·网络与通信·

苏凡军[1]，邬春学[1]，孙国强[1]，吕 勇[2]

摘 要：在高速网络中，HRED 算法使高速 TCP 流、普通 TCP 流和 UDP 流实现公平共享带宽。HRED 利用 RED 队列的包丢失历史来识别高带宽流，通过对高带宽流进行惩罚，使低带宽流获取更多的带宽。HRED 具有良好的扩展性，不需要保持每一流的状态信息。基于 NS2 的模拟实验证实，HRED 可以在高速网络中实现更好的 RTT 公平性，有效地解决了适应流和非适应流共享带宽的问题。

[1], WU Chun-xue[1], SUN Guo-qiang[1], LV Yong[2]

高速网络中的拥塞控制算法是最近几年的研究热点。研究者提出了许多基于源端的拥塞控制算法，如HSTCP[1]和CB-HSTCP[2]等。大量研究表明，仅仅依靠源端的拥塞控制算法，难以实现较好的网络性能，原因为：在网络中，除了TCP流外，还有许多基于UDP的多媒体数据流，甚至一些恶意流。因此，需要基于路由器端的拥塞控制算法——主动队列管理(Active Queue Management, AQM)算法。目前已经有许多用于普通网络的主动队列算法，其中RED[3]是IETF所推荐的一种主动队列管理算法，已经在许多网络路由器中实现。高速网络中主动队列算法的研究则刚刚展开，如Easy RED[4]。本文根据高速网络特点，在RED算法的基础上，提出了一个新的主动队列管理算法——HRED(High-speed RED)。HRED算法的思想为：根据一定标准，检测出占据大部分带宽的少量高带宽流，随后在路由器端惩罚高带宽流，使高带宽流降低发送速率，从而使原来的低带宽流获取更多带宽。通过这种方式来改善共享带宽的不同流之间的公平性，并采用仿真软件NS2 对该算法进行了分析和验证。

1 HRED 算法

HRED 的流程如图 1 所示。

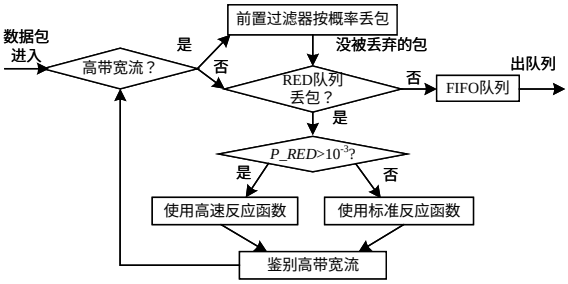

图 1 算法流程

HRED 算法是对 RED 算法的改进。RED 算法的主要缺陷有：无法实现完全的 RTT 公平性，即具有不同 RTT 的数据流不能公平共享带宽；难以解决适应流(如 TCP 流)和非适应流(如基于 UDP 的多媒体流)公平共享带宽的问题。为此本文提出了 HRED 算法。

1.1 高带宽流的检测和包前置丢弃概率的调整

Floyd 等人指出的 RED 队列丢包历史反映了不同流在网络中的带宽占用情况。可以收集 RED 队列的包丢失信息作为判断高带宽流的依据。

TCP和HSTCP的反应方程[1]为

$$T=\frac{c}{RTT\times p^{m}} \tag{1}$$

其中，T 是流的输出；p 是包丢失率。定义拥塞周期 $Cycle$ 为 2 次拥塞之间的时间间隔，拥塞周期为

$$Cycle=\frac{1}{T\times P}=\frac{RTT}{C\times P^{1-m}}$$

从以上公式可见，$Cycle$与RTT和p都有关系。如果知道了 p 值，并设定一个RTT值 (称这个RTT为标准RTT，简写为 S_RTT)，则可以计算出$Cycle$(本文称其为标准拥塞周期 S_Cycle)的值作为判断高带宽流的标准，拥塞周期短于 S_Cycle的流为高带宽流。对于标准TCP, $c=1.5^{0.5}$, $m=0.5$；对于HSTCP，则$c=0.12$, $m=0.835$[1]。采用标准TCP的反应函数还是高速TCP的反应函数取决于 P_RED。 P_RED 小于阈值，则认为是高带宽网络，采用HSTCP的反应方程；否则采

