PRED: 一种具有优先级自适应的队列管理新算法

张克平, 田 辽, 李增智
(西安交通大学计算机系统结构与网络研究所, 陕西西安 710049)

摘 要: 现有的拥塞控制采用以 TCP 为核心的基于窗口技术的端到端控制, 具有丢包、响应速度慢等缺陷. 本文提出的基于优先级的队列管理算法(PRED), 使路由器更加精确地管理队列, 算法的主要参数能够适应网络负载的动态变化, 有效地克服了现有拥塞控制的缺陷. 实验结果表明, 在相同的配置下, 采用 PRED 的网络在降低丢包率、减少队列抖动等性能上均优于端到端拥塞控制.

关键词: 队列管理; 优先级; RED; 自适应
中图分类号: TP393107 文献标识码: A 文章编号: 037222112 (2004) 062103920 5

PRED: A New Queue Management Algorithm with Priority and Self2Adaptation

ZHANG Ke2ping, TIAN Liao, LI Zeng2zhi
(Institute of Computer Architecture & Network, Xi. an Jiaotong University, Xi. an, Shannxi 710049, China)

Abstract: With various multimedia applications rapidly advanced in the current INTERNET, the frequent occurrence of congestion has led many researchers to re2examine the issue of congestion control. At present, the congestion control technologies we master and use adopt end2to2end control which regards TCP as core and is based on window. However, this art has its own drawbacks such as high packet loss, low response speed etc. In this study, the authors give a queue managing algorithm based on priority and self2adaptation. The algorithm is called Priority Random Early Detection (PRED). The algorithm can make router more accurately schedule queue. The primary parameter of algorithm can adapt dynamic change of network load in true surroundings. Validly, it avoids the drawbacks of congestion control we master. Experiments prove, under the same configurations, that PRED algorithm is better than end2to2end congestion control arts in terms of reducing the rate of packet loss and oscillation of queue.

Key words: queue manage; priority; RED; self2adaptation

1 引言

INTERNET 在过去几年已经经历了爆炸式的增长. 它的发展速度如此之快已经超出了人们的想象, 并且已经成为人们信息交换的重要手段之一. 随着主机、用户、应用(数据、语音、视频等)的不断增加, 使 INTERNET 经常发生拥塞现象, 造成数据传输延迟、抖动等性能指标的下降和带宽、缓存等网络资源利用率的降低. 在极端的情况下, 可导致整个网络崩溃. 因此, 在过去的十几年中, 如何预防和控制拥塞一直是计算机网络界研究的热点[1~13].

目前, INTERNET 上研究拥塞控制的技术路线主要有两条. 一条是以 TCP 为核心的基于窗口的端到端的闭环控制方式. 这种方式主要是以丢包作为拥塞发生的信号, 当源端确认已发出的数据包丢失后, 通过减少拥塞窗口的方法来降低发送速率, 达到控制的目的. 1988 年 Van Jacobson[1] 指出了 TCP 在控制网络拥塞方面的不足, 并提出了/ 慢启动0 算法、/ 拥塞避免0 算法. 1990 年出现的 TCP Reno 增加了/ 快速重传0 算法、/ 快速恢复0 算法, 避免了网络拥塞不够严重时采用/ 慢启动0 算法而造成无穷大地减少发送窗口尺寸的现象, 这样 TCP 的拥塞控制就由这 4 个核心部分组成.

最近几年又出现 TCP 的改进版本如 NewReno、SACK 等. TCP 拥塞控制是通过控制一些重要参数的改变来实现的.

另一条技术路线是在路由器中采用排队算法和数据包丢弃策略, 也就是 IP 拥塞控制问题. 排队算法通过决定哪些包可以传输来分配带宽, 而丢弃策略通过决定哪些包被丢弃来分配缓存. IP 拥塞控制算法主要包括: 传统的先进先出(FIFO)、随机早期检测(RED)、加权的随机早期检测(WRED)、分布式加权的随机早期检测(DWRED)、公平排队(FQ)、加权公平排队(WFQ)、Flow RED、核心无状态公平排队(CSFQ)等.

