一种精确度加强的主动队列管理算法 BLUE+*

徐　佳[1], 殷新春[1], 杨　云[1,2]

(1. 扬州大学 信息工程学院 计算机科学与工程系, 江苏 扬州 225009; 2. 南京理工大学 计算机科学与工程系, 江苏 南京 210004)

摘　要: 主动队列管理是 IP 拥塞控制的一种重要机制, BLUE 算法作为一种典型的主动队列管理算法, 使用丢包和连接空闲事件来控制拥塞, 但是其性能还不稳定。对 BLUE 算法进行了改进, 提出了一种精确度加强的主动队列管理算法 BLUE+, 并将其应用于区分服务(DiffServ) 模型。仿真表明 BLUE+能够进一步提高对队列的控制精确度, 改善 BLUE 算法的性能, 并且能有效地支持区分服务。

关键词: 拥塞控制; 区分服务; 主动队列管理; BLUE 算法

中图法分类号: TP301.6　　**文献标识码**: A　　**文章编号**: 1001-3695(2006)09-0023-04

Precision Enhanced Active Queue Management Algorithm BLUE+

XU Jia[1], YIN Xin-chun[1], YANG Yun[1,2]

(1. Dept. of Computer Science & Engineering, College of Information Engineering, Yangzhou University, Yangzhou Jiangsu 225009; 2. Dept. of Computer Science & Engineering, Nanjing University of Science & Technology, Nanjing Jiangsu 210004)

Abstract: Active queue management algorithm is an important mechanism of IP congestion control. As a kind of typical active queue management algorithm, BLUE uses packet loss and link idle events to manage congestion, but its performance is still unstable. This paper improved BLUE algorithm, proposed a precision enhanced active queue management algorithm BLUE+ and applied BLUE+ to the model of DiffServ. Simulations experiments show that BLUE+ improves the precision of queue management and the performance of BLUE algorithm, supports the DiffServ effectively.

Key words: Congestion Control; Differentiated Services; Active Queue Management; Blue Algorithm

1　引言

为提供有效的服务质量, IETF 提出了基于 RSVP(资源预留协议) 和相应服务类型的集成服务框(IntServ[1]), 但 IntServ 需要在核心路由器中保存每个传输流的状态和处理复杂的信令协议, 导致其扩展性差。为此, IETF 又提出了以性能和扩展性为目标的粗粒度的区分服务框架(DiffServ[2])。

在网络的边缘节点, DiffServ 根据与用户的协议标记 IP 分组头部的 DSCP(DiffServ Code Point)[3]字段并将分组划分为不同的通信量聚集; 核心节点根据 DSCP 标记决定通信量聚集的逐节点行为(Per Hop Behavior, PHB)[4,5], 大大简化了中间路由器的服务机制。

基于 TCP/IP 的网络大都采用以队尾丢弃作为　包策略的被动式队列管理, 这种方法虽然简单, 却很容易引起全局同步(Global Synchronization)、队列死锁(Lock-out) 和队列延迟等问题。一种有效的方法是在路由器上进行主动队列管理(Active Queue Management, AQM), 通过探测即将到达的拥塞, 将拥塞情况反馈给发送端。在文献[6] 中 Feng 提出了一种新的主动队列管理算法 BLUE, 但它存在参数设置和吞吐率不高等问题。

收稿日期: 2005-08-18; 修返日期: 2005-10-26

基金项目: 国家自然科学基金资助项目(602730350); 国防科工委应用基础基金资助项目(K1704060511)

2　BLUE+算法

2.1　BLUE 算法性能分析

BLUE 使用丢包事件和链路空闲事件来管理拥塞。BLUE 维持了一个概率 Pmark 用以标记包, 如果由于队列溢出导致持续地丢包, BLUE 就增加 Pmark; 如果由于链路空闲导致队列空, BLUE 就减小 Pmark。BLUE 算法的基本思想如下:

```
Upon packet loss (or qlen > qlim) event:
if ((now - last_update) > freeze_time) then
Pmark = Pmark + detin
last_update = now
Upon link idle event:
if ((now - last_update) > freeze_time) then
Pmark = Pmark - detde
last_update = now
```

