TFRC 拥塞控制策略的研究和改进①

王传安[1,2]，葛 华[1]，赵海燕[1,2]

[1](安徽科技学院，凤阳 233100)
[2](江苏大学 计算机科学与通信工程学院，镇江 212013)

摘 要：分析了目前在实时业务中常用的拥塞控制机制 TFRC。针对实时流媒体要求低延迟和低抖动的特点，提出将单程传输延迟的抖动作为反馈信号来调节发送端的发送速率，并针对抖动阈值的选定采用了自适应的调节方法。仿真结果表明改进后的算法减小了延迟抖动，提高了相关的服务质量。

关键词：拥塞控制；TCP 友好性；实时业务

Research and Improvement of TFRC Congestion Control Mechanism

WANG Chuan-An[1,2], GE Hua[1], ZHAO Hai-Yan[1,2]

[1](Anhui Science and Technology University, Fengyang 233100, China)
[2](School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China)

Abstract: The congestion control mechanism TFRC, which is widely used in current real-time streaming media, is analyzed in this paper. A proposal aiming at the requirement for short delay and jitter of real-time streaming media has been made to adjust to the transmit rate of the sender to the change of single trip delay jitter. Meanwhile, a dynamic adaptive method is used in the determination of the jitter threshold. The result of simulation indicates that with the improved algorithm the jitters has been reduced remarkably and other related QoS has also been improved.

Keywords: congestion control; TCP friendly; real-time stream

1 引言

随着网络多媒体技术的迅速发展和广泛应用，如何有效地利用有限的网络传输带宽提高传输服务质量成为研究的热点。针对流媒体信息的传输所要求的实时性、连续性、TCP 友好性以及视频传输速率的平稳性，国内外学者提出了众多拥塞控制机制[1-4]，TCP 友好性速率控制(TFRC)作为其中一员因其良好的综合性能而被广泛使用。TFRC 对网络拥塞的响应主要是通过对丢包事件做出反应来实现的，然而丢包事件的发生往往是网络拥塞引起的，这样基于丢包事件率的发送速率的调节具有一定的滞后性。与丢包事件追踪持续的网络拥塞不同，单程延时抖动追踪瞬间的拥塞，对单程延时抖动的测算可以在丢包发生之前检测出网络拥塞。因此，本文提出引入单程延时抖动作为潜在的拥塞信号对发送速率进行调节来改进 TFRC 算法，并用 NS-2 仿真验证了改进的有效性。

2 TFRC 实现

TFRC 是一种用于尽力服务网络环境中的基于速率的拥塞控制机制，设计它主要是为了当网络传输流在和 TCP 流竞争带宽时能够做到适度的公平，即在相同的条件下 TFRC 流的吞吐率不超过 TCP 流的吞吐率[3]，除此之外 TFRC 流还应拥有比 TCP 流更平滑的吞吐率变化，这让它很适合像流媒体这种对平滑发送速率要求很高的应用。RFC3448[2]对此机制进行了较为详尽的描述。

① 基金项目:安徽科技学院引进人才基金(ZRC2010255);安徽自然科学基金(KJ2009B121Z)
收稿时间:2010-06-19;收到修改稿时间:2010-07-15

2.1 TFRC 的基本机制

TFRC 保持对 TCP 流的友好性是通过直接使用 TCP 协议中吞吐率计算公式来实现的，一般来说，它的工作过程分为以下四个步骤：

1) 接受端测量丢失事件率并将其反馈至发送端；

2) 发送端利用收到的反馈包信息测量回环时间；

3) 发送端将丢失事件率和回环时间代入吞吐率公式，计算出当前允许的保证TCP友好性的最大发送速率；

4) 发送端比较计算结果和当前发送速率来决定增大或减小发送速率。

2.2 吞吐率公式

本文采用文献[3]中给出的 TCP Reno 的吞吐率响应公式来作为 TFRC 中速率调整参考的依据：

$$X_{calc}=\frac{s}{R\sqrt{\frac{2bp}{3}}+t_{RTO}\sqrt{\frac{3bp}{8}}p(1+32p^2)} \quad (1)$$

其中，X_f 为计算得到的吞吐率上限(单位：B/s)，s 为数据包的大小(单位：B)，R 为回环时间 RTT 的值(单位：s)，b 为 TCP 接受端一次确认的包数，通常为 1；p 为稳定状态下的丢失事件率，t_{RTO} 为 TCP 重传时钟值，被设置为 4R 和 1s 两者中较大者。

