**TCP友好拥塞控制分析**

朴泉宇

北京理工大学计算机学院07111403班，北京 100081

(1120141831@bit.edu.cn)

#### TCP-Friendly Congestion Control

Quanyu Piao

(Class 07111403,School of Computer Science, Beijing Institute of Technology, Beijing 100081)

**Abstract** Internet keeps developing with many protocols, TCP and UDP are on behalf of them, which are used widely in Internet transportation. But there is a problem said congestion made by stronger use of Internet but hardware is not effective as we think. TCP has congestion control so that it can ensure that transportation will not crash down. But UDP doesn’t. So when TCP and UDP are used in Internet at once, UDP will encroach TCP’s bandwidth but UDP will also crash down by Internet congestion made by itself. To solve this problem, TCP-Friendly was proposed. This article is aimed at analyzing TCP-Friendly Protocol and studying it with some examples.

**Key words** Computer Network; Congestion Control; TCP-Friendly

摘要 Internet经历了多年的发展，建立了许多协议，位于传输层的TCP以及UDP是其中的代表，也是当今Internet传输中的主力军。但是，日益强烈的Internet使用需求，与难以迅速迭代的硬件设备等因素产生了矛盾，带来了网络拥塞的问题。TCP拥有拥塞控制机制，保证了在TCP下的传输过程中不会出现网络拥塞而导致传输崩溃的问题；而另一方面，UDP没有拥塞控制机制。因此，在同一个Internet上执行的两种协议，却不能保证传输的公平性和效率性——UDP将大量侵占TCP的带宽资源，而自身也在网络拥塞中损失效率，这是我们不愿看到的。为解决这个问题，TCP-Friendly即TCP友好的拥塞控制协议被提出。本文意在对此协议进行简要分析，并结合应用实例对此协议进行学习。

关键词 计算机网络；拥塞控制；TCP友好

在Internet中，有时我们需要一个较高速率的、且比较稳定速率的数据传输，例如客户的在线视频观看，在线视频通话等。这类需求的主要特点是：（1）对延迟很敏感，延迟过大会导致网络连接的中断；（2）容许一定数据的丢失，即缺失一定的数据也能子啊应用层上进行忽略而维持正常工作；（3）数据传输的本质基本由速率来决定，即一旦速度低于某个阈值，上层的应用则无法正常工作。

这就需要我们提供比较稳定速率的数据传输以满足需求。当前Internet中的数据传输业务采用的传输协议以TCP为主，而TCP由于其传输协议的复杂性，相比一些无连接的传输协议，比如UDP，在传输速率（即效率）上没有任何优势，而我们的业务需求无法接受低效率的TCP，必须采用UDP等无连接协议来满足需求。UDP由于其轻便性，满足了无连接，进而传输速率是足够快的特点，但也因此，UDP理所当然的不会考虑网络拥塞的问题，这就使得UDP会侵占TCP协议的大量带宽资源，造成协议之间的不公平现象，而且自身也由于不断占用带宽资源，进而使自己陷入拥塞，最终使得Internet整体的效率崩溃。

TCP友好拥塞控制的相关协议，随着计算机发展也被提出了很多，例如RAP协议[4]、LDA协议[5]、TEAR协议[6]等。文中分析的，由Sally Floyd等人提出的TCP友好速率控制（TCP-Friendly Rate Control, TFRC）协议[1]是典型TCP友好拥塞控制协议之一，且已被IETF工作组收录在RFC3448[1]。TFRC协议基于数学模型，由发送端根据当前网络环境调整数据的发送速率，来达到拥塞控制的目的。

TFRC已被证明与TCP在同一网络情况下拥有近似相同的吞吐量，因此是TCP友好的；而这也侧面说明了TFRC的吞吐量稳定，也满足上文所提出的业务需求。

## 1. TFRC协议分析[2, 3]

### 1.1 TFRC协议的拥塞控制机制

TFRC协议[1]的一些基本属性如下：

1. 针对于单播通信
2. 以速率为调控机制，即通过改变单位时间内发送的数据包数量来调整速率
3. 端到端的连接方式
4. 适用于固定数据包大小的应用程序
5. 在接收端计算拥塞控制信息，例如丢包率等

基于这些属性，在这里给出TFRC协议的拥塞控制机制：

1. 数据接收端统计出丢包事件率*p*，然后附带时间戳一并发送回发送端
2. 发送端利用接收端提供的时间戳测量回环时间RTT
3. 将丢包事件率*p*和RTT代入TFRC的吞吐量方程，计算得到应该调整至的传输速率
4. 发送端根据计算所得的传输速率调整发送速率

