几种主动式队列管理算法的比较研究

吴春明,姜 明,朱淼良
(浙江大学人工智能研究所,浙江杭州 310027)

摘 要: 主动式队列管理(Active Queue Management,AQM)技术是 IETF 为了解决 Internet 拥塞控制问题而提出的一种路由器缓存管理技术. 本文对几种主要 AQM 算法 RED、BLUE、ARED 和 SRED 的性能在基于 ns-2 仿真实验的基础上进行了比较研究. 研究的性能包括队列长度、丢包概率、丢包率、连接数对吞吐量的影响及缓冲区大小对链路利用率的影响等. 仿真结果表明 BLUE、ARED 和 SRED 在这几方面的性能都要优于 RED 算法.

关键词: Internet; 主动式队列管理 (AQM); 随机早期检测算法 (RED); (BLUE); 稳定随机早期检测算法 (ARED); 主动随机早期检测算法 (SRED)

中图分类号: TP393 **文献标识码**: A **文章编号**: 0372-2112 (2004) 03-0429-06

A Research on Some Acitve Queue Management Algorithms

WU Chun-ming, JIANG Ming, ZHU Miao-liang
(AI Institute of Zhejiang University, Hangzhou, Zhejiang 310027, China)

Abstract: To address the problem of TCP end-to-end congestion control, IETF advocates deploying active queue management (AQM) mechanisms in the TCP/IP networks. Several popular AQM algorithms, such as RED (Random Early Detection), BLUE, ARED (Adaptive RED) and SRED (Stabilized RED), are compared based on simulations and the performance metrics used in the study are queue size, packet drop probability, packet loss rate, throughput on different connections and link utilization on different buffer size. The simulation results indicate that in all these aspects, the performance of BLUE, ARED and SRED are better than that of RED. Furthermore, ARED and SRED can stabilize queue size and control packet loss rate effectively thus they maintain high link utilization, bounded delays and more certain buffer provision.

Key words: Internet; AQM; RED; BLUE; ARED; SRED

1 引言

众所周知,Internet 的设计机制是基于无连接的端到端数据包交换技术以及"尽力而为"(best effort)的服务模型. 这种机制的最大优势是设计简单、扩展性好,但其存在的一个主要问题是会产生拥塞崩溃(congestion collapse)[1]现象,使得链路利用率大大降低,因此 Internet 必须提供拥塞控制机制. TCP 端到端的拥塞控制机制是确保 Internet 鲁棒性(robustness)的重要因素. 在网络发生拥塞时,TCP 源端会降低数据发送速率,从而使得大量的 TCP 连接能够共享一条拥塞的链路. Internet 的成功已证明 TCP 拥塞控制机制在防止拥塞崩溃方面取得了巨大的成功. 但这种机制的有效性依赖于两个基本假设:(1)几乎所有的流都采用了拥塞控制机制;(2)这些流采用的拥塞控制机制是相同的或者大体上是相同的,也就是说在相似的环境下按可比条件(丢包率、回路响应时间 RTT、最大传输单元 MTU 等)不会占用比 TCP 流更多的带宽,即所谓的 TCP 友好(TCP-friendly)[2]流.

但随着近十年来计算机网络的爆炸式增长,特别是多媒体业务的广泛应用,导致以上假设不再成立. 因此 Internet 不可能再仅仅依靠端节点提供的 TCP 拥塞控制机制. 而路由器直接掌握着 Internet 上各种传输的信息,能够有效地检测到拥塞;另外,路由器能够全面地审视各个流对拥塞造成的影响,从而决定将拥塞信息通知哪个源端. 因此,网络本身也必须参与到拥塞控制中去. 目前,基于路由器的拥塞控制和 TCP 端到端拥塞控制相结合的方法已经成为解决 Internet 拥塞控制问题的一个主要途径.

