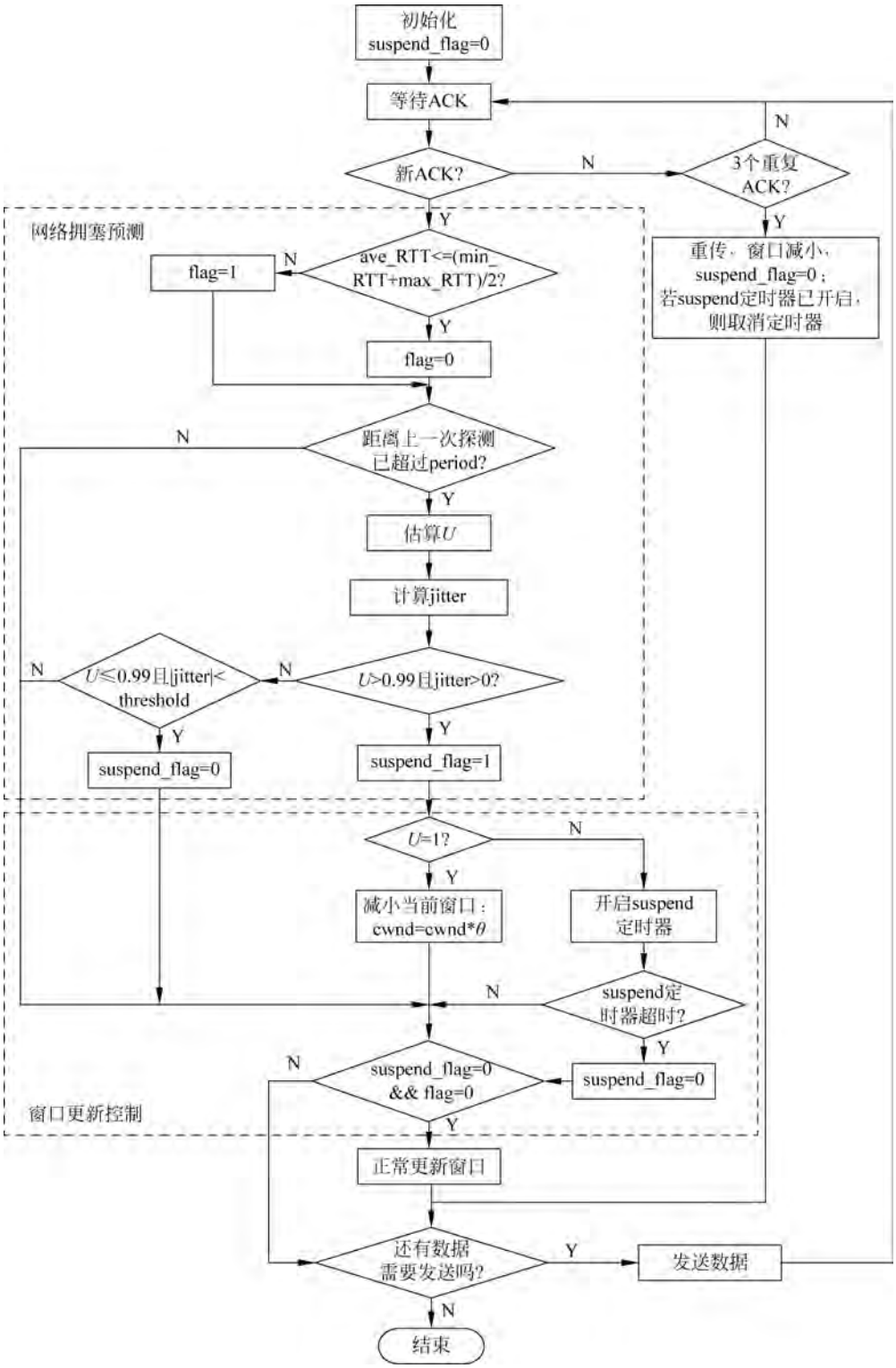

图 3.3　TCP 延迟更新方案流程图

塞，此时虽然还未检测到丢包，但为了避免加剧网络拥塞，造成大量丢包，于是模块提前将拥塞窗口减小为当前窗口的 θ 倍，同时开启 suspend 定时器；当网络负载变轻，即 $U \leqslant 0.99$ 且 $|\text{jitter}| < \text{threshold}$ 时，设置 suspend_flag 为 0。flag 用来辅助 suspend_flag 实现拥塞程度的估计，通过计算周期内的平均 RTT(即 ave_RTT)，并与最大 RTT(即 max_RTT)、最小 RTT(即 min_RTT)相比较来设置 flag 的值，即当 ave_RTT≤(min_RTT+max_RTT)/2 时，说明拥塞并非很严重，设置 flag 为 0，否则设置 flag 为 1。只有当 suspend_flag 和 flag 同时为 0 时，才意味着网络已不拥塞，此时可根据原来的协议进行正常的窗口更新，否则说明网络有一定程度的拥塞，因此需要暂停窗口更新，保持当前窗口进行数据发送。

suspend 定时器一旦超时，suspend_flag 就恢复为 0，以保证在网络负载减小时能够使用正常的窗口更新策略及时探测到可用带宽，从而有效利用带宽资源。此外，当检测到丢包时，若 suspend 定时器已开启，则取消定时器，同时设置 suspend_flag 为 0，重新开始探测拥塞。suspend 定时器时间的设置会影响到模块的性能，若时间设置过长，则可用带宽增加时，模块可能无法及时感知，而使窗口较长时间处于较小的值，降低了带宽利用率；若时间设置过短，则可能无法有效控制窗口，达不到缓解进一步拥塞的作用。因此模块引入一个随机数对定时器进行设置，同时保证时长不超过会话过程中测量到的最小 RTT，具体设置如下：

$$\text{suspend\_timer} = \text{min\_RTT} \times \text{rand}, \quad \text{rand} \sim U(0,1] \tag{3.3}$$