**1.2 TFRC吞吐量方程[1]**

为了使TFRC协议满足公平性原则，可以与TCP共享带宽，故在此提出简介中提到的数学模型，即TFRC采用的吞吐量计算公式。这个公式非常直接地采取了TCP的吞吐量计算公式，如下：

其中：

*X*：数据传输速率，单位：Bytes/s

*s*：数据包大小，单位：Bytes/s

*R*：回环时间RTT，单位：s

*b*：接收端一次确认的包数量，一般取*b* = 1

*p*：接收端统计的丢包事件率，即一个发送窗口内发送的包数与总包数的比值，故0≤*p*≤1.0

：TCP超时重传时间，单位：*s*，一般取

## 2. TFRC协议的数据发送、接受协议

### 2.1 数据发送协议[1]

发送端发出的包内容为：

1. 包序号
2. 估算的回环时间RTT
3. 数据

首先发送端进行初始化，在此不多赘述。

然后是数据发送。数据发送分为两个阶段：

1. 慢启动阶段

进入慢启动阶段时，数据的发送速率较慢，网络没有拥塞，此时丢包事件率为*p* = 0。但要保证发送端最少以1包/s的速率发送。根据慢启动的方法，发送端的速率不断翻倍，直到发生网络拥塞，即出现了丢包事件。此时进入拥塞控制阶段。

1. 拥塞控制阶段

发送端停止发送速率翻倍，转而使用上一节提到的概念和数学模型进行拥塞控制，稳定传输速率。拥塞控制阶段要保证最少1包/64 s的发送速率。

当发送端收到来自接收端包含丢包数据与时间戳的反馈数据包后，发送端根据时间戳信息计算在拥塞状态下的回环时间RTT，并由TFRC吞吐量计算公式计算出调整的传输速率。

当发送端在2倍RTT时间内没有收到来自接收端的反馈数据包时，发送端直接将发送速率减半。

### 2.2 数据接收协议[1]

接收端发送的反馈包内容为：

1. 最后一个收到的包的时间戳，以及接收数据包要用的时间。发送端用此来计算RTT
2. 从上一次反馈发出起接收端估算的接收速率*Y*
3. 丢包事件率*p*

当数据接收端接收到第一个数据包后进行初始化。

之后，在低速率传输情况下，即发送端的发送速率小于1包/RTT时，接收端每接到一个数据包就发送一个反馈数据包；一般的，接收端每RTT发送一次反馈数据包；而当发生丢包事件，就立刻发送反馈数据包而不等待一个RTT结束。

在这里给出接收端接收到数据包后的流程：

1. 接收端将数据包的序号添加到已接收的数据包的历史记录中
2. 调用丢包率计算模块，计算当前丢包率*p\_now*
3. 将*p\_now*与上一次计算的丢包率*p\_last*进行比较。若*p\_now* < *p\_last*，则结束；若*p\_now* > *p\_last*，则跳转到（4）
4. *p\_now* > *p\_last*，且若从上次发送反馈数据包后收到了发送端发出的数据包，则：
   1. 计算平均丢包事件率*p*，算法将在下节给出
   2. 计算数据包的接受速率：

其中*R\_m*是接收端的超时定时器

* 1. 发送反馈数据包
  2. 重置超时定时器*R\_m*

若从上次发送反馈数据包后没有收到发送端发出的数据包，则直接重置超时定时器*R\_m*

### 2.3 发送端具体行为[1]

若接收到接收端提供的反馈包时：

1. 计算新的回环时间RTT样本*R\_sample*

其中：

*t\_now*：当前时间

*t\_recv*：接收端提供的反馈包的时间戳

*t\_delay*：接收端接收数据所需时间

1. 更新回环时间*R*
2. 更新超时时间RTO
3. 更新发送速率

其中：

*X*：正式的发送速率

*X\_calc*：由TFRC计算出的速率值

*Y*：接收端发送的反馈包中包含的接收端估算的接受速率

*s*：每个包的大小

*t\_mbi*：为64秒

1. 重新设定无反馈时的超时定时器

若发送端没有收到接收端提供的反馈包时：

1. 发送端速率直接减半
2. 按照有接收端的反馈包的计算速率方式的部分内容再次计算发送速率
3. 重新设定无反馈超时定时器

### 2.4 接收端具体行为[1]

上一节中对接收端收到数据包的流程进行了概述，主体的部分为对丢包率*p\_now*的计算，在此给出相应概念与算法：

1. 丢包事件率的计算原则
   1. 当网络中的丢包事件率稳定时，我们所计算出的丢包事件率相比网络状况也应该是稳定的
   2. 当发生新的连续的几个丢包事件时，我们计算出的丢包事件率应该有大幅度增加
   3. 计算出的丢包事件率只应该随着新的丢包事件的发生而增长
2. 丢包事件率的算法
   1. 检测包的丢失