2 被动式队列管理和主动式队列管理

2.1 队列管理简介

路由器是基于包交换的设备,每个端口采用带宽统计复用. Internet 数据的突发本质[5,6]使得路由器中必须配置一定大小的队列以提高链路利用率、减少丢包. 目前,路由器有两类和拥塞控制相关的队列算法:队列调度算法和队列管理算法[3]. 前者决定下一个要发送哪个包,主要用来管理各流之间带宽的分配. 后者主要是在网络发生拥塞时通过丢包来管理队列长度. 对队列长度进行管理直接影响到路由器的拥塞

收稿日期:2003-01-27;修回日期:2003-04-10

控制能力和 QoS 能力.

目前的队列管理机制可以分为两大类:被动式队列管理(Passive Queue Management, PQM)和主动式队列管理(Active Queue Management, AQM).

2.2 被动式队列管理及其局限性

管理队列长度的传统方法是对每个队列设置一个最大值(以包为单位),然后接受包进入队列直到队长达到最大值,接下来到达的包就要被拒绝进入队列直到队长下降.这种技术也就是传统的"去尾"(drop-tail)算法.由于这种方法是在队列满了之后被迫丢包,因此称为被动式队列管理.

虽然"去尾"算法在当前 Internet 上得到了广泛的使用,但其存在两个重要问题[3]:(1)死锁(lock-out)现象:在某些情况下,由于同步或其他定时作用的结果,"去尾"算法会让某个流或者少数几个流独占队列空间,阻止其他流的包进入队列.(2)满队列(full queues)问题:由于"去尾"算法只有在队列满时才会发出拥塞信号,因此会使得队列在相当长时间内处于充满(或几乎充满)的状态.而 Internet 数据的突发性使得队列在满状态下会产生"TCP 全局同步"(TCP global synchronization)现象.

2.3 主动式队列管理简介

在当前的 Internet 上,丢包是对端节点进行拥塞通知的主要机制,解决被动式队列管理问题的方法便是在队列满之前丢包,这样端节点便能在队列溢出前对拥塞作出反应.这种方法便称为主动式队列管理.

AQM 是 IETF 推出的基于 FIFO 调度策略的队列管理机制[3],它使得路由器能够控制在什么时候丢多少包,从而有效地管理队列长度,以支持端到端的拥塞控制.

自从 IETF 提出了 AQM 技术并推荐了 RED 算法以来,已产生了许多 AQM 算法,本文将使用 ns-2 对一些主要的 AQM 算法,包括 RED、BLUE、ARED 和 SRED 进行研究,通过进行仿真实验对上述 AQM 算法在队列长度、丢包概率(packet drop probability)、丢包率(packet loss rate)以及缓冲区大小和连接数对吞吐量的影响等几个方面的性能进行比较,以加深我们对 AQM 算法的理解.

3 主动式队列管理算法

3.1 随机早期检测算法 RED

最早提出的 AQM 算法是随机早期检测(Random Early Detection, RED)算法[4],也是目前最常用的一种 AQM 算法.RED 的基本思想是路由器通过监控队列的平均长度来探测拥塞.一旦发现拥塞逼近,就随机地选择源端来通知拥塞,使它们在队列溢出之前降低发送数据速率,以缓解网络拥塞.

RED 算法主要包括两步,首先计算平均队列长度,然后计算丢弃包的概率.RED 在计算平均队长 avg_q 时,采用了类似低通滤波器(low-pass filter)带权值的方法:

$$avg_q = (1 - w_q) \times avg_q + q \times w_q$$

其中,w_q 为权值,q 为采样测量时实际队列长度.从而"过虑"掉由于 Internet 数据的突发性导致的短期队长变化,尽量反映长期的拥塞变化[4].权值 w_q 相当于低通滤波器的时间常数,它决定了路由器对输入流量变化的反应程度.计算平均队长的目的就是为了反映拥塞程度并据此来计算丢包概率.

RED 有两个和队列长度相关的阈值:min_{th} 和 max_{th}.当有数据包达到路由器时,RED 计算出平均队长 avg_q.若 avg_q 小于 min_{th},则没有包需要丢弃;当 $min_{th} \le avg_q \le max_{th}$ 时,计算出概率 P,并以此概率丢包;当 $avg_q > max_{th}$ 时,所有的包都被丢弃(如图 1 所示).

