REDu:一种新的识别并惩罚非适应流的主动式队列管理算法

黄 磊[1],吴春明[1],姜 明[2],张 栋[1]
(1. 浙江大学人工智能研究所,浙江杭州 310027;2. 杭州电子科技大学计算机学院,浙江杭州 310018)

摘 要: 本文提出一种新的主动式队列管理算法——热度算法(REDu).算法深入挖掘非适应流与适应流本质区别,利用 CHOKe 命中、RED 丢弃等信息预选非适应流,通过热度升降机制计算一种新的部分流状态——热度,以此识别并惩罚非适应流.基于 ns-2 的仿真实验显示,与其他几种主动式队列管理算法相比,REDu 具有更准确的识别并惩罚非适应流的能力,对适应流提供更好的保护,网络的鲁棒性也显著提高.

关键词: 拥塞控制; 主动式队列管理 (AQM); 非适应流; REDu
中图分类号: TP393 **文献标识码:** A **文章编号:** 0372-2112 (2010) 08-1759-04

REDu:A New Active Queue Management Algorithm for Detection and Punishment of Unresponsive Flows

HUANG Lei[1], WU Chun-ming[1], JIANG Ming[2], ZHANG Dong[1]
(1. *AI Institute of Zhejiang University, Hangzhou, Zhejiang* 310027, *China*;
2. *Computer Science Collage of Hangzhou Dianzi University, Hangzhou, Zhejiang* 310018, *China*)

Abstract: This paper proposes a new active queue management algorithm named REDu that excavates in depth the essential differences between non-adaptive and adaptive flows. Taking use of information like CHOKe Hit and RED Drop, this algorithm preselects non-adaptive flows and utilizes a heat increasing and decreasing mechanism to compute "heat", a new kind of partial flow state, achieving the detection and punishment of non-adaptive flows. Simulation results based on ns-2 show that, compared with several other active queue management algorithms, REDu can detect and punish non-adaptive flows more precisely, bring more adequate protection to adaptive flows and improve network robustness significantly.

Key words: congestion control; active queue management (AQM); unresponsive flows; REDu

1 引言

随着因特网的迅速发展,VOD、远程教育、视频会议、可视电话等以流媒体技术为核心的应用逐渐发展为因特网应用的主流.流媒体通常采用不对拥塞进行回应的非 TCP 协议,比如 UDP 协议.研究表明[1],这些非适应流会严重侵占带宽,使得适应流和非适应流之间存在不公平性.通过主动式队列管理算法,惩罚非适应流,保护适应流对网络资源的利用,已成为对多媒体流进行拥塞控制的研究热点.

Rong Pan 等人[2]提出了无状态的 CHOKe 算法,如果到来包和随机选出的包属于同一个流,就被认为属于非适应流而丢弃.P Chhabra 等人[3]提出了部分流状态的 XCHOKe 算法, CHOKe 命中的流被保存到周期性更新的表中,命中计数器的值被用来计算丢弃概率.这些仅基于 CHOKe 命中的算法在流的数目增多时,往往暴露出样本空间不足,识别准确率低的缺点.

本文基于 RED 提出一种新的主动式队列管理算法——REDu(u 代表 update,表示 REDu 是对 RED 的改进,另外 REDu 也取自"热度"的汉字拼音.).算法挖掘适应流和非适应流之间的本质区别,利用 CHOKe 命中、RED 丢弃等信息预选非适应流,通过热度升降机制计算一种新的部分流状态——热度,来对非适应流进行识别及惩罚.仿真实验表明,REDu 算法在保护适应流获得公平性及提高网络的鲁棒性方面都要明显优于 CHOKe 和 XCHOKe 算法.