其中, freeze_time 参数是用来决定连续两次更新 Pmark 的时间间隔, detin, detde 是用来确定 Pmark 增加和减小时改变的幅度, qlim 是队列长度。

图 1 显示了 100 个 TCP 连接时, 典型的 BLUE 队列的情况, 其中 freeze_time 为 100ms, detin 为 0.002 5, detde 为0.000 25, 数据包大小为 1 000bits, qlim 为 50 个包, Pmark 初始值为 0。

BLUE 算法直接依据丢包和链路的使用情况来进行拥塞控制。与 RED 算法比起来, 它明显地减小了丢包率及对路由器缓存空间的需求, 但是 BLUE 算法仍然存在着一些缺点。W. Feng 和 XU Jian 等人从公平性的角度出发, 对 BLUE 算法进行了不同程度的改进[7,8], 但还存在以下一些问题:

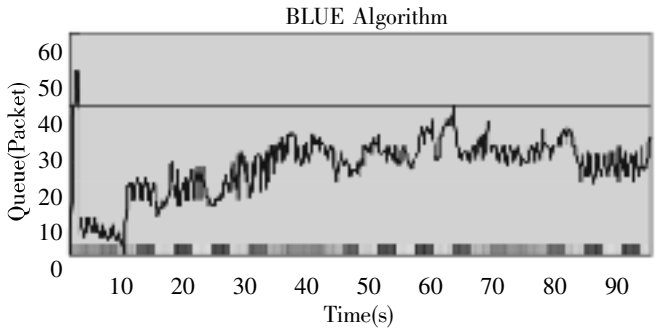

图 1 BLUE 队列在连接数不变时的情况

(1) 实验表明,当连接的主要 RTT 出现大的改变或连接的数量突然剧烈变动时,就使得所设参数无效,并将导致队列在丢包和低使用率状态间波动[9]。根据文献[10],保持流量平稳的 TCP 连接的带宽与 包率的关系是

$$B_i = \frac{\mathrm{MSS}_i \times C}{\mathrm{RTT}_i \times \sqrt{p_i}} \tag{1}$$

其中,B_i 是 TCP 连接所占用的带宽;MSS_i 是连接的最大数据包的大小;C 为常数,取 1.22;RTT_i 是最优情况下的往返时间;p_i 是连接的 包率。设瓶颈链路的带宽为 B,n 个连接共享链路,每个连接的 包率相同,则 包率为

$$P_i = \left(\frac{\mathrm{MSS}_i \times 1.22}{B_i \times \mathrm{RTT}_i}\right)^2 \tag{2}$$

若网络的瓶颈带宽为 L,连接的数量为 N。假设带宽在各连接之间公平分配,那么每个连接所获得的带宽为 L/N。代入式(2)可得:

$$P_i \approx 1.5 \times \left(\frac{\mathrm{MSS}_i \times N}{\mathrm{RTT}_i \times L}\right)^2 \tag{3}$$

从式(3)可以看出,丢包率 P 与 TCP 连接数 N 之间存在着一种二次方的变化关系,而 BLUE 算法是按线性关系来调整丢包标记 Pmark 的,所以当 TCP 连接数发生突变时,算法的控制力度不足。由于数据流在本质上是突发的,因此有必要做进一步的改进以加强控制。

(2) 从图 1 可以看出,在有突发流量到达或 TCP 连接处于启动阶段时,平均队列长度在急剧上升,极易引发连续的强制丢包。这表明在此期间,BLUE 算法对流量变化的反应过慢,这时应加大 Pmark 的值,尽量避免由于发生拥塞而导致的强制丢包;同样的原因,在流量减少时应相应地降低 Pmark 的值,以提高吞吐率和链路效率。这样算法就能自发地适应链路状态,使 Pmark 的值更加精确。

(3) 当突发流量产生后,不管是由于算法本身主动行为还是队列溢出都会产生大量丢包,其结果是:各连接的发送窗口迅速减小,队列长度急剧下降,虽然此时链路流量已经在下降,队列长度也在很快地减小,但由于没有到达刷新时间,算法仍然按照较大的丢包率在工作,很快便使得队列为空。这样,传输延迟增加,链路利用率也大大降低。一种有效的方法是在因连续丢包而造成数据流迅速减少时,提高 BLUE 算法的刷新率,使算法对数据流的反应更加敏感。