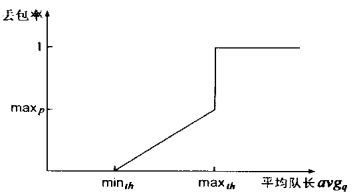

图 1 平均队长和丢包概率的关系

计算丢包概率 P 的方法如下:

$$Pb = max_p \times (avg_q - min_{th}) / (max_{th} - min_{th})$$

$$P = Pb / (1 - count \times Pb)$$

概率 P 不仅和 avg_q 有关,而且还和从上一次丢包开始到现在连续进入队列的包的数量 $count$ 有关.随着 $count$ 的增加,下一个包被丢弃的可能性也在缓慢增加.这主要是为了在到来的包之间均匀间隔地丢包,避免连续丢包,以消除对突发流的偏见和产生全局同步现象.

3.2 BLUE

BLUE[5]与 RED 最大的区别在于前者使用实际队长来反映拥塞状况,并且使用丢包事件和链路空闲事件来管理拥塞,而 RED 则使用平均队长.BLUE 使用了一个概率 P 用以丢弃包.如果由于队列溢出导致连续地丢包,BLUE 就增加 P,因而增加了向源端发送拥塞通知的速度.相反,如果由于链路空闲导致队列空,BLUE 就减小 P.这样 BLUE 就能有效地控制发送拥塞通知信息的速度.

为了避免丢包过于激进,BLUE 设置了一个最小时间间隔 freeze_time,在该时间间隔内,概率 P 只能更改一次.BLUE 另外两个重要参数是 $d1$、$d2$,其中,$d1$ 决定了当队列溢出时 P 增加的量,$d2$ 决定了当链路空闲时 P 减少的量.一般来说,$d1$ 要比 $d2$ 大很多,这主要是因为,通过赋予丢包事件更大的比重,BLUE 能够对流量大量迅速地增加很快作出反应[5].

3.3 Adaptive RED(ARED)

RED 算法的拥塞指示的发送速率是由参数 max_p 来体现的.如果 max_p 太大,那么丢包比例主要就是由于早期拥塞检测中产生的丢包造成的;如果 max_p 太小,丢包主要就是由于队列溢出造成的.RED 的一个弱点是平均队长对流量负荷和参数设置很敏感,导致队列长度波动较大,结果造成平均排队时延对流量负荷和参数设置很敏感.

ARED[6,7]提出了一种自动配置机制,根据流量的变化来配置参数 max_p.ARED 的基本思想就是通过检查平均队长的变化来决定 RED 是应更激进还是更保守,从而尽量保持平均队长在 min_{th} 和 max_{th} 之间.具体地说,如果平均队长是在 min_{th} 附近振荡,说明拥塞控制太激进了,那么就减小 max_p,$max_p = max_p / \alpha$;如果在 max_{th} 附近振荡,说明拥塞控制太保守了,那么就增大 max_p,$max_p = max_p * \beta$,其中 $\beta > \alpha > 1$.

ARED 是对 RED 改动很小的一种算法,它保留了 RED 的

基本结构,只需调节参数 $\max_p$ 从而保持平均队长在 $\min_{th}$ 和 $\max_{th}$ 之间,消除了 RED 的队列延时问题和参数敏感性问题.

3.4 Stabilized RED(SRED)

SRED[8]的设计目的是保持 FIFO 队列长度的稳定而与活跃流(active flow)数量无关. SRED 的基本思想就是进行比较. 当有一个包到达队列时,就从队列中随机选出的一个包进行比较,若这两个包属于同一个流,则称为"击中"(hit). SRED 设计了一种数据结构,称为"僵尸"列表(Zombie List),相当于一种流 Cache,其中记录了最近流经该队列的 M 个流. 每个"僵尸"包含了两个状态信息,其中 count 表示被击中的次数;时间戳则记录了该"僵尸"产生的时间或最近一次被"击中"的时间(count 大于 0).