与TCP判别丢包依据的方法相同：在接收到的包中，至少接收到了三个序号大于丢失的包的分组。

* 1. 转化丢失历史记录为丢包事件

每发现一个包的丢失，接收端按如下算法估算丢失的包本应该到达的时刻：

其中：

*Tloss*：丢失包的估算时间

*Tbefore*：丢失包之前的最后一个收到的包的时间

*Tafter*：丢失包之后的第一个收到的包的时间

*Sloss*：丢失包的序号

*Sbefore*：丢失包之前的最后一个收到的包的序号

*Safter*：丢失包之后的第一个收到的包的时间

设Sold为以前的丢包，作为丢失事件的开始，则判断下式：

其中：

*Told*：*Sold*的到达时刻

R：发送端算出的回环时间RTT

若满足此式，则说明此丢包事件属于当前丢失事件；若不满足，则为新的丢失事件。

* 1. 计算平均丢包间隔、丢包事件率

我们需要统计经过上一步中判断出的*n*次丢失间隔（即需要统计9次丢失事件），并运用加权运算来得到平均丢包间隔，进而得出丢失事件率。通常*n* = 8。

* + 1. 对n个丢失间隔的权值w­­i的计算（）

If(wi <= n/2)

wi = 1;

Else

wi = 1 – (i – (n/2 -1))/(n/2 + 1);

* + 1. 计算丢失事件率

init:

I\_tot0 = 0;

I\_tot1 = 0;

I\_tot = 0;

W\_tot = 0;

I\_mean = 0;

for(i = 0; i < n; i++)

{

I\_tot0 = I\_tot0 + (I\_i \* wi);

W\_tot = W\_tot + wi;

}

for(i = 1; i <= n; i++)

{

I\_tot1 = I\_tot1 + (I\_i \* wi-1);

}

I\_tot = max(I\_tot0, I\_tot1);

I\_mean = I\_tot/W\_tot;

以上算法得出的I\_mean即为平均丢包间隔。则丢包事件率为：

## 3. 历史上的TCP友好拥塞控制协议与对比[7]

表1 历史上的TCP友好拥塞控制协议

Table1 TCP-Friendly Congestion Control Protocol in History

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 协议 | 单播/多播 | 控制机制 | 网络支持 | 复杂度 | 友好性 |
| RAP | 单播 | 速率 | 端到端 | 低 | 有限 |
| LDA(+) | 单播 | 速率 | 端到端 | 高 | 可接受 |
| TFRCP | 单播 | 速率 | 端到端 | 中等 | 可接受 |
| TFRC | 单播 | 速率 | 端到端 | 中等 | 好 |
| LTRC | 多播 | 速率 | 端到端 | 中等 | 有限 |
| TRAM | 多播 | 速率 | 可选 | 低 | 有限 |
| TEAR | 多播 | 速率 | 端到端 | 低 | 好 |
| RLA&LPR | 多播 | 窗口 | 端到端 | 低 | 好 |
| MTCP | 多播 | 窗口 | 额外 | 低 | 好 |
| NCA | 多播 | 窗口 | 额外 | 中等 | 好 |
| PGMCC | 多播 | 窗口 | 额外 | 中等 | 好 |

解释：

1. 网络支持：端到端表示不需要额外的网络支持就可以在端节点上实现。
2. 友好性：即对TCP友好性。有限为在高丢失率网络情况下对TCP有明显不公平表现；可接受为在一些特殊情况下会出现问题。

从表中可以看出，我们所分析的TFRC协议是TCP友好拥塞控制协议中表现最突出的，且复杂度也可以接受，这也是TFRC能够广泛使用的原因。

## 4. 当前TCP友好拥塞控制机制的缺陷

通过查阅资料发现，对于TCP友好拥塞控制机制的研究大多集中在两点：稳定性以及TCP友好性。然而，我们的需求大部分在于多媒体流业务，多媒体流本身就具有很多的约束性，这也是这些协议所需要考虑的问题。