2.2 BLUE$^+$ 队列算法

针对 BLUE 算法的缺陷,我们提出了一种改进的 BLUE 算法。具体改进有三点:

(1) BLUE 算法的核心控制语句(Pmark = Pmark + detin 或 Pmark = Pmark - detde)可以转换为一个通用的表达式:

$$\mathrm{Pmark} = a \times \mathrm{Pmark} + b \times \mathrm{det} \tag{4}$$

其中,a,b 为精确参数。根据式(3),丢包率 P 与 TCP 连接数 N 之间存在着一种二次方的变化关系,因此,可以将 a 表示成一个关于连接数的二次方形式,而 b 可以根据队列长度分阶段地调整步长。在此我们利用平均队列的长度来反映连接数的数量。分别设置一个队列的高位标志 Maxpos 和一个低位标志 Minpos,当平均队列长度大于 Maxpos 时,队列进入高位区;当平均队列长度小于 Minpos 时,队列进入低位区。此时,相应的丢包标记以二次方的速度变化,具体计算方法如下:

$$\mathrm{Pmark} = \left(\frac{\mathrm{AvgQ}}{\mathrm{Last_AvgQ}}\right)^2 \times \mathrm{Pmark} \tag{5}$$

其中,Last_AvgQ 为上一次刷新时的平均队列长度。此时,

精确参数 $a = \left(\frac{\mathrm{AvgQ}}{\mathrm{Last_AvgQ}}\right)^2$ (6)

(2) 当队列在高位区且队列长度还在继续增长时,根据此时的队列长度相应地增大丢包标记的值,尽量避免拥塞的发生,以抑制路由器产生强制性的丢包。队列长度越大,丢包标记的值就越大。当 AvgQ - Minpos 的值为 qlim - Minpos 的 70% ~79% 时,将步长参数 detin 的值修正为初始值的两倍;当 AvgQ - Minpos 的值为 qlim - Minpos 的 80% ~89% 时,将 detin 的值修正为初始值的三倍;当 AvgQ - Minpos 的值为 qlim - Minpos 的 90% ~99% 时,将 detin 的值修正为初始值的四倍;当 AvgQ - Minpos 的值为 qlim - Minpos 的 100% 时,将 detin 的值修正为初始值的五倍;其他情况则采用初始值。具体计算方法如下:

令 $R1 = \mathrm{int}\ 10 \times \frac{\mathrm{AvgQ} - \mathrm{Minpos}}{\mathrm{qlim} - \mathrm{Minpos}}$ (7)

$$\mathrm{detin} = (R1 - 5) \times \mathrm{detin} \tag{8}$$

此时,精确参数 $b = R1 - 5$ (9)

同样地,当队列处于低位区且队列长度还在继续降低时,根据此时的队列长度相应地降低丢包标记的值,提高算法的吞吐率以降低传输时延和增加链路利用率。队列长度越小,丢包标记的值就越小。当 AvgQ - Minpos 的值为 Maxpos - Minpos 的 20% ~29% 时,将步长参数 detde 的值修正为初始值的两倍;当 AvgQ - Minpos 的值为 Maxpos - Minpos 的 10% ~19% 时,将 detde 的值修正为初始值的三倍;当 AvgQ - Minpos 的值为 Maxpos - Minpos 的 0% ~9% 时,将 detde 的值修正为初始值的四倍;其他情况则采用初始值。具体计算方法如下:

令 $R2 = \mathrm{int}\ 10 \times \frac{\mathrm{AvgQ}}{\mathrm{qlim} - \mathrm{Minpos}}$ (10)

$$\mathrm{detde} = (4 - R2) \times \mathrm{detde} \tag{11}$$

此时,精确参数 $b = -(4 - R2)$ (12)