起初列表为空,当有一个包到达时,则将该包的流标识(源、目的地址等)加入列表,这样,就产生了一个新的"僵尸". 一旦"僵尸"列表满了,那么就将新来的数据包和"僵尸"列表中随机选出的"僵尸"进行比较:(1)如果击中,则此僵尸的 count 加 1,时间戳改为当前的时间;(2)如果未击中,则以某个概率用新数据包的流标识覆盖选出来的僵尸.

SRED 用一个函数 $Hit(t)$ 来记录第 t 个包是否击中,如果击中,则 $Hit(t)$ 取值为 1,否则为 0. 然后计算第 t 个数据包到达时击中的概率 $P(t)$:

$$P(t) = (1 - \alpha) \times p(t - 1) + \alpha Hit(t), 0 < \alpha \ll 1$$

其中 α 是一个常数. $P(t)^{-1}$ 可以被认为是在第 t 个包到达前很短时间内对活跃流数量的估计[8]. 计算数据包的丢弃概率依赖于队列的占用情况和活跃流的估计值.

$$p_{sred}(q) = \begin{cases} p_{\max}, & (1/3)B \le q < B \\ (1/4)\, p_{\max}, & (1/6)B \le q < (1/3)B \\ 0, & q < (1/6)B \end{cases}$$

$$P_{zap} = P_{sred}(q) \times \min\left(1, \frac{1}{(256 \times P(i))^2}\right)$$

其中,B 是队列大小,q 是当前队列长度. P_{zap} 为丢包概率.

4 仿真实验与性能比较

4.1 仿真实验的配置

为了对 RED、BLUE、ARED 和 SRED 之间的性能进行比较,我们在 ns-2[9]平台上进行了一系列的仿真. 仿真实验在 P Ⅳ1.5G,128M 内存的机器上进行,操作系统为 Red Hat Linux 7.2.

实验中使用的网络拓扑如图 2 所示,其中包含了 10 个源子网、10 个目的子网和两台路由器. 图中所有链路带宽都是 25Mbps,这样路由器 $R1$ 和 $R2$ 之间形成一条瓶颈链路. 每个子网包含了 50 个 TCP 源,这样整个网络中共有 500 个 TCP 源,其输入模型是 FTP 模型.

我们将通过仿真实验对 RED、BLUE、ARED 和 SRED 在队列长度、丢包概率、丢包率以及 TCP 连接数和缓冲区大小对吞吐量的影响等五个方面的性能进行比较. 前四个实验中用到的参数如表 1 所示. 所有这些实验时间都是 100 秒,每 50ms 记录一次数据.

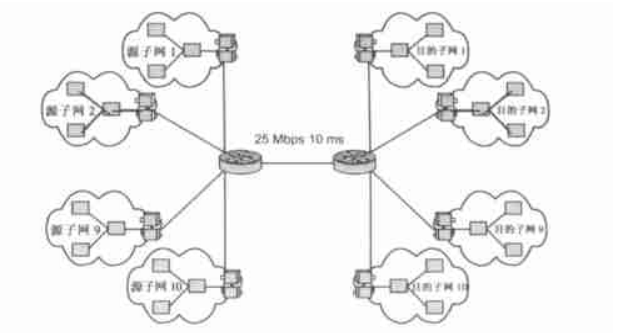

图 2　实验中用到的网络拓扑

表 1　实验中用到的参数

算法	参量	值
RED	最小阈值 $\min_{th}$	80
	最大阈值 $\max_{th}$	280
	权值 w_q	0.002
	最大丢包概率 $\max_p$	0.1
	缓冲区大小 B	350 packets
BLUE	初始丢包概率	0.005
	丢包概率增加步长(d_1)	0.00025
	丢包概率减小步长(d_2)	0.0025
	freeze_time	0.01s
	缓冲区大小 B	350 packets
SRED	僵尸列表中僵尸数目	1000
	最大丢包概率 $P_{\max}$	0.15
	僵尸列表更新概率 P	0.25
	缓冲区大小 B	500 packets
ARED	最小阈值 $\min_{th}$	80
	最大阈值 $\max_{th}$	280
	权值 w_q	0.002
	最大丢包概率 $\max_p$	0.1
	缓冲区大小 B	350 packets
	$\max_p$ 减小 α	2
	$\max_p$ 增加 β	3

4.2 队列长度的比较

图 3 中显示的是 RED、BLUE、ARED 和 SRED 四种算法队列长度变化的过程. 图 4 则是在这四中算法下,队列长度频率分布情况,其中采样间隔为 5. 我们可以很明显地发现在 RED 和 BLUE 算法中队列长度的变化很大;而 ARED 和 SRED 中队列长度的变化则相对较稳定.