2.3 丢包事件率 p 的测算

接收端测算丢包事件率 p 是 TFRC 算法中最重要的环节之一，主要计算步骤如下[4]：

1) 发现丢失包或有拥塞标记的包；

2) 从历史丢失记录换算到丢失事件；

3) 计算丢包事件间隔；

4) 计算平均丢包事件间隔 I_mean；

5) 计算丢包事件率：p=1/I_mean。

2.4 发送端对发送速率的调整

发送端当前的发送速率为 X,并且持有一个当前回环时间的估计 R 和超时间隔估计 tRTO。当发送端在 t_now 时刻收到一个反馈包时，做以下操作：

1) 计算最新的回环时间取样。

R_sample=(t_now-t_recvdata)-t_delay，其中 recvdata 为最后收到的一个包的时间戳，即此包在发送端被发送的时间，t_delay 为接受到最后一个数据包离产生反馈包间的时延。

2) 更新回环时间估计：

If 之前没收到反馈包

R=R_sample;

Else

R=q*R+(1-q)*R_sample;

3) 更新超时间隔：

tRTO=4*R.

4) 更新发送速率如下：

If (p>0)

用 TCP 吞吐率公式(1)计算 X_calc。

X=max(min(X_calc,2*X_recv), s/t_mbi);

Else If (t_now-tld>=R)

X=max(min(2*X, 2*X_recv),s/R);

tld = t_now;

这里，当 p=0 时，发送端处于慢启动阶段，每个回环时间加倍一次发送速率直到有丢失事件发生为止。X_recv 为接收端估测的从上个反馈包被发送之后的接受数据的接受速率。s/R 给出了发送端在慢启动阶段的最小发送速率，即每个回环时间一个包。参数 t_mbi 为 64 秒，它表示在持续未收到反馈包时相邻包间的最大时间间隔。因此，当 p>0 时，发送端至少每 64 秒发送一个包[4]。

3 TFRC算法的改进

现有的 TFRC 流控机制虽然能根据最近的往返时间的的变化而对发送速率作出相应的修正来减小速率波动和提前避免拥塞，但此措施只是采用了回环时间作为反馈因素，未对单程传输延时加以考虑，事实上，端到端得实时通信不仅对回环时间要求稳定，也对单程传输延时要求稳定[6]。在实时通信业务中画面能否流畅播放很大程度上取决于接受设备能否稳定地接受视频流，当然这可以通过在接受端引入缓存解决，但引入过大的缓冲将损害业务对实时性的要求，以单程延时作为反馈对发送速率作调整比用回环时间作为反馈时能对网络拥塞做出更及时的预测，这在实时通信中更显重要[7]。本文正是通过在接受端以单程传输延时抖动作为实时业务即将发生拥塞的信号，来修正 TFRC 发送端得发送速率。根据 TFRC 协议中定义的反馈包的报文内容，接收端已有足够的信息来计算单程传输延迟，本算法中定义如下计算方法：

R_singletrip(i) = t_fb(i)-t_recvdata(i)-t_delay.

其中，R_singletrip(i)为第 i 个数据包的单程传输延迟；t_fb(i)为接收端对第 i 个接受到得数据发出反馈包的时间；t_recvdata(i)为收到的第 i 个包的时间戳，即此包在发送端被发送时的时间；t_delay(i)为接受端收到第 i 个数据包到发送反馈包的时延。设 D 为两个包的单程传输延时的差值，则第 i 个包和其前一个包的单程传输延时的差值为：

D(i-1, i) = R_singletrip(i)-R_singletrip(i-1)

这样当接收端接受到第 i 个包时，其抖动估计 J(i)可以通过这个差值 D(i-1,i)和第 i-1 个包的抖动估计 J(i-1)用下式计算：

$$J(i) = J(i-1) + (|D(i-1,i)|-J(i-1))/16 \quad (2)$$

这是一个最佳一阶估计量，增益参数 1/16 在实现了较好的降噪比的同时，又维持了合适的收敛速度。

在 TFRC 中，发送端有 4 种速率状态：慢启动，退出慢启动，拥塞避免和减速状态[5]。若没有丢包发生，则进入慢启动状态，速率在每个 RTT 内加倍直到有数据包丢失为止，此时 TFRC 流进入短暂的退出慢启动状态，保持发送速率；然后待由吞吐率公式(1)计算出的速率如前 2.4 节所述对当前发送速率进行减小或增加，这就相应地进入减小速率状态或增大速率的拥塞避免状态。本文对这种速率调整方案进行了修改，即当没有发现丢包事件时首先要利用式(2)对当前的单程延时抖动进行测量，如果发现其超过抖动阈值，这意味着拥塞可能即将发生，则直接进入拥塞避免状态，提前预防网络拥塞。但由于此时并没有丢包，丢失事件率为零，不能用(1)式来修正速率，因此这里提出对发送速率进行直接修正的方法,修改后算法伪代码如下：

```
If(p>0)
   X=max(min(X_calc,2*X_recv), s/t_mbi);
Else if(J(i)<J_threshold&& t_now-tld>=R){
          X=max(min(2*X, 2*X_recv),s/R);
           tld = t_now
       }
      Else if(J(i)>=J_threshold)
            X=X*w ;
```