式中，rand 为服从均匀分布的随机数。

3.4　仿真实验结果

Cubic TCP 具有扩展性好、稳定性高等特点，已作为默认的 TCP 版本广泛用于最近的 Linux 内核版本(Linux2.6.18 版本之后)中。此外，最近对网络中 5000 个最流行的 Web 服务器的研究显示，在这些服务器所使用的 TCP 版本中，TCP Reno 占 16.85%～25.58%，BIC/Cubic 占 44.5%，HSTCP/CTCP 占 10.27%～19%[1]。

TCP 延迟更新模块可作为一个单独的模块用于不同的端到端拥塞控制协议中，为了验证模块的有效性，本节基于仿真软件 OPNET Modeler[2] 在 Cubic 中实现延迟更新模块(记为 Cubic+)，并与 Cubic 的性能进行比较。实验对比了不同网络场景以及不同大小的路由缓存下的带宽利用率、丢包数、公平性、友好性等性能。

3.4.1　实验拓扑

实验采用如图 3.4 所示的哑铃状拓扑结构，为了充分验证 TCP 延迟更新模块在广泛的网络环境中的有效性，本节将针对两种网络环境进行实验，即低带宽、低时延网络(瓶颈链路带宽为 2Mbps，往返时延 RTT 为 20ms)和高带宽、长时延网络(瓶颈链路带宽为 400Mbps，往返时延 RTT 为 120ms)。在低带宽、低时延网络环境中，链路的 BDP 仅为 500B，收发双方的接收缓存设为 1MB，实验中考虑远大于 BDP 的缓存 500KB 和接近于 BDP 的 50KB。高带宽、长时延网络中的链路 BDP 为 6MB，收发双方的接收缓存设为 6MB，实验中考虑在 BDP 之内的 1500KB(约为 1000 个分组)和 300KB(约为 200 个分组)。分组大小设为 1500B。每个仿真场景持续 200s。

值得注意的是，模块中拥塞估计的周期应该与拥塞控制算法中的任何定时器相互独立，也就是说，这些时间点与拥塞控制算法无关。互联网测量报告[3]中曾指出，75%～90%的数据流的 RTT 不超过 200ms，根据图 3.4 的拓扑结构，将低带宽、低时延网络场景中的估计周期(period)设置为 100ms，threshold 设为 30ms；将高带宽、长时延网络场景中的估计周期设置为 200ms，threshold 设为 0.6ms。两种场景中均设置参数 $\theta=0.9$。

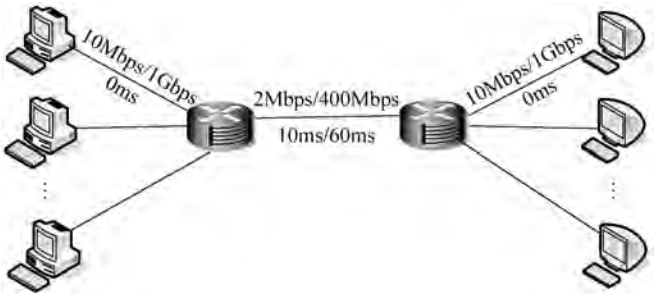

图 3.4　仿真实验拓扑图

3.4.2　实验结果与分析

1. 单条流的实验结果

首先比较网络中只有一条流时 Cubic 和 Cubic+的吞吐率、丢包率、缓存使用率以及每个包的平均排队时延，实验结果见表 3.1。

表 3.1　单条流的实验结果

(a) 低带宽、低时延场景				
缓存大小	500KB		50KB	
协议	Cubic	Cubic+	Cubic	Cubic+
平均吞吐率/Mbps	1.9992	1.9988	1.9992	1.9988
丢包率/%	3.27	0	0.52	0
缓存使用率/%	81.68	1.74	67.14	17.38
平均排队时延/s	1.61	0.03	0.13	0.03
(b) 高带宽、长时延场景				
缓存大小	1500KB		300KB	
协议	Cubic	Cubic+	Cubic	Cubic+
平均吞吐率/Mbps	368.67	348.87	203.93	285.68
丢包率/%	0.0284	0.0298	0.2187	0.0537
缓存使用率/%	0.11	0.11	1.81	0.56
平均排队时延/ms	0.13	0.13	0.42	0.13

从表 3.1 可见，在两种不同场景中，不同的缓存大小下，Cubic 和 Cubic+均实现了较高的带宽利用率，Cubic+的平均吞吐率稍低于 Cubic。

表 3.1(a)中显示，在低带宽、低时延场景中，当路由缓存为 500KB(远大于链路的 BDP)时，Cubic 的丢包率高达 3.27%，而 Cubic+没有产生任何丢包。在缓存使用率中，Cubic 几乎占用了整个缓存，达到 81.86%，而 Cubic+占用的缓存仅为 1.74%，同时平均排队时延与

Cubic 相比，也减少了 98.14%。这是由于 Cubic 仅以丢包作为拥塞指示，在感知到丢包时才将拥塞窗口减小，此时大量分组可能已经丢失，同时缓存中充斥了大量分组，大大增加了分组的排队时延，出现了明显的过度缓存问题。而 Cubic＋利用带宽利用率和时延抖动及时估计网络拥塞，虽然还未感知到丢包，但根据拥塞程度不同，或者将拥塞窗口减小，或者暂停窗口更新，保持原窗口一段时间，因此大大减少了丢包，同时保证了较低的排队时延。

当路由缓存为 50KB(接近于链路的 BDP)时，Cubic 能够更早地探测拥塞，因此仅出现少量丢包，而 Cubic＋仍未出现丢包。与缓存为 500KB 时相比，Cubic 的缓存使用率有所下降，但仍然超出了可用缓存的一半，而 Cubic＋的缓存使用率仅为缓存的 1/3 左右。同时，Cubic 的平均排队时延也大大减少，但仍然比 Cubic＋高出 3 倍多。