我们认为,RED 算法下队列长度之所以会波动非常频繁并且范围很大,主要是因为当有大量的活跃的 TCP 连接时,总的流量往往突发性非常高[8],队长的增减非常迅速,而 RED 还没有来得及作出很好的反应,也就是说 RED 不能有效地估计拥塞的严重程度. 对于 BLUE 来说,由于发生丢包事件后会相对大地增加丢包概率,从而会产生连续丢包,大大降低源端发送速率,并且也使得队长发生较大的波动. 而 ARED 和 SRED 的目的都是为了保持队长的稳定. 前者的设计目标是保持队列长度稳定在 $\min_{th}$ 和 $\max_{th}$ 之间;后者则是保持队列长度的稳定而和连接数无关. 从实验中我们可以看到 ARED 和 SRED 都较好地达到了目的.

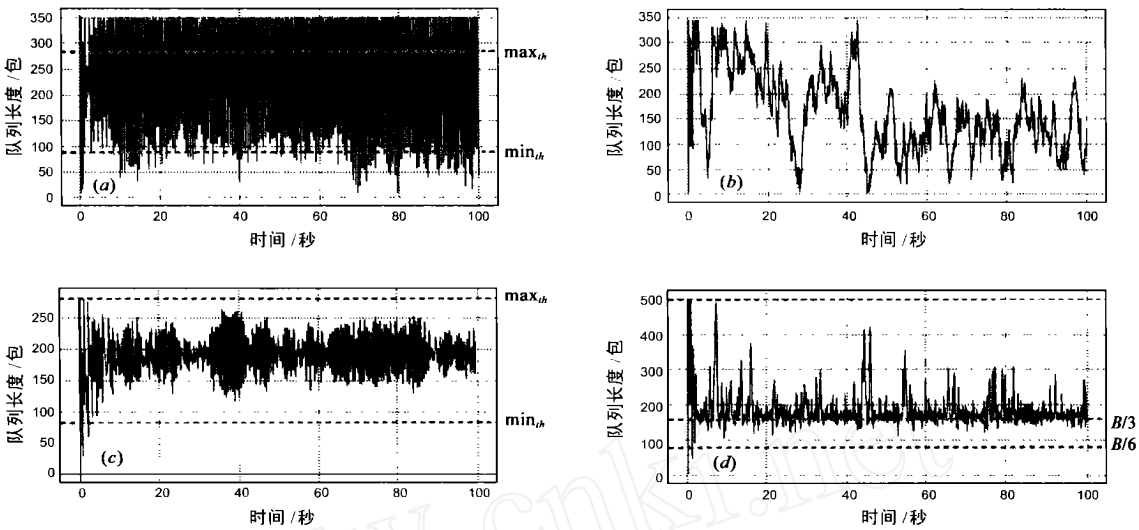

图 3 500 个 TCP 连接时的队列长度布 (*a*) RED;(*b*) BLUE;(*c*) ARED;(*d*) SRED

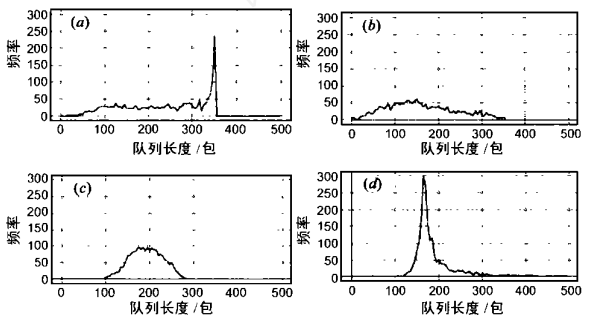

图 4 500 个 TCP 连接时的队列长度频率分布
(*a*) RED;(*b*) BLUE;(*c*) ARED;(*d*) SRED

4.3 丢包概率和丢包率的比较

图 5 中显示了这四种算法情况下的丢包概率. RED 算法的丢包概率波动很频繁,范围很大,这和其队长的波动是一致的. 而 BLUE 的变动则很小,这说明其对拥塞变化的反应较慢,从而使其队长发生较大波动. ARED 和 SRED 的丢包概率则波动范围较小. SRED 的丢包概率波动范围则明显地稳定在 P_{max} 和 $P_{max}/4$ 之间. 图 6 中显示了这四种算法情况下的丢包率. RED 算法的丢包率仍然波动范围很大,并且其丢包率也是最高的. 我们还可以发现一个有趣的现象,ARED 的丢包概率(如图 5)和丢包率(如图 6)以及队列长度(如图 3)的变化基本是一致的,区别只是丢包率的波动范围和值较小.