基金项目：上海高校选拔培养优秀青年教师科研专项基金资助项目

作者简介：苏凡军(1976－)，男，讲师、博士，主研方向：计算机网络 QoS，网络协议；邬春学，教授、博士；孙国强，副教授、博士；吕 勇，讲师、博士

收稿日期：2007-09-02 **E-mail：**sufanjun@163.com

用标准TCP的反应方程。参考HSTCP[1]中的参数*Low_P*，阈值设置为 10^{-3}。因此，当*P_RED*≥10^{-3}时，即

$$S\_Cyle = \frac{S\_RTT}{\sqrt{1.5 \times P\_RED}} \tag{2}$$

当*P_RED*<10^{-3}时，则

$$S\_Cyle = \frac{S\_RTT}{0.12 \times P\_RED^{0.165}} \tag{3}$$

为实现这个算法，需要检测 RED 队列的包丢失信息。设定检测时间 *time_D*=*S_Cycle*，如果某个流在 *time_D* 时间内有多于一次的拥塞事件，则相当于该流的拥塞周期小于 *S_Cycle*，因此，认为该流是高带宽流。数据包是突发的，为了防止将一次拥塞事件中的多个包丢失认为是多次拥塞，则需要设置另外一个参数 *time_C*，在这个时间间隔内的丢包认为是一个拥塞事件。由于 TCP 和 HSTCP 需要一个 *RTT* 时间才可以检测出包丢失并采取拥塞控制策略，因此本文设置 *time_C* = *S_RTT*。

以上的算法需要保存少量高带流的信息。采用一个双向链表数据结构来保存高带宽流的信息。每个高带宽流需要保存的信息用一个五元组来表示(流ID, *count, NC, ND,* P_r)，本文称为一个节点信息。其中，*count*表示在*time_C*时间间隔内的包丢失数目；*NC*表示在检测周期*time_D*时间内的拥塞事件个数；如果*time_D*时间内*NC*>1，则表示有多个拥塞事件发生，因此，认为是高带宽流，从而需要调节包丢弃前置概率P_r；*ND*表示被鉴别为高带宽流的次数。

所谓的包前置丢弃概率P_r是指数据包到达路由器时，包被前置过滤器丢弃的概率。每个*time_C*时间后，如果*NC* >1，并且*count*>1，则包丢弃概率调整公式为

$$p_\gamma = p_r + P\_RED \tag{4}$$

此外算法还需要考虑非适应流。非适应流在拥塞发生后并不降低发送速率。HRED 算法引入一个新的变量 *ND*，用于记录被标记为高带宽流的次数。如果一个流被判断为高带宽流，则 *ND*++。显然如果一些占用过多带宽的非适应流在包丢失后不降低发送速率，则会被多次标记为高带宽流。需要加大对它的惩罚，包丢弃概率调整公式为