表 3.1(b)显示，在高带宽、长时延场景中，由于瓶颈链路带宽较大，当网络中只有单条流存在时，没有出现明显的过度缓存现象，因此 Cubic 和 Cubic＋的性能相差不大。当缓存较大时，由于 Cubic＋不如 Cubic 激进，因此其吞吐率稍低于 Cubic，但仍然达到了 87%的带宽利用率。在缓存较小时，Cubic＋的丢包率还不到 Cubic 的 1/4，平均排队时延也不及 Cubic 的 1/3，由于较高的丢包率导致拥塞窗口频繁减小，Cubic 的吞吐率低于 Cubic＋。

图 3.5 为在高带宽、长时延网络场景中，Cubic 和 Cubic＋单条流的拥塞窗口变化曲线，可以看出，当路由缓存为 300KB 时，Cubic 的拥塞窗口和吞吐率出现了剧烈抖动，这是因为 Cubic 不断以增加传输速率来探测可用带宽，因此极易产生丢包，致使拥塞窗口频繁减小；而 Cubic＋的拥塞窗口最大值虽然稍低于 Cubic，但窗口抖动却没有 Cubic 明显。这是因为：①Cubic＋采用带宽利用率估计来预测拥塞，虽然出现了丢包，但并不影响估计的准确性；②在探测到网络拥塞时，Cubic＋就暂时停止窗口更新，保持当前窗口一段时间，直到网络拥塞有所缓解才开始正常的窗口更新。如图 3.5(b)中 60～80s 以及 180～200s 之间的窗口基本保持在平稳状态，避免了大量丢包，从而实现了比 Cubic 更高的平均吞吐率。当路由缓存为 1500KB 时，Cubic 和 Cubic＋在经过一次丢包调整后很快达到了稳态。Cubic 的拥塞窗口一直增加，但受到接收窗口的限制，所以大小保持在 6MB，吞吐率也实现了满带宽利用率。Cubic＋在接近满带宽时，估计到的带宽利用率和时延抖动增加，因此窗口进行了适当的调整，吞吐率也在可用带宽附近出现了轻微的抖动。

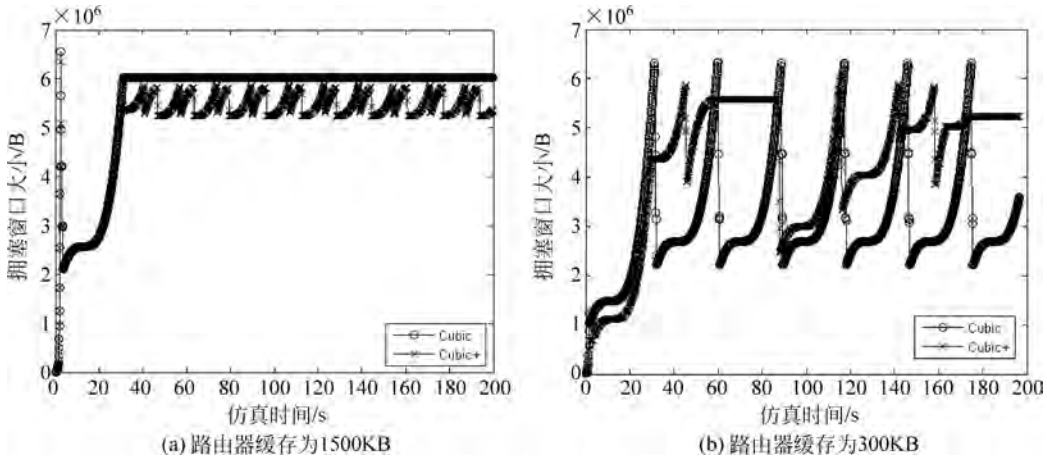

图 3.5　拥塞窗口变化(高带宽、长时延场景)

2. 两条相同流的实验结果

考虑网络中有两条相同流的情况，此时两条流的行为会相互影响。先考察两条流同时进入网络的情况，即两条流均使用 Cubic 或 Cubic+，且具有相同的 RTT。图 3.6 对比了低带宽、低时延场景中 Cubic 和 Cubic+的平均吞吐率和丢包率。从图中可看出，无论路由缓存为 50KB 还是 500KB，Cubic+的平均吞吐率均稍低于 Cubic，而 Cubic+的丢包率也以更大的比例低于 Cubic，尤其是当路由缓存为 500KB 时，Cubic+的丢包率为 0。