(3) 在队列下降阶段,提高 BLUE 算法的刷新频率,即减少 freeze_time 的值,以避免因对队列的反应迟缓而造成的大量丢包。队列的增长幅度越大则刷新的速度越快。设 qlim 为队列缓冲区大小,当 last_AvgQ - AvgQ 的值为 qlim 的 0% ~9% 时,freeze_time 的值为初始值;当 last_AvgQ - AvgQ 的值为 qlim 的 10% ~19% 时,freeze_time 的值为初始值的 1/2;当 last_AvgQ - AvgQ 的值为 qlim 的 20% ~29% 时,freeze_time 的值为初始值的 1/3;当 last_AvgQ - AvgQ 的值为 qlim 的 30% ~39% 时,freeze_time 的值为初始值的 1/4;其他情况采用默认值,默认值为初始值的 1/5。具体方法如下:

令 $K = \mathrm{int}\ 10 \times \frac{\mathrm{last_AvgQ} - \mathrm{AvgQ}}{\mathrm{qlim}}$ (13)

$$freeze_time = \frac{freeze_time}{K+1} \tag{14}$$

BLUE$^+$算法根据队列的长度以及队列的变化情况分阶段地动态调整各参数的值, 以达到精确控制的目的。BLUE$^+$算法的参数调整过程实现如下:

```
Upon AvgQ update:
if ( AvgQ > = Maxpos)  then
   Pmark = [ AvgQ/last_AvgQ] 2Pmark
   if ( AvgQ-last_AvgQ > 0)  then
   R1 = int[ 10*  ( AvgQ-Minpos) /( qlim-Minpos) ]
      if ( 7 <  = R1 <  = 10)
         detin = ( R1 - 5) detin
      else
         detin = initial value
      end if
   end if
end if
if ( AvgQ <  = Minpos)  then
   Pmark = [ AvgQ/last_AvgQ] 2Pmark
   if ( AvgQ-last_AvgQ < 0)  then
   R2 = int[ 10*  AvgQ/( qlim-Minpos) ]
      if ( R2 < 3)
         detde = ( 4 -  R2) detde
      else
         detde = initial value
      end if
   end if
end if
if ( AvgQ-last_AvgQ < 0)  then
   K = int[ 10 *  ( last_AvgQ-AvgQ)  / qlim]
      if( K < 4)
         freeze_time = freeze_time/ ( K + 1)
      else
         freeze_time = freeze_time/5
      end if
   end if
```

算法中的 detin, detde 和 freeze_time 均是以原值作为参照来调整的, 所以在 BLUE$^+$每次调用完 BLUE 算法后, 这些值均要求复位成初始值。

2.3　基于 DiffServ 的 BLUE$^+$

2.3.1　DiffServ 路由器体系结构

适应 DiffServ 的路由器按功能分主要由以下四个机构构成: 接入控制机构、流量调节机构、数据包分类机构和流量控制机构(数据包调度装置/队列管理装置) 。接入控制机构的作用是使预留的聚合流量带宽不会超出路由器的容量。当应用没有预留资格或网络拥塞时, 接入控制机构拒绝提供新服务。流量调节机构原则上都设在 DS(Differentiated Service, DiffServ) 域的入口和出口, 流入域的 DS 流或者行为集合依照事先规定的流量控制框架(Profile) 进行流数据包的整形, 操作通常有丢弃、赋予 DS 字节等。数据包分类机构通常被设置在流量调节机构和流量控制机构的前面, 根据数据包报头的内容选择容纳这个数据包的调度装置的队列。流量控制机构实际上是数据包调度装置和队列管理装置的组合, 它通常分为快速转发 EF PHB(Expedited Forwarding PHB) 、可靠转发 AF PHB(Assured Forwarding PHB) 、预置 PHB(Best Effort PHB) 等逐节点行为。一个完整的 DiffServ 路由器体系结构如图 2 所示。

2.3.2　DiffServ 业务量调节模型

DiffServ 的目标在于简单有效, 以满足实际应用对可扩展性的要求。其实现途径是: 将状态相关的控制转移到网络边界节点, 简化网络内部节点的服务机制, 将复杂性推到网络边缘。网络边界节点进行流的分类、整形、聚集等, 在内部节点实施基于 PHB 的与状态无关的调度转发。区分服务业务量调节结构如图 3 所示。