2 新的 AQM 算法 REDu

2.1 REDu 的基本原理

REDu 的设计目标是识别并惩罚非适应流,保护适应流对网络资源的占用.其基本理论依据是非适应流和适应流之间的本质区别:当网络发生拥塞时,适应流

收稿日期:2009-08-28;修回日期:2010-03-09
基金项目:国家自然科学基金(No.60773182);国家 863 重大项目(No.2008AA01A323;No.2009AA01A334)

由于收到拥塞指示(丢包或 ECN),会自适应的降低其发送速率;而非适应流则不会对此作出反应.据此,REDu 算法设计并利用了两个重要机制——非适应流预选机制和热度升降机制来识别并惩罚非适应流.

降低算法实现复杂度是另一重要目标.降热机制使算法只保存最近一段时间通过本队列的非适应流,降低了算法复杂度.这个设计的理论依据是 Internet 的类似时间局部性原理.由于 Internet 上数据的突发本质[4],各流到达路由器的包也往往是突发的.

2.2 REDu 算法描述

2.2.1 流热度及热度表

流热度(*HotDegree*)是 REDu 算法引入的一种新的流状态,用来量度流属于非适应流的可能性大小,以及该流对网络的恶意程度.非适应流预选机制选出的候选非适应流将被保存到一个热度表(*HotTable*)中,每个表项记录该流的流 ID 及流热度值.

2.2.2 非适应流预选机制

非适应流预选机制的主要功能是在拥塞避免阶段,REDu 能预选一些流作为候选非适应流.预选机制包括 CHOKe 命中预选和 RED 标记丢弃预选,当且仅当 CHOKe 命中或 RED 标记丢弃时,才在热度表中以预设的初始热度值(*InitDegree*)添加新表项.

当队列中流数目较少,非适应流总是比适应流占据更多队列空间,故具有更大的概率被 CHOKe 命中;而当队列中流数目较多时,CHOKe 命中仅以当前队列中的包作为统计样本,不足以用来预选非适应流.但非适应流具有更大的概率被标记为 RED 丢弃,因此 REDu 算法还将每一个被 RED 标记丢弃流也预选进热度表.

2.2.3 热度升降机制

热度升降机制的主要功能是对被预选进热度表的候选非适应流进行筛选,识别出其中的非适应流,并根据其恶意程度进行不同力度的惩罚.热度升降机制规定了热度表中各流热度提升或降低的方法.

热度表中各项的热度值在以下 3 种情况下获得提升:过热命中(*HotHit*),CHOKe 命中(*ChokeHit*)以及 RED 丢弃(*RedDrop*).包到来时,首先就判断所属流是否过热,热度值高于 *HotThreshold* 即为过热命中,该包将被丢弃.而降热则以固定周期发生,即每隔 *CoolInterval* 时间,REDu 以幅值 *CoolDegree* 降低当前热度表中每一表项的热度值.热度值低于 *ColdThreshold* 的表项从表中删除.不同于 XCHOKe 的生存时间(TTL)机制,降热机制保证了热度表各表项的持续有效性,不会在每个生存周期后损失大量有效信息.

2.2.4 丢弃概率计算及参数设置

REDu 根据流热度判断流对网络的恶意程度,从而决定惩罚力度.如果到来包所属的流被判为过热流(过热命中),算法计算概率 P_{Hot} 对包进行丢弃:

$$P_{Hot} = 2^{HotDegree} \times [\alpha * P_{Table} + (1-\alpha) * P_{Red}]$$

其中,*HotDegree* 为该到来包所属流的热度值;$\alpha(0 \leqslant \alpha \leqslant 1)$为加权因子,用以权衡流热度和队列长度信息在丢弃概率计算中所占的比重.

P_{Table}是由流热度计算得到的概率值,其计算公式如下所示:

$$P_{Table} = \frac{HotDegree - InitDegree}{MaxDegree - InitDegree}$$

其中 *HotDegree* 为流热度值;*InitDegree* 为流被预选加入表中的初始热度值;*MaxDegree* 为预设的流热度提升的最大值.