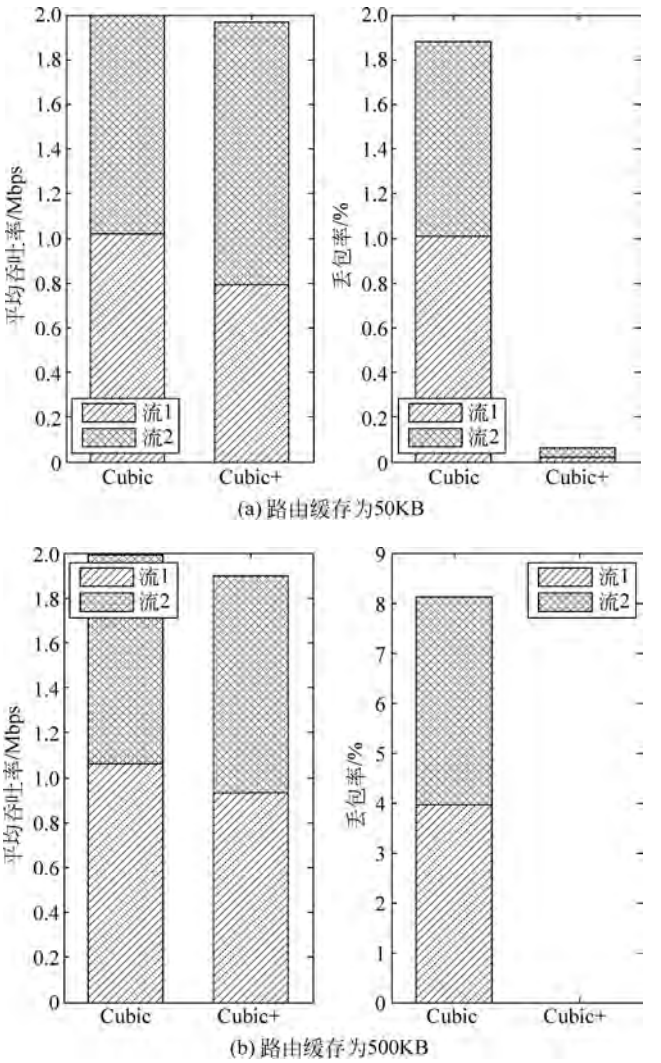

图 3.6 低带宽、低时延场景中的平均吞吐率和丢包率对比

图 3.7 对比了高带宽、长时延场景下 Cubic 和 Cubic+的平均吞吐率和丢包率。从图中可见，Cubic+的总吞吐率仍然稍低于 Cubic。此时，虽然丢包率比低带宽、低时延场景均有所下降，但 Cubic+的总丢包率仍然不及 Cubic 的一半。

表 3.2(Ⅰ)和(Ⅱ)分别给出了两种场景下，同时存在于网络中的两条流的平均吞吐率、丢包率、缓存使用率以及链路中平均排队时延的具体结果。

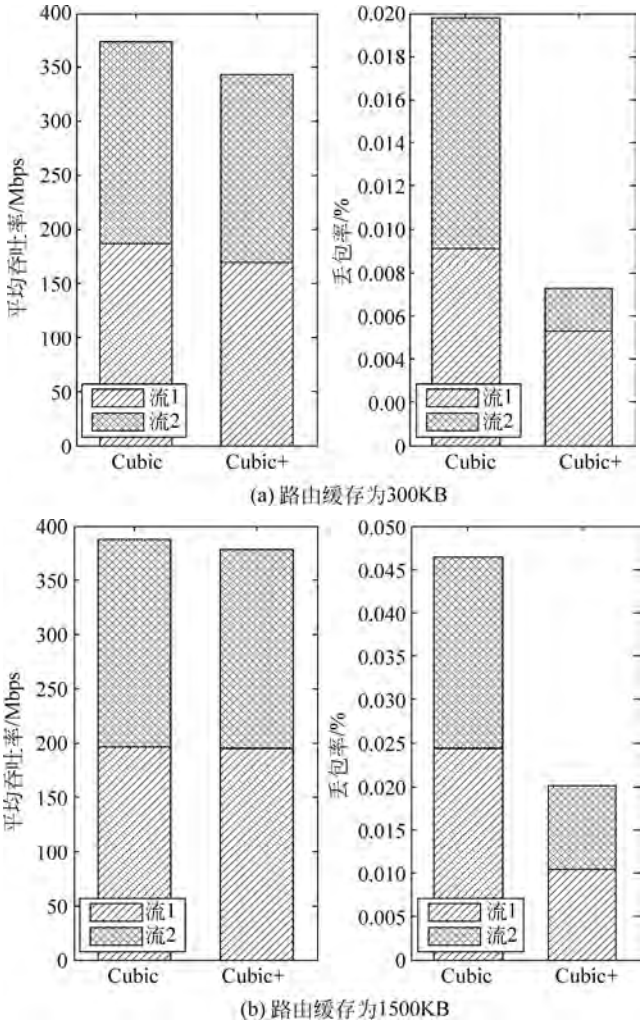

图 3.7　高带宽、长时延场景中的平均吞吐率和丢包率对比

从表 3.2(Ⅰ)中可以看出，在低带宽、低时延场景中，不论路由器缓存为 500KB 还是 50KB，总体上 Cubic+的平均吞吐率稍低于 Cubic，即 Cubic+的带宽利用率稍逊于 Cubic，那是因为 Cubic+在感知到网络拥塞时，就会停止窗口增长，甚至提前减小窗口，降低发送速率，这使得 Cubic+不如 Cubic 激进，因此当网络中存在其他流时，Cubic+的带宽争抢能力不如 Cubic。但也正因为此，Cubic+不会增加网络的拥塞程度，也不会产生更多的丢包。当路由缓存为 50KB 时，由于缓存较小，两条流在竞争过程中均产生了丢包，可是 Cubic+的丢包率与 Cubic 相比，明显较低，且缓存的使用率还不到 Cubic 的 1/3，因此平均排队时延也仅是 Cubic 的 1/3 左右。当路由器缓存增加到 500KB 时，Cubic 会比缓存较小时更晚感知到拥塞，即发现丢包时的窗口值会更大，也就意味着可能产生更多的丢包。同时，缓存的增加也导致 Cubic 排队时延增大。然而，Cubic+的丢包率和排队时延基本保持不变，并未因为缓存增加而增加。从实验结果可以看出，此时 Cubic 的丢包率有所增加，两条流的总丢包率达到了 8.12%，而 Cubic+两条流的丢包数均为 0。Cubic 两条流对缓存的平均使用率达

到了82.08%，而Cubic+仅为17.41%，因此导致Cubic的平均排队时延为Cubic+的40倍还多。

从表3.2(Ⅱ)中可以看出，由于高速网络中的链路不像低速网络容易拥塞，因此两条流共存时，其缓存使用率的变化并不如低带宽中的明显。虽然Cubic+的吞吐率不如Cubic，但Cubic+的总体性能仍然高于Cubic。随着缓存的增大，Cubic和Cubic+的丢包率和排队时延均有所增加。但无论缓存为300KB或1500KB，Cubic+总能实现较低的丢包率和较短的平均排队时延。当缓存为300KB时，Cubic+的平均排队时延仅为Cubic的7.14%。当缓存增加为1500KB时，所有流的平均吞吐率、丢包率和平均排队时延都有所增加，但Cubic+的平均排队时延却仅为Cubic的1/20左右。这仍然是源于Cubic+中准确的拥塞感知特性和对窗口的延迟更新机制。

表3.2 两条流同时进入网络的实验结果

(Ⅰ)低带宽、低时延场景

(a) 路由器缓存为50KB

协　　议	Cubic1	Cubic2	Cubic+1	Cubic+2
平均吞吐率/Mbps	1.0191	0.9805	0.79	1.18
丢包率/%	1.01	0.87	0.02	0.04
缓存使用率/%	65.86		17.20	
平均排队时延/s	0.13		0.04	