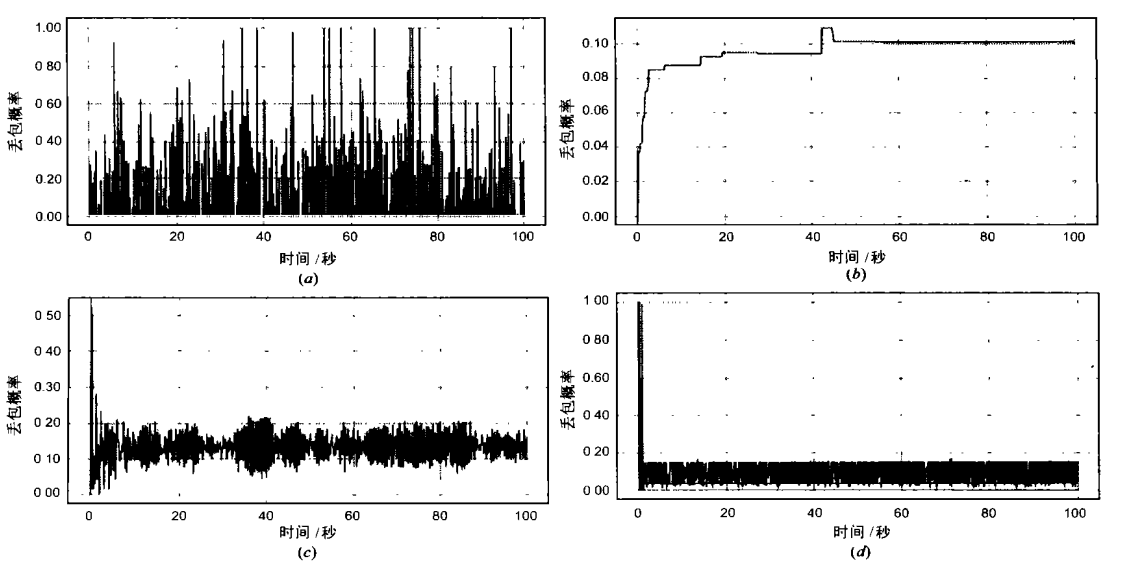

图 5 500 个 TCP 连接时的丢包概率 (*a*) RED;(*b*) BLUE;(*c*) ARED;(*d*) SRED

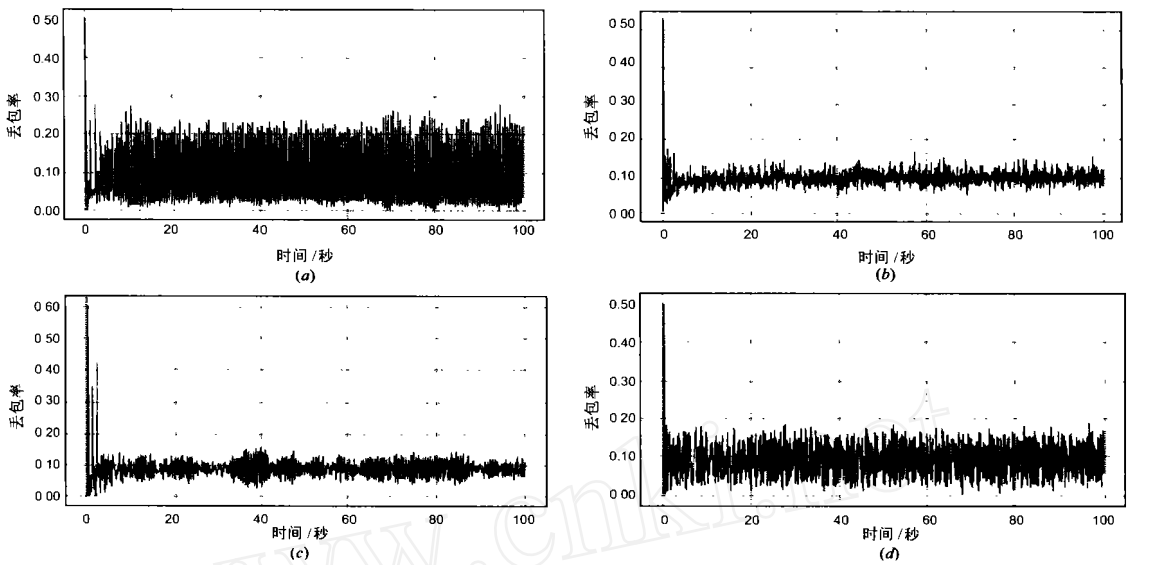

图 6　500 个 TCP 连接时的丢包率 (*a*) RED; (*b*) BLUE; (*c*) ARED; (*d*) SRED

4.4　TCP 连接数对吞吐量的影响

图 7 显示了不同数目 TCP 连接情况下吞吐量的变化情况. TCP 连接的范围从 20 个到 200 个,间隔为 20. 我们可以看到 ARED、BLUE 和 SRED 都达到了很高的吞吐率. 并且 TCP 连接数目的变化对其吞吐率影响不是很大. 相比之下,TCP 连接数目在 20 到 100 之间对 RED 算法下的吞吐率则有相对较大影响.