InitDegree、*HotThreshold* 及 *HitInc* 决定了 REDu 识别非适应流的敏感度;而 *MaxDegree* 则决定了对过热流的惩罚力度.这些值的设定取决于队列的实际网络环境.

P_{Red}是由 RED 算法根据平均队列长度计算的丢弃概率,RED 标记丢弃(*RedDrop*)中也使用这个概率,计算公式[5]如下所示:

$$P_b = \max_p \times \frac{\text{avg}Q - Th_{\min}}{Th_{\max} - Th_{\min}}$$

$$P_{Red} = \frac{P_b}{1 - count \times P_b}$$

其中 $\max_p$ 是 RED 算法预设的最大丢弃概率值;avg$Q$ 是 RED 中计算的平均队列长度;*count* 是从上次 RED 丢包到现在为止进入队列的包的数目.

REDu 算法使用加权因子 α 权衡流热度和队列长度信息在丢弃概率计算中所占比例.为了获得不同的惩罚力度,α 的值可以作不同的设置.此计算方法也使丢弃概率因热度值和平均队列长度的增减引起的变化趋于平缓,一定程度上保证了网络鲁棒性.

3 仿真实验及结果分析

3.1 实验场景设置

为了分析 REDu 的性能,我们设计了两个场景,并在 ns-2.33[6]下进行仿真.两个场景均采用哑铃状拓扑结构,如图 1 所示:中间两个路由器间为瓶颈链路,带宽 1Mb,链路延时 1ms,分别使用 CHOKe、XCHOKe 和 REDu 进行仿真;其余链路带宽均为 10Mb,延时 1ms,队列为 DropTail;数据包大小均为 1kB,仿真持续时间均为 100s;两个场

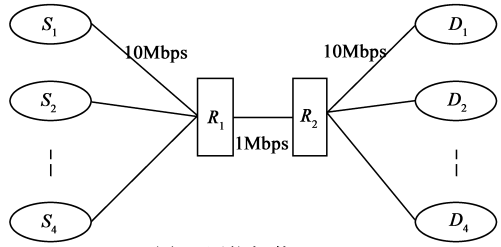

图1 网络拓扑

景的其他设置分别如下:

场景一:共设 11 个发送节点,其中 5 个节点发送稳定 *TCP* 流,速率分别为 0.2、0.4、0.6、0.8 和 1.0*Mbps*;两个节点发送稳定 *UDP* 流,速率分别为 0.5 和 1.0*Mbps*;4 个节点发震荡 *UDP* 流,速率均为 1.0*Mbps*;震荡 *UDP* 流从第 10 秒开始发送,第 30 秒停止;第 50 秒又重新开始,第 70 秒停止;

场景二:固定地设置 10 个节点发送稳定 *TCP* 流,理论速率 0.5*Mbps*;*UDP* 流发送节点的数目由 1 个递增到 10 个,理论速率均为 1.0*Mbps*.

衡量拥塞控制算法好坏的三个关键量度[7]是:*i*).公平性;*ii*).资源利用率;*iii*).计算复杂度.3.2 – 3.5 将多个方面对算法性能进行衡量.

3.2　不同队列的各流总吞吐量对比

图 2 统计了瓶颈链路上各流的实际总吞吐量.可以明显看到,REDu 队列中 TCP 流的吞吐量总和远高于 CHOKe 和 XCHOKe.另外,REDu 在 5 个不同速率的 TCP 流之间也实现了更好的公平性.

与 XCHOKe 相比,REDu 算法不仅两个稳定 UDP 流受到限制,4 个震荡 UDP 流的总吞吐量也比 CHOKe 降低了约 50%.这说明在大量 UDP 流同时存在的情况下,REDu 对非适应流的识别和惩罚的表现优于 XCHOKe.