本文在研究了 RED 及其改进算法优点和不足的基础上, 提出了队列实时优先级的概念, 利用队列实时优先级的思路, 提出了一个简单实用的具有优先级自适应的队列管理新算法(Priority RED, PRED). 该算法解决 RED 的实时优先级问题; 解决了 RED 计算分组的到达对平均队列长度的影响, 而未考虑分组离去对平均队列的影响问题, 使路由器更加精确地管理队列; 同时, 算法的主要参数能够适应网络负载的动态变化, 减少了在拥塞避免时的不必要丢包, 从而提高了系统的吞吐量.

收稿日期: 2002212223; 修回日期: 2003206218
基金项目: 国家 863(No. 863251129462008, No. 2001ba201a33); 宁夏自然科学基金(No. F001)

2 RED 及其改进算法的分析

INTERNET 上目前使用的队列管理算法大部分仍是 Drop Tail 算法. 该算法存在如下缺点: ¹ 当网络发生拥塞时, 只进行简单的拥塞控制, 而不进行拥塞避免处理; º 文献[16]指出: 对于 Drop Tail 路由器, 每个拥塞周期都会引发网络中的/ 全球同步0 现象; » 没有分组优先级的概念; ¼ 不区分 UDP 流和 TCP 流, 使 TCP 流在资源竞争中处于不利的地位, 无公平性保证.

为了克服 Drop Tail 路由器的上述缺陷, 文献[11]提出了 RED 算法. 当平均的队列长度 Q_{avg} 小于缓冲门限最小值 Min_{th} 时, 不进行分组的丢弃; 当平均的队列长度在缓冲门限最小值 Min_{th} 和最大值 Max_{th} 之间时, 其分组丢弃概率 p 随 Q_{avg} 的增长而线性增长, 直到最大分组丢弃概率 Max_p; 当平均的队列长度超过最大门限 Max_{th} 时, 所有来到的包都被丢失. 这个方法比以前算法更加有效的控制队列长度. RED 很容易实现, 因为它丢弃的仅仅是来到的包, 并且允许 FIFO 排队.

RED 算法实现了下列目的: ¹ 拥塞避免. RED 路由器在网络发生拥塞之前即进行了早期分组丢弃, 使发送方进入拥塞避免阶段, 可避免网络拥塞的发生; º 拥塞控制. RED 算法能保证即使缺乏传输层的协作, 也能控制路由器的最大缓冲队列长度, 进而控制网络拥塞; » 避免/ 全局同步0. 由于分组的随机丢弃方式, 多个连接同时增加或减少它们发送窗口的几率很小, 因而可以避免全局同步现象; ¼ UDP 流间的公平竞争: 一个 UDP 流被丢弃的分组数与该流所占带宽成正比.

但是, 根据已有的研究表明, RED 算法存在如下几方面的问题: ¹ 公平性问题. 主要表现在两个方面: 一是各个 TCP 连接之间的公平性, RED 算法中某时刻的分组丢弃概率 p 对各连接是相同的, RTT 较小的连接在资源竞争中处于优势, 导致各 TCP 连接间的不公平. 另外, 虽然 RED 比 DROP TAIL 好, 但所有连接具有同样丢失概率就意味着, 即使一个连接使用少于它的公平分配, 也将经历包丢弃, 这就使得窄带 TCP 无法得到公平分配. 二是 TCP 流与 UDP 流间的公平性, TCP 流对分组丢弃是有响应的, 而 UDP 流并不响应分组的丢弃, 结果使 UDP 流占据更多的带宽. º 优先级问题. RED 算法没有分组优先级的概念, 不能实现 INTERNET 商业化的现实. » 稳定性问题. ¼ RED 算法的性能对参数设置和网络的状况很敏感, 改变参数对性能影响很大. 在特定的网络负载状况下依然会导致多个 TCP 的同步, 造成队列震荡、吞吐量降低和时延抖动加剧. 到目前为止, 这些参数还没有明确的设定方法. ½ 随着网络中 TCP 连接数目的增加, 网关的平均队列长度会逐渐增加. ¾ 仅计算分组的到达对平均队列长度的影响, 而未考虑分组离去对平均队列的影响, 这样计算的平均队列长度显然是偏高, 导致不必要的分组丢失, 造成网络的利用率偏低.