其中 w 为修正因子，取值范围为[0,1],目的是根据当前单程延时抖动占实时业务允许的最大单程延时抖动的比值来减少发送速率,从而降低波动。记当前单程传输延时抖动占最大单程延时抖动的为 x= J(i)/J_max ,比例因子 w =sqrt(1-x)的函数曲线如图 1 所示。

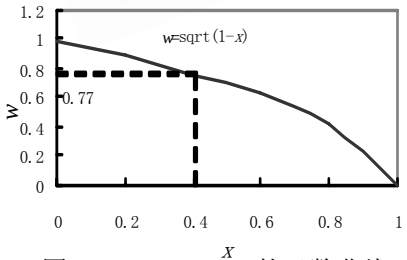

图 1　w = sqrt(1-x)的函数曲线

特别地，当前单程延时抖动超过抖动阈值且占实时业务允许的最大单程延时抖动的 40%时，该算法将发送速率调节为原发送速率的 77%。其中 J_max 为实时业务允许的最大单程延时抖动，在本文的仿真实验中取 RTT/5。这样，只有当抖动超过设定的阈值时才对当前发送速率作如上调整。在此算法中对于阈值 J_threshold 的选取，也会影响传输的效果。J_threshold 选取的太小，传输会对抖动表现的过于敏感，甚至对那些并不预示着随后的拥塞

的较小抖动也作出反应；而 J_threshold 选取的太大，传输则会对抖动表现得不够敏感，最终导致不能有效的预示即将到来的拥塞。由此可见使用单一的阈值门限 J_threshold 并不是所有情况的最佳选择。因此本文采用一种自适应阈值调节方案如下：

(a) 设 J_threshold 的初值为 RTT/10，以步长 0.5ms 进行调整。

(b) 如果系统在丢包前没检测到延时抖动超出 J_threshold，则说明 J_threshold 设置过大，将 J_threshold 下调一个步长。

(c) 如果系统连续 5 次由于延时抖动超出 J_threshold 的值，则说明 J_threshold 设置过小，上调一个步长的 J_threshold 值。

如此根据历史情况来调节 J_threshold 值，让延时抖动更准确地反馈网络的拥塞状况。

4　仿真测试

4.1 TCP 友好性性能分析

下面利用 ns-2 网络模拟器分析本文算法的效果。在模拟实现中，我们采用常用的哑铃拓扑机构如图 2 所示。

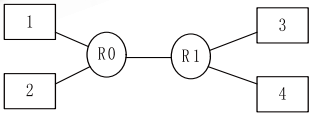

图 2　仿真网络拓扑

图 2 中，R0 和 R1 为路由器，瓶颈带宽 2Mb/s,链路延迟为 20ms,队列调度采用 RED 机制。瓶颈链路以外均以 100Mb/s 带宽和 3ms 延时的链路相连。节点 1 和 3 之间建立 TCP 流，节点 2 和 4 之间先后分别建立改进了的 Improved-TFRC 流和原 TFRC 流。各流数据包的大小均为 1000B，仿真时间为 100 秒。

为了验证改进算法的 TCP 友好性，这里引入友好性比率来对其进行描述，令 Tt 和 Ti 分别为 TCP 流和改进后 TFRC 流的吞吐量，则改进后的 TFRC 流的友

好性比率可定义为

$$Rt=Tt/Ti \quad (3)$$

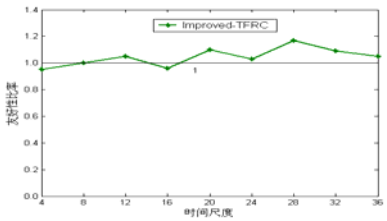

图 3 Improved- TFRC 流的 TCP 友好性比率

图 3 所示为由仿真实验得到的 Improved- TFRC 流的 TCP 友好性比率，根据 TCP 友好性的定义，由其在 1 附近起伏表明 Improved-TFRC 流是 TCP 友好的。