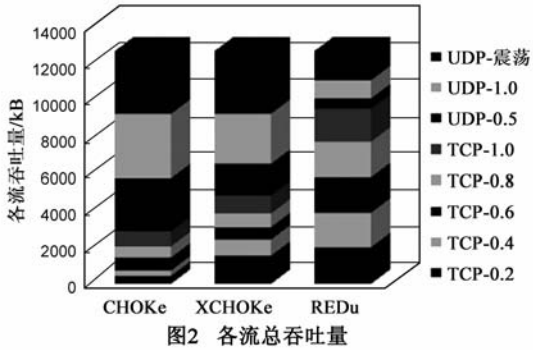

图2　各流总吞吐量

3.3　不同队列 TCP 流总吞吐率对比

如图 3、图 4、图 5 所示,对比 TCP 流总吞吐率随时间变化的曲线图,看到以下两个现象:

(1)在震荡 UDP 流开始发包后,REDu 算法的 TCP 流吞吐率所受影响最小,平缓下降到 600kbps 左右,远高于 CHOKe 的 20kbps 以下和 XCHOKe 的 100kbps 以下;

这充分体现了 REDu 算法识别多条非适应流和应对突发流的能力.在 4 个 UDP 震荡流启动时,通过 CHOKe 命中预选和 RED 丢弃预选两种机制,REDu 快速捕捉到这 4 个 UDP 流,并开始丢包.

(2)在 4 个震荡 UDP 流停止发包期间,REDu 和 XCHOKe 的 TCP 流吞吐率分别达到了 900kbps 和 700kbps,明显高于 CHOKe 的 450kbps.REDu 和 XCHOKe 通过保存部分流状态,有效识别并惩罚了两个 UDP 流,提高了 TCP 流的吞吐率.

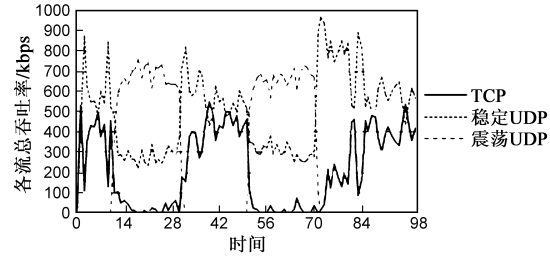

图3　CHOKe:TCP流总吞吐率随时间变化

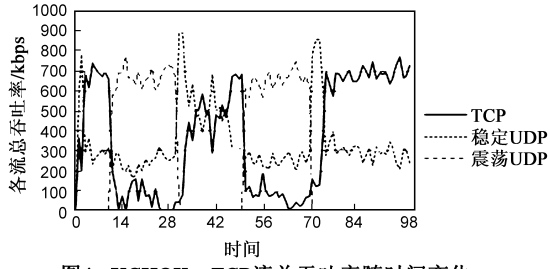

图4　XCHOKe:TCP流总吞吐率随时间变化

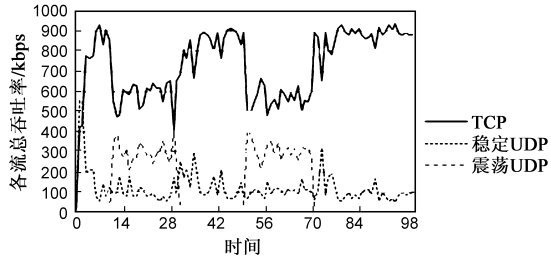

图5　REDu:TCP流总吞吐率随时间变化

3.4　实际队列长度

队列的实际长度能够有效反映网络的稳定性和鲁棒性.图 6、图 7 及图 8 分别是 CHOKe、XCHOKe 和 REDu 算法实际队列长度随时间变化的曲线图,单位是字节.

CHOKe 算法在最初流启动时以及震荡 UDP 流开始发包的时刻,队列长度急剧增加到 60kB 左右,明显高于 XCHOKe 和 REDu 的 38kB.当突发状况发生时,XCHOKe 和 REDu 都能够通过保存突发流的状态,给予有效的惩罚.