$$p_\gamma = p_r \times ND \tag{5}$$

采用这种方式，可以迅速地增加对高带宽非适应流的惩罚，降低其发送速率。

如果*time_D*时间内NC_i<2，说明该流的发送速率已经降低，因此调节ND_i=ND_i/2，并在ND_i<1 时删除该流的记录。

算法的伪代码如下：

```
Variables:
  i: 流的 ID
  p_i: 流i的前置包丢失率
  P_RED：RED 队列的平均丢包概率，用于计算检测周期
  NC_i: 拥塞事件个数
  ND_i: 检测为高带宽流次数
  count_t: 检测周期 time_D 间隔内总丢包个数
  count_i: 流i在time_C时间内丢失包数
  time_C: 拥塞事件统计间隔
  time_D: 检测周期
  S_RTT: 标准 RTT 值
  Packetsize：包大小
  Bandwidth: 瓶颈带宽
On packet m in RED dropped or marked:
  i=conn(m); //计算流的 ID 值
  search(i); //查找流 i 的记录
  if( i is new )
    creat(i); //在链表中为流 i 创建一个新的记录
  count_t++; //统计 time_D 时间内丢失包的总数
  count_i++; //统计流i在time_C时间内丢包数
Timer event (on time_C  expiry):
  if(count_i>1)
    NC_i++;
    count_i = 0;
  if(NC_i >=2 && count_i>1)
    p_i= p_i+P_RED;
Timer event (on time_D  expiry):
  if(NC_i >=2)
    ND_i++;
    p_i= ND_i ×P_RED;
  else
    ND_i/=2;
  if(ND_i<1)
    delete(i); //删除流 i 的记录
  P_RED = count_t /(bandwidth×time_D/ packetsize/8);
  if(P_RED <10^-3)
    time_D =S_RTT/0.12/ P_RED^0.165;
  else
    time_D =S_RTT/(1.5×P_RED)^0.5;
  count_t＝0；
  NC_i=0;
  count_i = 0;
  count_t= 0;
```

1.2 前置过滤器的丢包处理

前面的算法已经确定了高带宽流。因此，前置过滤器将检查每个到达的数据包的包头，如果包的流 ID 与保存的链表中某个流的 ID 相同，则认为是高带宽流，从而根据该流的丢弃概率丢弃包。未被丢弃的包则进入 RED 队列。如果不是高带宽流，则直接进入 RED 队列。

算法的伪代码如下：

```
Variables:
  i: 流的 ID
  p_i: 流i的前置包丢弃概率
On packet k arrive:
  i=conn(k); //计算流的 ID 值
  search(i); //在高带宽流链表中查找流 i 的记录
  if( i is high bandwidth flow )
    //产生一个介于(0,1)之间的随机数
    double u = Random::uniform(0,1);
    if (u <= P_i)
      drop (packet);  //丢弃高带宽流的包
    else
      enter_RED( ); //未被丢弃的高带宽流包进入 RED
  else
    enter_RED( ); //非高带宽流包直接进入 RED
```

2 算法的实现

HRED算法实现简单，仅需在RED算法基础上添加代码。添加代码主要包括高带宽流的检测和前置过滤器的丢包处理。采用了双向链表来保存高带宽流的信息。为加快检索速度，可以使用哈希表，记录某个流的 ID 和其记录的存储位置

$f(ID)$，从而快速检索某个流 ID 所对应的表项。计算复杂度为 $O(1)$。

3 模拟分析

采用NS2 进行模拟实验分析了HRED的性能。模拟拓扑见图 2。包大小设为 1 000 B，TCP使用SACK版本，TCP的拥塞窗口最大值设置为 100 000，从而允许大拥塞窗口发送数据包。另外设定接收端有足够的缓冲空间。缓冲空间大小设置为带宽延迟乘积。设置队列的$\min_{th}$为缓冲空间大小的 1/5，$\max_{th}$为缓冲空间大小的 4/5。HRED算法的S_RTT为 150 ms。其余设置采用NS2 中的默认设置。模拟运行 400 s，统计结果取平均数据。

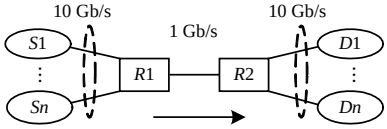

图 2 模拟拓扑

3.1 对 RTT 不公平性的改善

为了尽可能模拟现实环境，同时消除 phase effect 问题，使用了 Web 业务流以及一些正向和反向短期 TCP 流，另外还运行 3~5 个标准 TCP 流。连接起始和结束时间随机设定。这些称为模拟的背景业务，每次模拟都运行。瓶颈链路延迟为 10 ms，边缘链路延迟为 10 ms ~100 ms。运行 5 个 HSTCP 流，流的 RTT 分别为 60 ms, 80 ms, 140 ms, 200 ms, 400 ms。

实验结果列于表 1。RED 并不能完全消除 RTT 不公平性，这主要是由于 RED 对所有流采用了相同的包丢弃概率。HRED 则可以有效地消除 *RTT* 不公平性，主要是通过式(3)的机制来实现对短 RTT 流的惩罚。RED、HRED 的背景流为 12.68%、17.30%。RED、HRED 总利用率为 93.11%、94.39%。