(b) 路由器缓存为500KB

协　　议	Cubic1	Cubic2	Cubic+1	Cubic+2
平均吞吐率/Mbps	1.0631	0.9269	0.9335	0.9632
丢包率/%	3.97	4.15	0	0
缓存使用率/%	82.08		17.41	
平均排队时延/s	1.63		0.04	

(Ⅱ)高带宽、长时延场景

(a) 路由器缓存为300KB

协　　议	Cubic1	Cubic2	Cubic+1	Cubic+2
平均吞吐率/Mbps	186.85	186.69	169.53	174.28
丢包率/%	0.0091	0.0107	0.0053	0.0020
缓存使用率/%	15.14		1.09	
平均排队时延/ms	0.98		0.07	

(b) 路由器缓存为1500KB

协　　议	Cubic1	Cubic2	Cubic+1	Cubic+2
平均吞吐率/Mbps	195.78	191.52	195.07	183.21
丢包率/%	0.0243	0.0221	0.0104	0.0097
缓存使用率/%	7.33		0.37	
平均排队时延/ms	2.36		0.12	

然后考察两条流不同时进入网络的情况，即第一条流在0s开始传输数据，直到仿真结束，而第二条流在第15s开始传输15MB(低带宽、低时延场景)/2GB(高带宽、长时延场景)的数据。图3.8和图3.9给出了两种场景下Cubic和Cubic+的丢包率对比。从图中可见，

与 Cubic 相比，Cubic＋在大多数情况下均能够降低网络中的丢包率，尤其是在低带宽、低时延场景中。具体实验结果见表 3.3，在低带宽、低时延场景中，当路由缓存为 50KB 时，Cubic＋除了保持较少的丢包以及较低的排队时延外，短流完成传输所需的时间也比 Cubic 少 6s。虽然 Cubic 有较好的带宽利用率，但是由于丢包太多，一部分带宽被用于重传，浪费了带宽资源，而且分组在路由缓存中排队时延增长，从而导致同样大小的数据需要更长的时间才能传完。在路由缓存为 500KB 的情况下，两种协议的丢包率和平均排队时延均比缓存为 50KB 时有所增加，因此短流的完成时间也相应增加，但 Cubic＋中短流的传输时间仍然比 Cubic 少 12s。

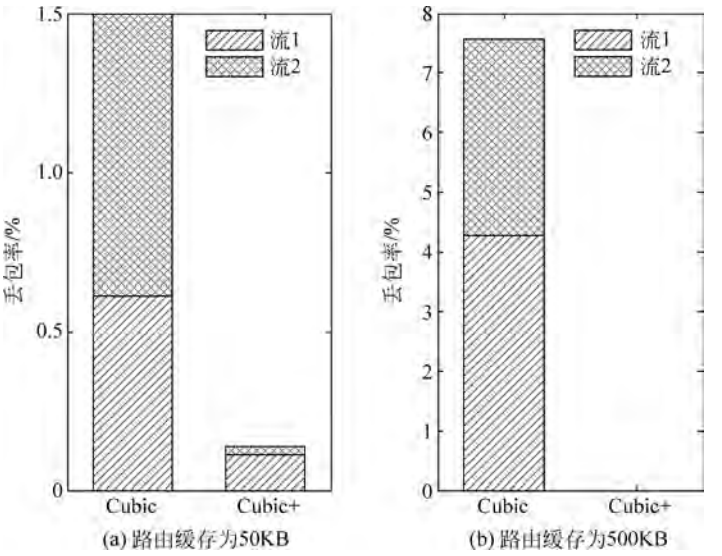

图 3.8　低带宽、低时延场景中的丢包率对比

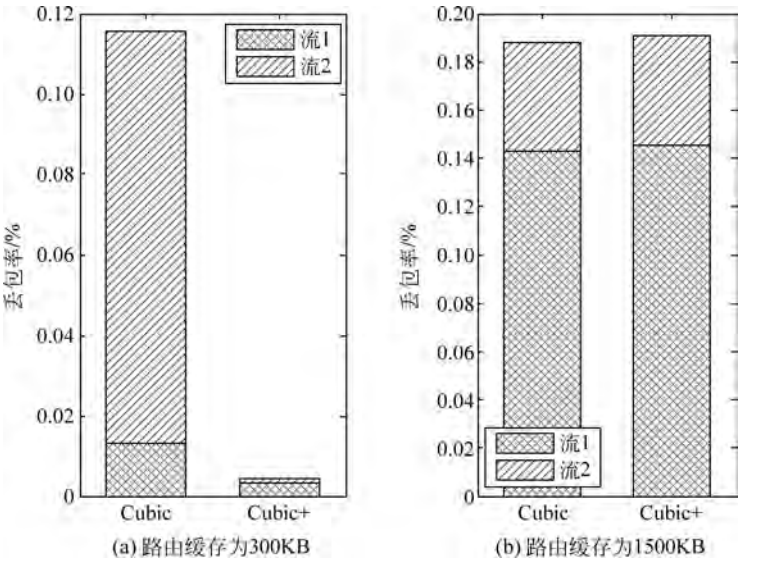

图 3.9　高带宽、长时延场景中的丢包率对比

表 3.3 两条流先后进入网络的实验结果

（Ⅰ）低带宽、低时延场景

（a）路由器缓存为 50KB

协　　议	Cubic1	Cubic2	Cubic+1	Cubic+2
丢包率/%	0.61	0.89	0.11	0.03
平均排队时延/s	0.12		0.04	
短流完成时间/s	118		112	

（b）路由器缓存为 500KB

协　　议	Cubic1	Cubic2	Cubic+1	Cubic+2
丢包率/%	4.27	3.29	0	0
平均排队时延/s	1.59		0.23	
短流完成时间/s	171		159	