4.2 速率平滑性性能分析

为了验证改进后的 TFRC 算法具有更好的速率平滑性，本文采用 R_Jain 定义的偏差系数[](COV)来描述连接吞吐量的变化率，即平滑系数(SmCo)。设测量的时间尺度为 θ ,TC(t_0,t_0+ θ)为连接 C 在时间段(t_0,t_0+ θ)内发送的数据量(Byte),则连接 C 在时间(t_0,t_0+ θ)的平均吞吐量可定义为

$$\overline{T}_{C,\theta}(t_0)=\frac{T_C(t_0,t_0+\theta)}{\theta} \quad (4)$$

时间[t_0,S]的吞吐量序列为 $\{\overline{T}_{C,\theta}(t_0+i\times\theta)\}_{i=1}^{S-t_0/\theta}$，由此可计算出当测量时间尺度为 s 时，吞吐量的偏差系数为

$$COV_{C,\theta}=\frac{\sqrt{\frac{1}{S-t_0/\theta}\sum_{i=1}^{S-t_0/\theta}[\overline{T}_{C,\theta}(t_0+i\times\theta)-\overline{T}_{C,S}(t_0)]^2}}{\overline{T}_{C,S}(t_0)} \quad (5)$$

其中，$\overline{T}_{C,S}(t_0)$ 为连接 C 在时间[t_0,S]内的平均吐量。偏差系数越小，则连接速率变化就越小，即越平滑。

在以上仿真环境中，在其他参数不变的情况下对节点 2 和 4 之间先后建立本文改进后的 TFRC 流和 TFRC 流进行仿真计算，测得改进后的 Improved-TFRC 与原 TFRC 算法以及 TCP 流的平滑系数如下图 4 所示

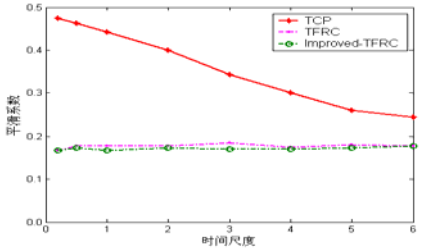

图 4 Improved-TFRC 与 TFRC、TCP 平滑系数比较

由图 4 可见，改进后的算法的平滑系数更小，即其发送速率更平滑，且在不同的时间尺度下趋于稳定。同时，其平滑系数较 TCP 的平滑系数也远小。

下表给出了改进后的算法 Improved-TFRC 与原算法其他相关 QoS 的比较

表 1 Improved-TFRC 流与 TCP 流的比较

	吞吐率均值（Kbps）	平均单程传输延时(ms)	平均单程延时抖动(ms)
Improved-TFRC	193	43.2	6.67
TFRC	186	45.6	7.16

由表 1 可知，与原 TFRC 算法相比，改进后的 Improved-TFRC 算法提高了 4%的吞吐量，同时平均单程传输延时和平均单程延时抖动分别降低了 5%和 7%，这体现了 Improved-TFRC 更好的实时性和传输稳定性。

5 结语

针对 TFRC 流控制机制对单程传输延时抖动这一能预示即将来临的拥塞的有用标示量缺乏控制和利用，本文在研究现有算法的基础上，提出了基于自适应的单程延时抖动阈值的 TFRC 算法的改进。通过仿真实验，验证了改进算法的有效性。接下来的工作重点将放在如何在更复杂的网络环境下解决算法的鲁棒性。

参考文献

1 LAM S. General AIMD Congestion Control. Proc. of IEEE ICNP'00. Osaka, Japan,2000.

2 王庆生,杨新芳,王庆伟.TCP 拥塞控制中一种改进的 AIMD 算法.微计算机信息,2009(30):56－59.

3 Floyd S, Handley M, Padhye J, et al. Equation-based Congestion Control for Unicast Applications: the Extended Version. International Computer Science Institute, Tech. Rep.: TR-00-003, 2000-03.

4 Floyd S, Handley M, Padhye J, et al. TCP Friendly Rate Control (TFRC):Protocol Specification. Network Working Group RFC 3448,2003-01.

5 Widmer J, Denda R, Mauve M. A Survey on TCP-friendly Congestion Control. IEEE Network,2001,15(3):28－37.

6 甘泉,薛质.控制端到端传输延迟抖动的改进 TFRC 算法.计算机工程,2008,34(10):105－107.

7 Lai YC. Taxonomy and Evaluation of TCP-friendly Congestion-control Schemes on Fairness, Aggressiveness, and Responsiveness. IEEE Network, 2007,21(6):6－15.