3 PRED 算法

311 队列实时优先级的概念

对于单队列来说, 不存在优先级问题. 本文作者提出采用多队列的方法, 进而引出分组入队与离队时的次序问题. 为了解决此问题作者提出了实时优先级的概念, 即当分组到达时总是进入实时优先级最高的队列, 进而达到降低丢弃率的目的.

312 PRED 算法的设计过程

假设路由器(如图 1 所示)在其缓冲池中有三个队列 q_1, q_2, q_3(以三队列为例), $P_{1\text{-}in}$、$P_{2\text{-}in}$、$P_{3\text{-}in}$ 分别是入三个队列的优先级, $P_{1\text{-}out}$、$P_{2\text{-}out}$、$P_{3\text{-}out}$ 分别是出三个队列的优先级. 在入队列中, 其初始值设为 $P_{1\text{-}in}$ 的优先级最低, $P_{2\text{-}in}$ 的优先级居中, $P_{3\text{-}in}$ 的优先级最高. 在出队列中, $P_{1\text{-}out}$ 的优先级最低, $P_{2\text{-}out}$ 优先级居中, $P_{3\text{-}out}$ 的优先级最高.

设 q_1, q_2, q_3 的平均队长分别为 Q_{avg1}、Q_{avg2}、Q_{avg3}, K 为常数. 当分组到达时, 入队列的优先级可按照公式(1)计算各队列的实时优先级:

$$P_{i\text{-}in} = k / Q_{avgi}, \quad i = 1, 2, \cdots, n \tag{1}$$

平均队列越长表示该队列在这一时刻包含分组数目越多. 式(1)的含义是优先级与平均队列长度成反比.

重新对 $P_{1\text{-}in}$、$P_{2\text{-}in}$、$P_{3\text{-}in}$ 进行排序, 我们总是把到来的分组放在优先级最高的队列中, 这样就可保证总是把分组放入平均队长最短的队列中. 对于出队列的优先级可按照公式(2)计算:

$$P_{i\text{-}out} = a * Q_{avgi}, \ i = 1, 2, \cdots, n \tag{2}$$

其中, a 为常数. 重新对 $P_{1\text{-}out}$、$P_{2\text{-}out}$、$P_{3\text{-}out}$ 进行排序, 按照从高优先级到低优先级顺序读出数据. 式(2)的含义是分组在出队列时平均队列越长, 优先级越高.

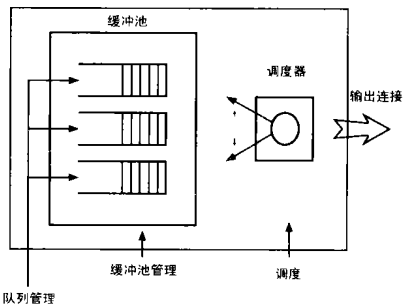

图 1 路由器资源管理

将图 1 的缓冲池看成一个时间离散、先到先服务的单服务队列系统, 缓冲池的容量有限(这比较符合实际情况). 服务速率 C 固定, q_i 表示第 j 类业务源在时隙$[t-1, t)$结束时缓冲池中的队列长度, Y_{t+1} 表示第 j 类业务源在时隙$[t, t+1)$开始时新到达的分组数. 设缓冲池容量为 B, 分组输出速率同样为 C. 则缓冲池中第 j 类业务源的队长的递推关系式为:

$$q_j[0] = 0$$

$$q_j[it, (i+1)t] = \begin{cases} 0, & \text{满足条件 1} \\ q_j[(i-1)t, it] + Y_j[it, (i+1)t] - C, & \text{满足条件 2} \\ B, & \text{满足条件 3} \end{cases} \tag{3}$$

其中:

条件 1: $q_j[(i-1)t, t] + Y_j[it, (i+1)t] - C < 0$

条件 2: $0 < q_j[(i-1)t, t] + Y_j[it, (i+1)t] - C < B$

条件 3: $q_j[(i-1)t, t] + Y_j[it, (i+1)t] - C \geq B$

那么, 所有 n 个业务源占队长的瞬时值 q 为:

$$q = \sum_j q_j \tag{4}$$

我们用分组的瞬时队列长度 q 计算平均队列长度 Q_{avg}:

```
if q= = 0
   Qavg= Qavg(1- Wq)(q_time2now)/s;
   q_time= now;
else
   Qavg= Qavg+ Wq(q+ Qavg);
```

其中, q_time 指最近的 q= 0 时的时刻, s 是典型的传输时间. 当分组离开时, 按同样的方法计算 Q_{avg}.