（Ⅱ）高带宽、长时延场景

（a）路由器缓存为 300KB

协　　议	Cubic1	Cubic2	Cubic+1	Cubic+2
丢包率/%	0.0131	0.1024	0.0033	0.00098
平均排队时延/ms	0.535		0.038	
短流完成时间/s	110		76	

（b）路由器缓存为 1500KB

协　　议	Cubic1	Cubic2	Cubic+1	Cubic+2
丢包率/%	0.1431	0.0452	0.1456	0.0452
平均排队时延/ms	0.81		0.21	
短流完成时间/s	86		71	

在高带宽、长时延场景中，当路由缓存为 300KB 时，Cubic+的丢包率和平均排队时延均明显少于 Cubic，因此短流的传输时间也比 Cubic 减少了 34s。当路由缓存为 1500KB 时，各条流的丢包率和排队时延均有所增加，此时 Cubic+和 Cubic 的丢包率接近，但 Cubic 的排队时延却约为 Cubic+的 4 倍，因此 Cubic+的短流传输时间仍然比 Cubic 短 15s。

3. 作为背景流的实验结果

接下来考察 Cubic 和 Cubic+分别作为背景流时，对 TCP Reno 和 TCP SACK 传输性能的影响。仿真开始时 Cubic/Cubic+先传输数据，15s 后 TCP Reno/SACK 开始传输 FTP 数据。在低带宽、低时延场景中，当缓存为 50KB 时，前景流传输 5MB，当缓存为 500KB 时，前景流传输 500KB。在高带宽、长时延场景中，前景流均传输 300MB 的 FTP 数据。考察 TCP Reno/SACK 的流完成时间及丢包率，实验结果如图 3.10 和图 3.11 所示。

图 3.10(Ⅰ)给出了在低带宽、低时延场景中的实验结果，路由缓存为 50KB 的情况下，当 Cubic 为背景流时，TCP Reno 和 TCP SACK 的传输时间均为 117s，而当 Cubic+为背景流时，TCP Reno 和 TCP SACK 的传输时间分别为 38s 和 32s，传输时间分别减少了 67.52%和 72.65%。在路由缓存为 500KB 的情况下，虽然前景流传输的数据减少为 500KB，然而 Cubic 为背景流时，Reno 流和 SACK 流传输的时间并未大幅度减小。这主要有两个原因，一是由于 Cubic 的窗口增长激进，抢了 Reno 和 SACK 应该公平共享的带宽；二是这三个协议均为基于

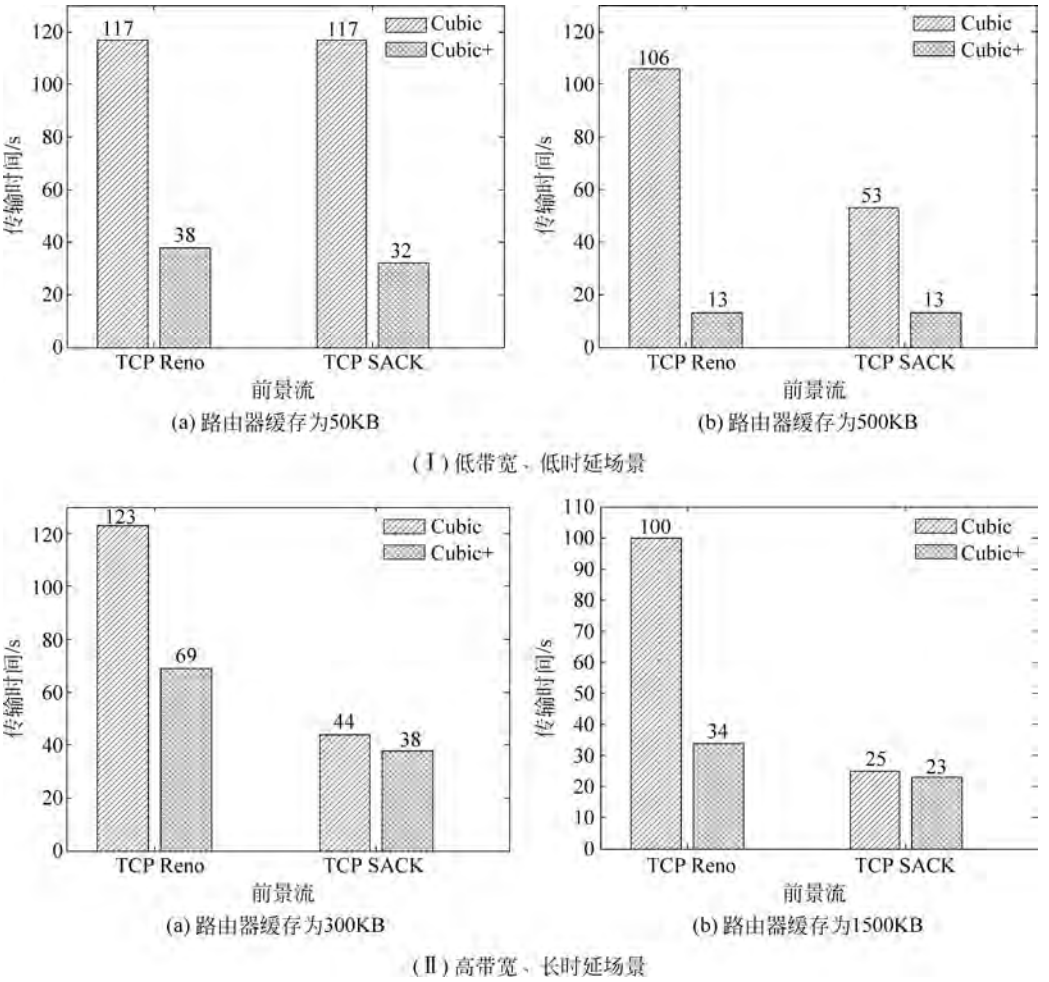

图 3.10　不同背景流下前景流完成时间

丢包的协议，它们的协议机制会导致缓存被占据得满满的，从而增加分组的排队时延，这也是增加流完成时间的原因之一。而 Cubic+在感知到有新的数据流进入网络时，利用时延进行带宽利用率探测，并根据探测结果进行适当的窗口调整，并不会抢占大部分缓存，也不会引入太大的排队时延，从实验结果来看，在缓存为 500KB 时，Cubic 使用的缓存约为 408KB，Cubic+使用的缓存约为 91KB，仅为 Cubic 的 22.37%。而 Cubic 的平均排队时延为 1.62s，Cubic+为 0.36s，也仅为 Cubic 的 22.55%。因此当 Cubic+为背景流时，TCP Reno 和 TCP SACK 的传输时间比 Cubic 为背景流时分别减少了 87.74%和 75.47%。