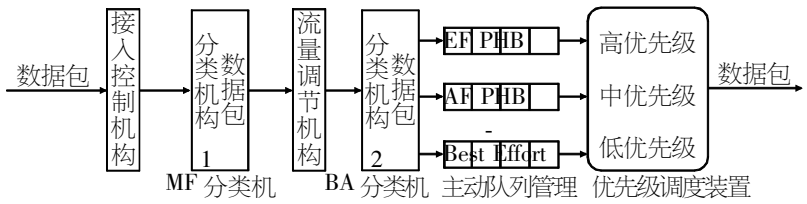

图 2　DiffServ 路由器体系结构

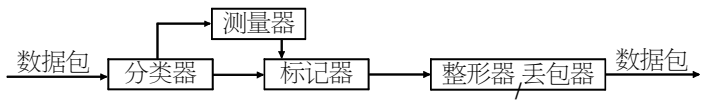

图 3　DiffServ 业务量调节结构框图

业务量调节器包括下列组件: 测量器、分类器、标记器、整形器和丢包器。业务流首先经过分类器, 分类器负责对传进来的输入数据包进行分类, 以便决定传向测量器还是标记器, 最后进入基于某种机制的整形器和丢包器, 目的是调节流量。

2.3.3　流量调节与标记

IETF 的区分服务结构模型中, 在 DS 域的入口和出口处设置了流量调节机构, 目的是支持可控负载服务, 我们采用了令牌桶机制在网络接入点进行侦察(Policing) 和标记(Marking) 处理, 使进入网络的流量符合条件。为了确定流量的负载特征, 设平均速率为 r(Bps) , 桶的深度为 b(Bytes) , 最大速率为 p(Bps) , 最大数据包长度为 M(Bytes) , 最小数据包长度为 m(Bytes) 。当分组未到达时, 平均每秒有 r 个令牌补进, 如果令牌桶已满, 将添加的令牌溢出; 当分组到达时, 突发计数器记录当前的突发量, 检查令牌桶中是否有足够的令牌传输该组。如果令牌够用, 则分组被标为顺从流(Compliant) , 并从桶中删除与分组长度相等的令牌; 否则为非顺从流(Non-Compliant) , 不进行标记。

传统的 BLUE 算法无法区分数据包的优先级, 而 BLUE$^+$算法对此进行了改进。它能根据数据包的标记作出相应的丢弃策略, 以保证顺从流的服务质量。具体方法如下: 对于标记包, 不根据 Pmark 来丢弃数据包, 而只有在队列溢出时才丢弃, 采用 BLUE$^+$算法中的 Pmark 来对非标记包进行丢弃。算法如下:

```
if( packet is marked)  then do
      begin
      if( qlen < qlim)
         transmit this packet
      else
         drop this packet;
      end;
if( packet is unmarked)  then
drop this packet as Pmark;
```

3　仿真及算法的性能分析

为了验证 BLUE$^+$的效率性, 对 BLUE$^+$进行了仿真试验。仿真试验在网络仿真软件 NS-2[11] 上进行。试验的网络拓扑结构如图 4 所示。假定 N_1 到 N_2 之间的链路为瓶颈链路, 设定其延迟为 100ms, 带宽为 1.0Mbps, 其他链路的带宽为 10Mbps, 延迟为 1ms。$S_0 \sim S_4$ 为发送端, $R_0 \sim R_4$ 为接收端, TCP 连接的版本是 TCP Reno。

首先, 我们对 BLUE 和 BLUE$^+$在连接数突变时的队列情况进行了对比。其缺省参数设置如下: freeze_time = 100ms, de-

tin = 0.002 5, detde = 0.000 25, qlim = 50, packetsize = 1 000bits, 初始 Pmarkt = 0, 仿真时间为 100s。TCP 连接数在 0 ~30s 和 60s ~100s 之间为 100, 在 30s ~60s 之间为 40, 试验结果如图 5 和图 6 所示。从图中我们可以看出, 当 TCP 连接数突变时 BLUE 算法出现了长时间的空队列现象, 并且队列溢出情况也出现较多, 吞吐率明显降低。BLUE$^+$ 算法相对 BLUE 算法能够明显地减少空队列现象的发生, 队列溢出也得以减少, 从而在一定程度上减少了丢包率。