表 1 *RTT* 公平性实验结果

RTT	网络带宽份额/(%)	
	RED	HRED
50	33.24	22.10
100	25.33	18.60
150	13.16	16.47
200	6.43	10.62
400	2.27	9.30

3.2 对非适应流的处理

瓶颈链路和边缘链路的延迟都设置为 20 ms。在高速网络中运行 5 个 HSTCP 流(流 1~流 5)，同时运行 3 个 CBR 流(流 6~流 8)，速率分别是 150 Mb/s, 200 Mb/s, 250 Mb/s。模拟运行 400 s。从实验结果看出，RED 算法基本无法降低 CBR 流的带宽，而 HRED 算法则可以有效地减少高带宽的 CBR 流的带宽，使适应流和非适应流可以公平共享带宽。HRED 对非适应流的处理如图 3 所示。

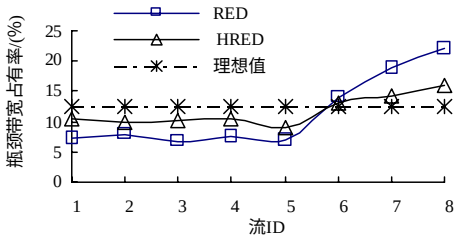

图 3 HRED 对非适应流的处理

4 结束语

其他一些模拟实验也证实 HRED 可以使普通 TCP 流得到更多的带宽。并且 HRED 在普通网络中同样具有好的性能。HRED 算法通过鉴别并惩罚少量的高带宽流，从而使 HSTCP 流、普通 TCP 流和 UDP 流能够公平共享带宽。而且 HRED 不需要每一流都保留状态信息，而仅仅动态地保留少量流的状态信息。算法简单并具有扩展性，在保持 RED 算法优点基础上，实现了更好的性能。

所需保留的高带宽流的信息量(即链表的大小)和 *S_RTT* 的选择有关。显然 *S_RTT* 越大，则标记为高带宽的流越多，从而公平性越好，但需要保存的信息越多。而 *S_RTT* 越小，则需要保存的信息越少，但公平性有所降低。文中 *S_RTT*＝150 ms。关于 *S_RTT* 取值的深入分析和探讨是下一步研究的方向。

参考文献

[1] Floyd S. HighSpeed TCP for Large Congestion Windows[S]. RFC3649, 2003.

[2] 苏凡军, 潘雪增, 蔡 亮. CB-HSTCP: 高速网络中的公平 TCP 算法[J]. 电子学报, 2005, 33(11): 2084-2089.

[3] Floyd S, Jacobson V. Random Early Detection Gateways for Congestion Avoidance[J]. IEEE/ACM Transactions on Networking, 1993, 1(4): 397-413.

[4] Grieco L A, Mascolo S. TCP Westwood and Easy RED to Improve Fairness in High-speed Networks[C]//Proc. of the 7th International Workshop on Protocols for High-speed Networks. Berlin, Germany: [s. n.], 2002.

(上接第 104 页)

4 结束语

本文提出一种以网络层协议为核心的半覆盖自组织组播体系结构 NSMM。与覆盖组播相比，NSMM 的组播逻辑回归了网络层与链路层。与 IP 组播相比，NSMM 用自组织结构替代了仅含网络节点的传统转发结构。NSMM 可以成为 TCP/IP 协议栈的组成部分，作为独立的体系，它能与现有的组播技术同时工作。

参考文献

[1] Chu Yanghua, Rao S G, Zhang Hui. A Case for End System Multicast[C]//Proc. of ACM SIGMETRICS. Santa Clara, USA: ACM Press, 2000.

[2] Fahmy S, Kwon M. Characterizing Overlay Multicast Networks[C]// Proc. of IEEE ICNP'03. Atlanta, GA, USA: [s. n.], 2003.

[3] Mohapatra P, Gui C, Li J. Group Communications in Mobile Ad hoc Networks[J]. IEEE Computer Magazine, 2004, 37(2): 52-59.