图 3.10(Ⅱ)给出了在高带宽、长时延场景中的实验结果，在路由缓存为 300KB 的情况下，TCP Reno 和 TCP SACK 的传输时间分别为 123s 和 44s，而当 Cubic+为背景流时，TCP Reno 和 TCP SACK 的传输时间分别为 69s 和 38s，传输时间分别减少了 43.90%和 13.64%，这主要是因为当 Cubic 为背景流时，总的丢包率高于 Cubic+为背景流时。在路由缓存为 1500KB 的情况下，两种背景流下的总丢包率差不多，由于吞吐率增加，不论是 Cubic 还是 Cubic+作为背景流，TCP Reno 流和 TCP SACK 流的完成时间都有所减少，而

Cubic+作为背景流时，排队时延少于 Cubic 作为背景流时，因此这些流的完成时间仍然是最少的。

图 3.11 显示了低带宽、低时延场景中，不同背景流下网络中丢包的情况，从图中可以看出，以 Cubic+为背景流时，网络中总的丢包率远小于 Cubic 为背景流时的丢包率，特别是在缓存较大(500KB)时，Cubic+未产生任何丢包，而 Cubic 的丢包率高达 3.08%(前景流为 TCP SACK 时)。由此可见，Cubic 在作为背景流时，由于过于激进的窗口更新，更易加剧网络拥塞，造成更多的丢包，由此影响到网络中其他流的传输性能，而 Cubic+在估计到网络拥塞时，提前控制发送速率，因此不会加剧网络拥塞，保证丢包较少，一定程度上也就保证了网络中其他流的传输性能，有效利用了带宽资源。

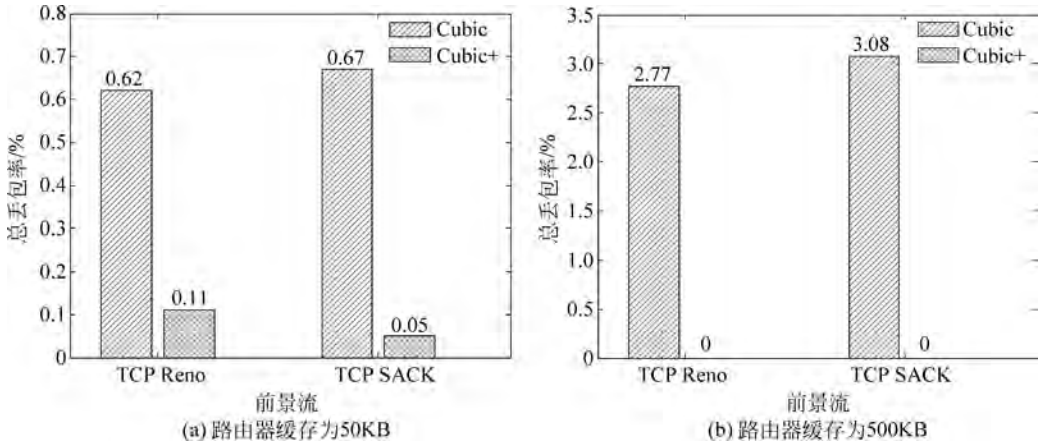

图 3.11 不同背景流下丢包率

4. 公平性

为了评价 Cubic+的公平性，考虑两个不同的场景，即多条流具有相同 RTT 和不同 RTT 的场景，然后使用 Jain 提出的公平指数(fairness index，FI)[4]量化并评价协议的公平性。

在具有相同的 RTT 情况下，两条流同时进入网络，同时停止传输，在表 3.2 中已得到两条流的平均吞吐率，则 Cubic 和 Cubic+的公平性指标见表 3.4。可见，在相同的 RTT 条件下，两种场景中的 Cubic+基本上保持了和 Cubic 一样良好的公平性，甚至在缓存为 500KB 时，Cubic+的公平性还稍高于 Cubic。

表 3.4 公平性比较

	低带宽、低时延场景		高带宽、长时延场景	
缓存/KB	50	500	300	1500
Cubic	0.9996	0.9953	0.9999	0.9999
Cubic+	0.9623	0.9997	0.9998	0.9990

对于不同 RTT 的情况，两种场景下 Cubic+仍然能够实现较好的公平性，这里主要给出低带宽、低时延场景下的结果并进行分析。实验考虑两条 TCP 流共享瓶颈链路，链路带宽仍为 2Mbps，两条流的 RTT 具有不同的比例。其中一条流的 RTT 值为 20ms，另一条流的 RTT 值为 40ms 和 60ms 之一，因此两条流的 RTT 比例分别为 2 和 3。表 3.5 给出了当

缓存分别为 50KB 和 500KB 时的结果，其中 T1 和 T2 分别表示两条流的平均吞吐率，效率指数(efficiency index，EI)是指共同存在的两条流的吞吐率之和，FI 是公平性指数。从表 3.5 可以看出，当路由缓存为 50KB 时，三个协议均实现了较好的 RTT 公平性。当路由缓存增大时，RTT 较小的流能够占据更多的缓存，导致各个流之间更加不公平，因此 Cubic 和 Cubic＋的公平性指标均有所减弱。但 Cubic＋能够很好地控制数据流占用的缓存量，所以仍然实现了比 Cubic 更好的公平性。