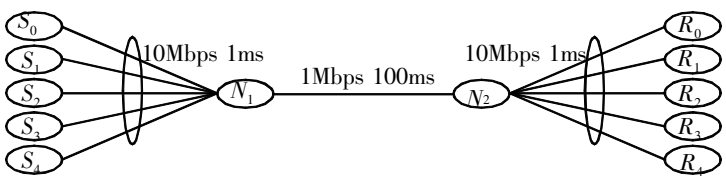

图 4 网络拓扑

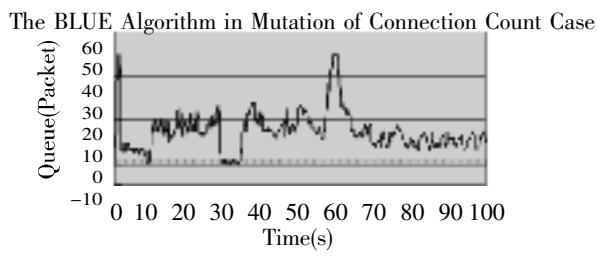

图 5 在连接数突变情况下的 BLUE 算法

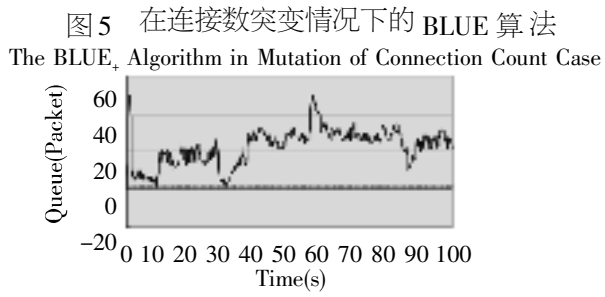

图 6 在连接数突变情况下的 BLUE$^+$ 算法

为了检验 BLUE$^+$ 的性能, 我们对 RED, BLUE, BLUE$^+$ 的丢包率和吞吐率进行了对比。试验中, 网络拓扑和 BLUE 的参数配置不变, BLUE$^+$ 中的 Minpos 和 Maxpos 分别为 qlim 的 20% 和 80%, 其他参数与前述 BLUE 相同; RED 算法的缺省参数为: maxp = 0.1, qlim = 50 个包, minth 和 maxth 分别为 qlim 的 25% 和 75%, wq = 0.002。试验结果如图 7、图 8 所示。在丢包率方面, RED 算法明显要高于 BLUE 和 BLUE$^+$, 相对于 BLUE 来说, BLUE$^+$ 的丢包率略微减少; 在吞吐率方面, RED 算法表现得最好, 但付出的代价是较高的队列时延, BLUE$^+$ 的吞吐率在绝大多数情况下比 BLUE 有明显的提高。

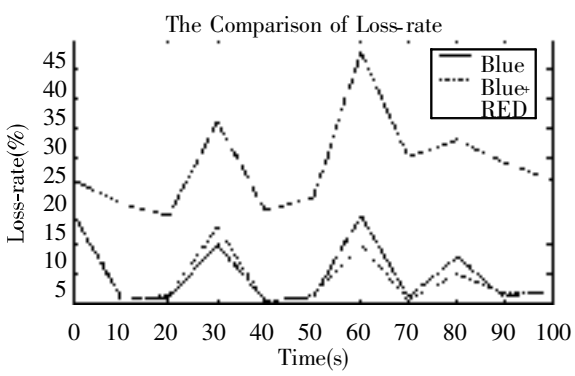

图 7 丢包率对比

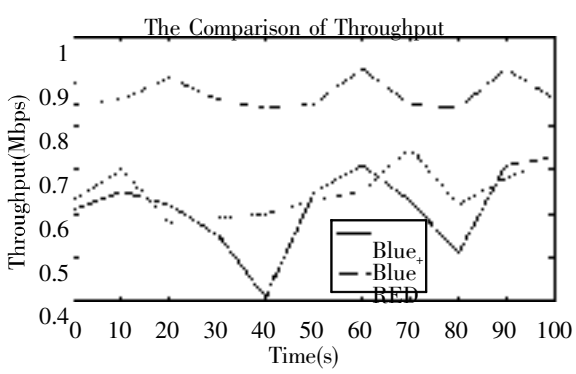

图 8 吞吐率对比

为了验证在区分服务中 BLUE$^+$ 算法的性能, 采用图 4 的网络模型实现了该算法, BLUE$^+$ 队列的各项参数与上面实验相同, 令牌桶深度 b = 50ms ×预留速率。我们实验了两个保证带宽的 TCP 连接和两个尽力传输的 TCP 连接竞争瓶颈链路带宽的情况, 实验结果如图 9 所示。从图中可以看出, 基于 BLUE$^+$ 的拥塞控制方法可以保证预留连接的带宽, 尽力服务的连接也可以公平分配剩余带宽。