分组的丢弃概率 p 按下式计算:

$$p=\begin{cases}0, & Q_{avg}< Min_{th};\\ \dfrac{Q_{avg}- Min_{th}}{Max_{th}- Min_{th}}Max_p, & Min_{th}[Q_{avg}< Max_{th};\\ 1, & Q_{avg}\backslash Max_{th}\end{cases}$$

为合理地保持队列长度以及根据连接的数目动态自适应调整 Max_p, 每当平均队列长度更新时自动调整 Max_p[15], Max_p 的算法如下:

```
if(Qavg\ Maxth)&&(status ! = Above) {
   status= Above;
   Maxp= Maxp* b;
   }
if((Minth[ Qavg< Maxth)
   status= Between;
if((Qavg< Minth)&&(status! = Below)) {
   status= Below;
   Maxp= Maxp/a; }
```

根据以上算法, 每当平均队列长度超过 Max_{th}, 则增加 Max_p 而当 Q_{avg} 低于 Min_{th} 时则减少 Max_p. 这样, 网关能尽量保持平均队列长度在一个合理的范围内, 不至于溢出或空闲.

下面我们给出 PRED 算法示意图:

```
(1) Write queue manage
Initialization:
   Pi2in(i= 1,2,3,,,n);
For each pocket arrival:
   Calculate qi average queue size Qavgi;
   Calculate real time priority Pi2in;
   Sort out the highest priority Pk2in
      from Pi2in;
// Pk2in queue is qk
   if Qavgk[ Minth
Write arrived packet to queue Qk;
else if Minth< Qavgk< Maxth
   Calculate Probability p;
   Write arrived packet to queue qk with
      drop Probability p;
else if Qavgk\ Maxth
   Drop the packet;
(2) Read queue manage
Initialization: Pi2out;
For each packet departure:
If queue become empty
   Q_time= now;
Calculate qi average queue size Qavgi;
Calculate real time priority Pi2out;
Sort out the highest priority Pk in
   from Pi2out;
Read arrived packet from queue qk;
```

4　性能分析

我们通过实验网对 PRED 与 RED 相比较进行性能评价. 实验网的结构如图 2 示. 实验采用 NS 网络模拟器[16]进行. 瓶颈链路位于节点 A 和节点 B 之间, 链路容量为 15Mbps (3750packets/s, 分组的缺省大小为 500bytes), 延时 15ms; 所有业务均为持久的 FTP 业务源, 他们与节点 A 之间的链路容量为 10Mbps, 延时 15ms; 除节点 A 的队列分别为 RED 和 PRED 外, 其余队列均为 Drop Tail, 所有节点的缓冲大小均为 300packets. 假设 N= 300, 我们做两组实验. 实验 1, 将 300 个 FTP 业务源均分为 3 组, t= 0s 时启动第一组; t= 10s 启动第二组; t= 20s 启动第三组; t= 30s 停止第一组; t= 40s 停止第二组; min_{th}= 80, max_{th}= 120, 持续 100 秒. 实验 2, 同时加载 300 个持续的 FTP 业务源; min_{th}= 20, max_{th}= 60, 持续 100 秒. 性能分析和实验结果如下.

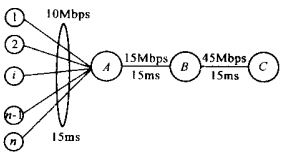

图 2　实验网的拓扑结构

411　对拥塞的反映

PRED 算法改进了平均队列的计算方法: 不仅考虑分组的到达对平均队列长度的影响, 还考虑分组的离去对平均队列长度的影响, 若分组的到达或离去时队列为空, 则平均队列长度随时间按指数衰减(使其迅速趋近于 0). 因此, 该算法能更准确反映网络的拥塞程度.