表 3.5　EI 和 FI 的仿真结果

(a) 路由器缓存为 50KB

RTT 比例	2				3			
协议	T1	T2	EI	FI	T1	T2	EI	FI
TCP Reno	0.96	1.02	1.98	0.9989	0.86	1.10	1.97	0.9854
Cubic	0.99	1.01	2.00	0.9999	0.87	1.13	2.00	0.9838
Cubic＋	0.94	1.06	2.00	0.9963	0.97	1.03	2.00	0.9990

(b) 路由器缓存为 500KB

RTT 比例	2				3			
协议	T1	T2	EI	FI	T1	T2	EI	FI
TCP Reno	0.96	1.04	2.00	0.9982	0.94	1.06	2.00	0.9959
Cubic	0.71	1.29	2.00	0.9215	0.73	1.27	2.00	0.9325
Cubic＋	0.99	1.01	2.00	0.9998	0.87	1.13	2.00	0.9830

5. 友好性

为了比较 Cubic 与 Cubic＋的 TCP 友好性，实验采用四个发送端，其中两个运行 TCP Reno 协议，而另外两个使用其他相同的 TCP 版本(Cubic 或 Cubic＋)，四条数据流具有相同的 RTT。在不同的场景中，Cubic＋均实现了较好的友好性。

图 3.12 显示了在高带宽、长时延场景中的实验结果。当路由缓存分别为 300KB 和 1500KB 时四条流的平均吞吐率，其中 1 和 2 表示使用新的 TCP 版本的流，而 3 和 4 表示使用 TCP Reno 的流。从图中可看出，在缓存较大时，Cubic 和 Cubic＋实现的友好性比小缓

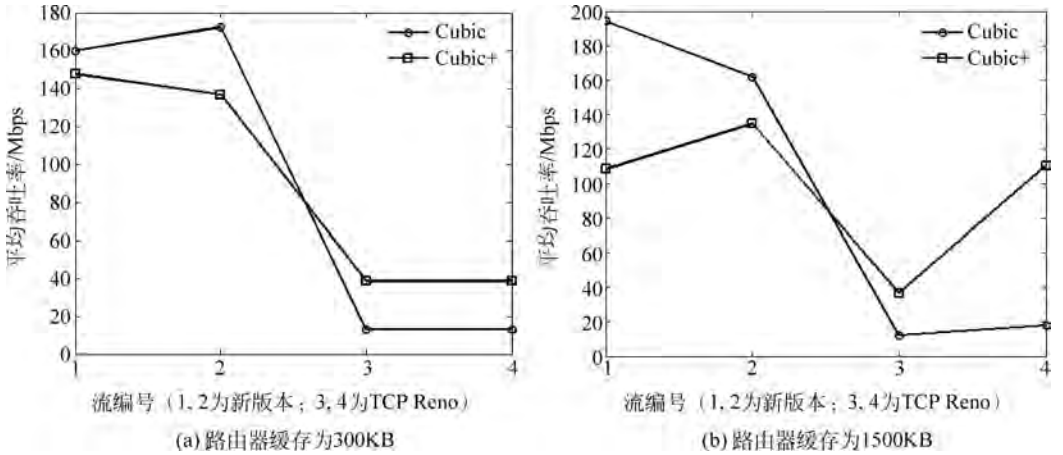

图 3.12　TCP 友好性比较(高带宽、长时延场景)

存下稍好。在两种路由缓存下，Cubic 明显地“偷”了 TCP Reno 的带宽，而 Cubic＋并未抑制与之并存的 TCP Reno 流，从而比 Cubic 实现了更好的 TCP 友好性。这是因为 Cubic＋并不像 Cubic 一样，持续增加拥塞窗口直到丢包，而是通过估计带宽利用率和计算时延抖动来估计拥塞，并适当减小拥塞窗口，因此并未占据所有的路由缓存，于是 TCP Reno 可以利用剩余的缓存，最终 TCP Reno 流可与 Cubic＋流实现较好的带宽共享。

3.5 本章小结

针对导致网络时延增加的路由过度缓存问题，本章提出了一个 TCP 延迟更新模块并基于 Cubic 进行了实现和验证。该模块通过引入带宽利用率和时延抖动共同预测拥塞。根据拥塞程度，适当减小拥塞窗口，或延迟窗口更新，以避免加剧拥塞，减少不必要的丢包并降低排队时延。而网络不拥塞时则依照原有协议进行窗口更新。模块可与目前的 TCP 相结合，具有较强的适应性。仿真实验结果表明，总体上来看，Cubic＋在低带宽、低时延场景中的性能提升较高带宽、长时延场景中的性能提升显著。而在不同的场景下，无论缓存大或小，与 Cubic 相比，使用了模块的 Cubic＋在保证较高吞吐率的同时，大大减少了丢包和排队时延，有效地利用了带宽资源；作为背景流时，Cubic＋能够通过适当调整拥塞窗口，减少丢包，同时不占用大量路由缓存，因此不影响其他流的传输性能，保证其他流能够在较短时间内完成传输；对于公平性，Cubic＋总能在保证整个网络传输性能的同时，保持公平性以及对其他协议的友好性。由上可见，TCP 延迟更新模块可与互联网中的 TCP（特别是基于丢包的协议）结合，缓解网络（尤其是低带宽、低时延网络）中的过度缓存问题。

参考文献

[1] Yang P, Luo W, Xu L, et al. TCP congestion avoidance algorithm identification. In Proc. of 2011 31st International Conference on Distributed Computing Systems (ICDCS), Minneapolis, Minnesota, UA, 2011: 310-321.

[2] OPNET Technologies. http://www.opnet.com/solutions/network_rd/modeler.html, 2003.

[3] Jiang H, Dovrolis C. Passive estimation of TCP round-trip times. ACM SIGCOMM Computer Communication Review, 2002, 32(3): 75-88.

[4] Jain R, Chiu D M, Hawe W. A quantitative measure of fairness and discrimination for resource allocation in shared systems. DEC TR-301, Littleton, MA: Digital Equipment Corporation, 1984.