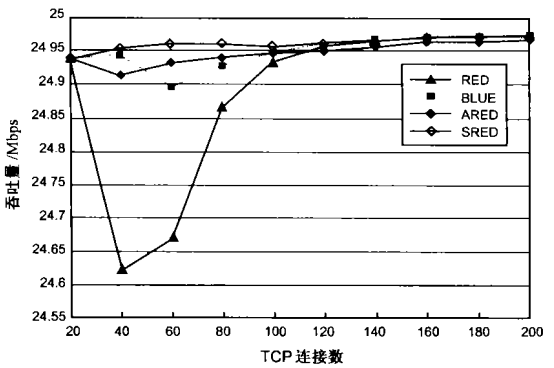

图 7　TCP 连接数对吞吐量的影响

4.5　缓冲区大小对链路利用率的影响

该实验中用到的参数除缓冲区大小及 RED 和 ARED 的参数 $\min_{th}$和 $\max_{th}$外,其他都和表 1 中参数一致. 图 8 显示了不同的缓冲区大小对链路利用率的影响. 缓冲区大小 B 的变化范围为 50 个数据包到 500 个数据包,间隔为 50;不同缓冲区大小情况下,RED 和 ARED 中参数 $\min_{th} = 10 + 20 * i$, $\max_{th} = 30 + 30 * i$,其中 $i = 0, 1, \cdots, 9$. 我们可以明显发现在 RED 算法下链路利用率是随着缓冲区的增加而增加的,并且几乎是呈线性关系. 而对于其他三种算法,当缓冲区大小超过 100 个数据包后,链路利用率基本上比较稳定,特别是 SRED 和 BLUE 都维持了很高的链路利用率.