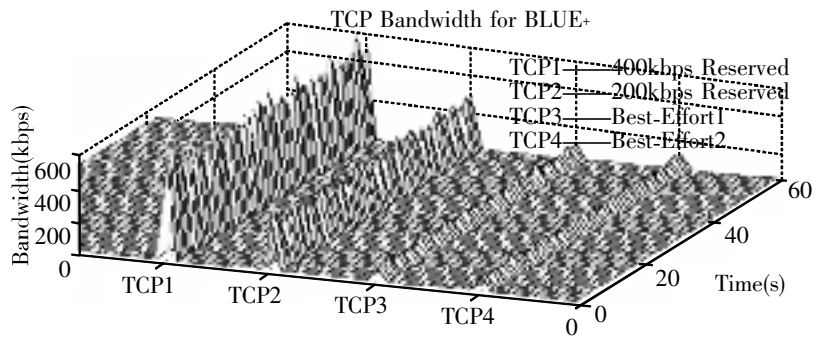

图 9 在区分服务环境下各 TCP 连接的吞吐率

4 结论

根据平均队列长度, 对 BLUE 算法的步长参数 detin, detde 进行动态调整, 并且依据 TCP 友好公式中丢包率与连接数之间的关系, 调整 Pmark 值。此外在队列长度减少时提高算法的刷新速度, 避免了大量丢包。在此基础上, 研究了在区分服务环境下基于 BLUE$^+$ 的 TCP 拥塞控制方法。仿真表明: 在 TCP 连接数突变的情况下, BLUE$^+$ 精确度加强算法的性能明显要好于 BLUE 算法, 并且能提高吞吐率, 降低丢包率; 在区分服务环境下能有效地工作。因此, BLUE$^+$ 是一种稳定、有效的队列管理算法。

参考文献:

[1] raden B, Clark D, Shenker S. Integrated Services in the Internet Architecture: An Overview[S]. RFC 1633, 1994.

[2] Blake S, Black D, *et al*. An Architecture for Differentiated Services [S]. RFC 2475, 1998.

[3] Nichols K, Blake S, Baker F, *et al*. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers[S]. IETF RFC 2474, 1998.

[4] Jacobson V, Nichols K, Poduri K. An Expedited Forwarding PHB [S]. IETF RFC 2598, 1999.

[5] Heinanen J, Baker F, *et al*. Assured Forwarding PHB Group[S]. IETF RFC 2597, 1999.

[6] W Feng, D D Kandlur, *et al*. BLUE: A New Class of Active Queue Management Algorithms[R]. Technical Report CSETR-387-99, Department of EECS, University of Michigan, 1999.

[7] W Feng, D Kandlur, D Saha, *et al*. Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness[C]. Alaska: Proceedings of INFOCOM, 2001. 1520-1529.

[8] 徐建, 李善平. 数据流的活动队列管理算法: MBLUE[J]. 电子学报, 2002, 30(11): 1732-1736.

[9] 张顺亮, 叶澄清, 李方敏. 一种加强的主动队列管理算法——EBLUE[J]. 通信学报, 2003, 24(11): 109-115.

[10] Mathis M. The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm[J]. Computer Communication Review, 1997, 27(3): 67-82.

[11] NS-2[EB/OL]. http://www.isi.edu/nsnam/ns/.

作者简介:

徐佳(1980-), 男, 江苏常州人, 硕士研究生, 主要研究方向为 IP QoS、拥塞控制; 殷新春(1962-), 男, 江苏姜堰人, 教授, 硕士生导师, 博士, 主要研究方向为信息安全、密码学; 杨云(1960-), 男, 江苏扬州人, 副教授, 硕士生导师, 博士后, 主要研究方向为网络安全、拥塞控制。