412　丢包数

两个实验分别做 10 次, 每次的结果分别属于 TEST# 1 到 TEST# 10, 见表 1. 从表 1 可看出: 在两个实验中, PRED 的丢包数分别为: 2386、2430, 而 RED 的丢包数分别为: 10946、8573. 通过比较可以得出: PRED 的丢包数比 RED 大大减少.

表 1　丢包数汇总

TEST#	实验 1		实验 2	
	RED	PRED	RED	PRED
1	11703	2360	9034	2045
2	12033	1798	8572	2179
3	9768	1844	8671	1207
4	10035	2632	8113	2586
5	8934	2597	9456	3004
6	11458	2799	7169	2343
7	12893	2337	8632	2638
8	11096	2931	9019	2749
9	10738	2548	8994	3350
10	10802	2013	8067	2201
Total	109460	23859	85727	24302
Average	10946	2386	8573	2430

4.3 平均队列的长度

从表 2 可看出, 采用 RED 时, 节点队列长度大部分时间都较大, 因而数据包在节点经历一个较长的排队延迟. 实验 1 采用 PRED 算法时, 节点队列基本上保持在一个较短的水平, 与 RED 的平均队长相比, 缩短了约 15.6% (实验二的为 15.55%). 因而数据包在节点会经历的排队延迟也较短, 从而最终缩短了数据包在网络中的传输延迟.

表 2 平均队列的长度

TEST#	实验 1		实验 2	
	RED	PRED	RED	PRED
1	117	103	54	45
2	120	102	50	43
3	103	96	49	49
4	100	100	51	43
5	115	89	56	44
6	114	87	52	43
7	119	95	49	38
8	110	101	53	49
9	107	87	56	50
10	108	95	46	48
Total	1113	955	516	452
Average	111	96	52	45

4.4 队列长度的抖动

抖动是指分组在传输时的延迟的变化程度. 对表 3 的数据进行分析, 可看出: 在对 10 次实验值的均值和方差取平均 (avg) 后, 实验 1、实验 2PRED 的队列长度的(平均) 均值和(平均)方差分别为(0.112, 0.053)、(0.132, 0.065), 而 RED 对应的值分别为(0.153, 0.161)、(0.169, 0.172), 比较发现: PRED 的队列长度抖动比 RED 的要小.

表 3 队列长度的抖动

TEST#	实验 1				实验 2			
	PRED		RED		PRED		RED	
	均值	标准方差	均值	标准方差	均值	标准方差	均值	标准方差
1	0.11	0.05	0.15	0.16	0.13	0.06	0.17	0.19
2	0.10	0.07	0.15	0.15	0.13	0.07	0.17	0.17
3	0.13	0.06	0.15	0.15	0.13	0.07	0.18	0.18
4	0.12	0.05	0.15	0.16	0.13	0.07	0.17	0.16
5	0.11	0.05	0.15	0.16	0.12	0.07	0.17	0.15
6	0.09	0.05	0.15	0.17	0.13	0.06	0.17	0.19
7	0.12	0.05	0.16	0.16	0.13	0.07	0.16	0.17
8	0.14	0.05	0.16	0.16	0.14	0.06	0.16	0.19
9	0.09	0.05	0.15	0.15	0.13	0.06	0.17	0.18
10	0.11	0.06	0.15	0.16	0.13	0.06	0.16	0.14
avg	0.11	0.05	0.15	0.16	0.13	0.07	0.17	0.17

4.5 吞吐量

吞吐量是指在不丢包的情况下, 被测对象(系统、设备、特定连接、特定服务类等) 所能达到的最大传输速度. 从表 4 可看出 PRED 获得比 RED 高的吞吐量, 至少提高 5.7% 以上.

表 4 吞吐量 Mbps

TEST#	实验 1		实验 2	
	RED	PRED	RED	PRED
1	2.95	3.16	2.58	3.09
2	2.99	3.14	2.55	3.08
3	3.07	3.16	2.56	3.09
4	2.90	3.18	2.54	3.04
5	2.93	3.12	2.56	3.03
6	2.87	3.14	2.57	3.01
7	2.96	3.15	2.58	2.97
8	2.92	3.10	2.59	2.94
9	3.03	3.09	2.59	3.01
10	3.06	3.12	2.53	3.12
Average	2.97	3.14	2.57	3.04
$\frac{PRED-RED}{RED}*100\%$	5.72% 用平均值计算		18.3% 用平均值计算	

5 结论

本文阐述了一种基于优先级的队列管理算法 PRED 的设计方案以及该算法与 RED 的比较. 实验结果表明, 采用基于优先级的队列管理算法可降低丢包率、减少抖动、提高吞吐量等.