例如，多媒体流的传输速率通常要工作在一定范围内，就像在线观看视频，没有一定的速率客户端无法正常工作，即要满足Smin <= Snow <= Smax。低于Smin，业务不可用，而这个最低速率阈值却常常被TCP友好拥塞协议的设计者们忽略，而更侧重于TCP友好性，如上文分析的TFRC也是。但也有设计者考虑到了此类问题，比如《实时多播流的弹性公平性和基于门限的拥塞控制策略》[8]和《支持最少速率保证的UDP 拥塞控制机制》[9]。文献[8]中所使用的方法比较暴力：当Snow <= Smin时，直接将Snow = 0，即直接停止传输；而文献[9]中所使用的方法为：对路由器RED机制进行加强，对标记和不标记的包进行不同的处理，从而支持Smin <= Snow，然而这种方法需要对路由器进行改变，成本较高，大范围推广难度很大。

故，我们在研究TCP友好拥塞控制协议时，也要考虑到业务的可行性需求。

## 5. 结束语

本文章较为详细地分析了TCP友好拥塞控制中最为出色的TFRC协议。下边简要总结一下TFRC的一些特性：

1. TFRC是单播、基于速率的端到端TCP友好拥塞协议，这也使得TFRC能够既在传输方面，也可以在应用程序方面得到应用。
2. TFRC的速率是通过数学建模的方式，且此模型与TCP的模型几乎一致，保证了友好性。
3. TFRC在与TCP进行竞争的时候，在平滑网络吞吐量抖动方面比TCP要有明显的优势[2, 3]。
4. TFRC引入了“丢包事件”的概念。这个概念的引入，使得TFRC能够规避偶然丢包对吞吐量的抖动，特别是进行8个丢包事件间隔的加权平均采样，使得TFRC的吞吐量能够进行平滑的变化。
5. 但是也由于第（2）点，在可用带宽发生变化时，TFRC相比于TCP要反应迟钝，但是最终也能因为二者各自且相似的拥塞控制机制而能够较为贡品地分享带宽。

以上的特点造就了TFRC的出色。但是在第4节的缺陷中我们也提到了目前的TCP友好拥塞控制协议面临的一些问题。虽然在当今时代带宽非常小的网络很难见到，但随着在线多媒体产业的持续走高，（以在线视频为例）视频的质量也是随着硬件设备的提升而提高，从十年前几乎还在应用雏形的1080P，到现在的4K分辨率大行其道，无不对网络的硬件以及相关软件提出了更高的要求，即视频所需的最低传输速率都大于经过与TCP竞争后的传输速率，但又没到带宽的最大值。在能够解决这个问题的协议出现之前，我们只能在更高层的应用层进行网络带宽的控制与争夺。

随着计算机科学技术的不断发展，这个问题也必将得到解决，但也会有更多的问题困扰我们。唯有与时俱进，保持学习，才能继续计算机科学世界的开拓，才能让计算机科学技术继续造福人类。

**参 考 文 献**

[1] Handley H, Floyd S, Padhye J, et al. RFC 3448 TCP Frien-dly Rate Control(TFRC) Protocol Specification[S] .[s.l.]:[s.n.] , 2003.

[2] ZENG Jing-ping, YANG Wen-jun, PENG Li, QIN Xiao-zhen, WANG Bing-wen. Analysis and Application of TCP-Friendly Rate Control Protocol[A]. Computer Technology and Development, 2007, Vol.17 No.1.

[3] ZHOU Yi-ran. TCP-Friendly Rate Control[A], 电信工程技术与标准化, 2004.09.

[4] Rejaie R, Handley M, Estrin D. RAP: An End-to-End Rate-based Congestion Control Mechanism for Realtime Streams in the Internet. In: Proc. of IEEE INFOCOM, New York, NY, 1999, 1337-1345

[5] Sisalem D, Schulzrinne H. The loss-delay based adjustment algorithm: a TCP-friendly adaptation scheme. In: Proc. of NOSSDAV, Cambridge, UK, 1998, 215-226

[6] Rhee I, Ozdemir V, Yi Y. TEAR: TCP emulation at receivers - flow control for multimedia streaming. Technical report, NCSU, 2000

[7] Widmer J, Denda R, Mauve M. A Survey on TCP-Friendly Congestion Control. IEEE Network Magazine, 2001, 15(3): 28-37

[8] 马钰璐，王重钢， 程时端等. 实时多播流的弹性公平性和基于门限的拥塞控制策略. 计算机学报， 2002，25(11):1231-1235

[9] 李方敏,叶澄清,李仁发. 支持最少速率保证的UDP 拥塞控制机制. 计算机研究与发展, 2001, 38(8):988-993