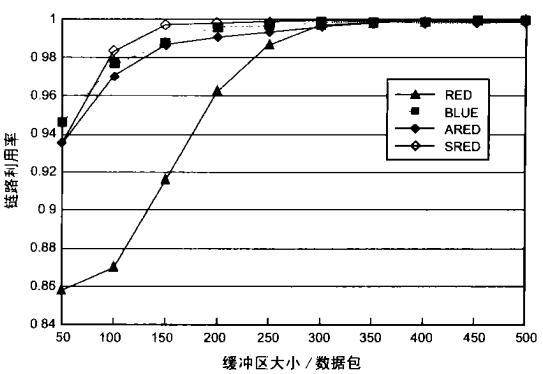

图 8　缓冲区大小对链路利用率的影响

5　总结和今后的工作

本文通过在 ns-2 上进行仿真实验对 4 种 AQM 算法 RED、BLUE、ARED 和 SRED 的性能进行了比较. 从模拟结果中我们可以看出,BLUE、ARED 和 SRED 几乎在所有方面都要优于 RED 算法,而这 3 种 AQM 算法之间则很难说哪种算法性能更好. RED 算法由于不能很好地控制队列长度,使得其队长在出现大量 TCP 连接时波动很大,从而导致出现相对较高的丢包率,影响了链路利用率. BLUE 基于丢包事件和链路空闲事件的拥塞管理,能够较好地估计拥塞程度,从而作出适当的反应,因此其丢弃包的比率相对较低、较稳定. 但 BLUE 算法中由于没有考虑到队长变化的影响,从而导致队长变化较大,这对数据包的端到端延迟有很大影响. ARED 和 SRED 则较好地稳定了队长的变化,这也是这两个算法的主要设计目标,并且 ARED 和 SRED 在其他指标方面的性能也较好.

在我们的实验中所采用的输入模型是 FTP 流,并没有考虑到短时的 web 流,而目前的 Internet 上的流量则是各种各样异质流的混合. 我们今后的工作将进一步考察在异质流的情

况下这些 AQM 算法的性能. 另外,随着多媒体应用的兴起, Internet 上也充斥着越来越多的 UDP 之类的非拥塞自适应的流,如何保证这些流和 TCP 流之间带宽享用的公平性也是 AQM 需要解决的一个主要问题. 我们今后的工作也将考察这些 AQM 算法在确保公平性方面的性能.

参考文献:

[1] J Nagle J. IETF RFC896. Congestion Control in IP/TCP Internetworks [S]. 1984.

[2] S Floyd, K Fall. Promoting the use of end-to-end congestion control in the Internet[J]. IEEE/ACM Transactions on Networking, 1999, 7(4): 458 - 472.

[3] B Braden, D Clark, et al. IETF RFC2309. Recommendations on queue management and congestion avoidance in the Internet[S]. 1998.

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

[5] W Feng, D Kandlur, D Saha, K Shin. BLUE: A new class of active queue management algorithms[R]. US: University of Michigan, CSE-TR-387-99, 1999.

[6] W Feng, D Kandlur, D Saha, K Shin. Techniques for eliminating packet loss in congested TCP/IP networks[R]. US: University of Michigan, CSE-TR-349-97, 1997.

[7] W Feng, D Kandlur, D Saha, K Shin. A self-configuring RED gateway [A]. Proc of IEEE INFOCOM[C]. Amsterdam: Elsevier Press, 1999.

[8] T J Ott, T V Lakshman, L H Wong. SRED: Stabilized RED[A]. Proc. of IEEE INFOCOM[C]. Amsterdam: Elsevier Press, 1999. 1346 - 1355.

[9] S McCanne, S Floyd. ns-LBNL Network Simulator [CP/OL]. http://www-nrg.ee.lbl.gov/ns/, 1996.

作者简介:

吴春明 男,1967 年 6 月生于浙江省萧山县,博士,副教授,现工作于浙江大学计算机学院,主要研究方向为智能机器人体系结构,计算机网络技术,人工智能,图象处理等.

姜 明 男,1974 年生于江苏省如皋市,现为浙江大学计算机学院博士研究生,主要研究方向为计算机网络 QoS、网络拥塞控制、路由器队列调度和队列管理算法等.