本算法与其他拥塞控制算法相比, 具有以下特点: 具有实时优先级的概念; 计算分组的到达对平均队列长度的影响, 同时考虑分组离去对平均队列的影响; Max_p 可以随网络负载的变化而动态变化, 算法能够适应网络负载的不同情况; 降低丢包率、减少抖动、提高吞吐量等.

参考文献:

[1] Jacobson V. Congestion avoidance and control IEEE/ACM transaction nerworking[J]. 1998, 6(3): 314-329.

[2] Zhang H. Traffic Control and QoS Management in the INTERNET[R]. http://www.cs.cmu.edu/~hhaang/.

[3] RFC 2018, 1996. TCP Selective Acknowledgment Options[S].

[4] RFC2001, 1997. TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms[S].

[5] Paxson V, Floyd S. Wide area traffic: The failure of Poisson modeling [J]. IEEE/ACM Transactions on Networking, 1995, 3(3): 226-224.

[6] Floyd S, Fall K. Promoting the use of End2to2End congestion control in the INTERNET[J]. IEEE/ACM Transaction on Networking, 1999, 7(4): 458-472.

[7] Mathis M, Semske J, Mahdavi J, Ott T. The macroscopic behavior of the TCP congestion avoidance algorithm[J]. Computer Communication Review, 1997, 27(3).

[8] Jacobson V. Modified TCP Congestion Avoidance Algorithm[R]. Technical Report. http://ftp.ee.lbl.gov/.

[9] Hoe J. Start2up Dynamics of TCP. s congestion control and avoidance schemes[D]. USA: MIT, 1995.

[10] Lawrence S, Brak M, Larry L. TCP Vegas: End to end congestion avoidance on a global INTERNET[J]. IEEE Journal on Selected Areas in Communications, 1995, 13(8): 1465- 1480.

[11] Floyd S, Jacobson V. Random early detection gateways for congestion avoidance[J]. IEEE/ACM Transactions on Networking, 1993, 1(4): 397- 413.

[12] D Lin RMorris. Dynamics of random early detection[C]. Cannes, France: SIGCOMM. 97, 1997.

[13] Ott T J, lakshman T V, Wong L H. SRED: StabilizedRED[A]. In Doshi B ed Proceedings of the IEEE INFOCOM[C]. New York: IEEE Communications Society, 1999. 1346- 1355.

[14] Feng W, Kandlur D, Saha D, et al. Blue: A New Class of Active Queue Management Algorithms[R]. USA: CSE2TR2382299, University of Michigan, 1999.

[15] Feng W, Kandlur D, Saha D, et al. A self2configuring RED gateway [A]. In Doshi, B., ed, Proceedings of the IEEE INFOCOM[C]. New York: IEEE Communications Society, 1999. 1320- 1328.

[16] http://www.isi.edu./nsnam/ns/.

[17] E Heshem. Analysis of Random Drop for Gateway Congestion Control. Laboratory for computer science[R], MIT, Tech Rep: LCS TR2465, 1989.

[18] 张克平. INTERNET 拥塞控制算法及其关键技术研究[D]. 西安: 西安交通大学研究生院, 2003.

作者简介:

张克平　男, 1963 年 11 月生于宁夏盐池, 博士, 高级工程师, 主要研究方向为计算机网络管理、高级路由技术、智能网络控制技术等. Email: zhangkeping@263.net.

田　辽　男, 1976 年 5 月生于宁夏固原, 现为西安交通大学计算机科学 2001 级硕士研究生, 主要研究方向为网格计算技术, Email: tl. future@263.net.

李增智　男, 1938 年 4 月出生于陕西省, 教授, 博士生导师, 主要研究领域为高速网络体系结构、信息安全等, 先后承担多项国家七五、八五、九五国家科技攻关课题, 国家自然科学基金, 863 课题等.