在震荡 UDP 发包和停止发包期间,REDu 的队列长度则一直稳定保持在 10kB 左右.这是因为震荡 UDP 流突然停止发包时,队列中只剩下两个稳定 UDP 流和 5 个稳定 TCP 流,三种队列对 UDP 流的包都大量丢弃;但 CHOKe 和 XCHOKe 的 TCP 流速率比 REDu 低得多,而且增速缓慢,这导致了一段时间内队列长度的下降.

此外,由于 XCHOKe 中丢弃概率计算如下:

$$P^{*} = \min(1, P_{Red} \times 2^{n})$$

其中 n 为该流命中计数器的值.因为 4 个震荡 UDP 流停止发包时,P_{Red}的数值仍然较高,这直接导致队列对各流的惩罚过重,队列长度急剧下降.

而 REDu 则引入加权因子 α,在队列长度下降时,虽然 P_{Red}数值较高,但由热度值决定的丢弃概率也占了

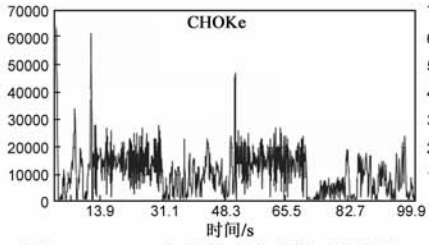

图6 CHOKe:实际队列长度随时间变化

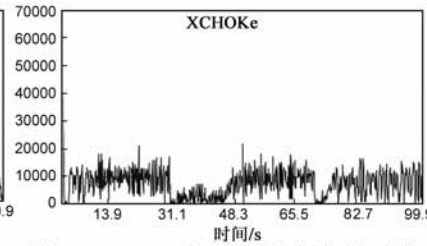

图7 XCHOKe:实际队列长度随时间变化

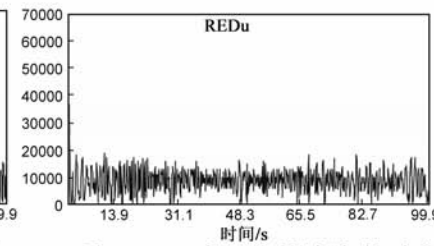

图8 REDu:实际队列长度随时间变化

比重,因此降低了错误的丢弃,使队列长度保持稳定.

3.5 不同队列 TCP 流总吞吐量随 UDP 流数目的变化

当非适应流的数目逐渐增多时,AQM 对适应流的保护能力是另一个需要研究的方面[8].

如图 9 所示,当 UDP 数目逐渐增多时,CHOKe 和 XCHOKe 算法的 TCP 吞吐量急剧降低,并分别在 UDP 数目达到 4 个和 6 个时降到 0;而 REDu 的 TCP 吞吐量则一直呈线性逐步降低.

CHOKe 和 XCHOKe 仅基于 CHOKe 命中,在非适应流增加到一定数目时,效果急剧下降;另外,TTL 机制也使 XCHOKe 周期性损失大量有效的历史识别信息.

对于 REDu 来说,即使非适应流数目较多,预选机制也能使它快速感知非适应流;配合热度升降机制与合理的丢弃概率计算方法,REDu 很好的保护了相对脆弱的适应流.并且降热机制和基于 hash 表的数据结构也保证了较低的实现复杂度.

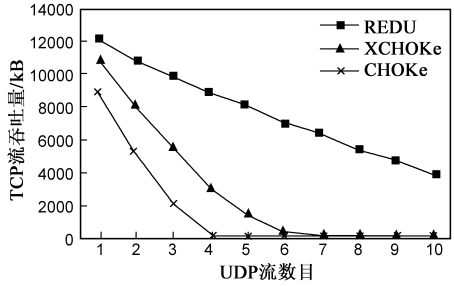

图9 TCP流总吞吐量随UDP数目的变化

4 结论

本文提出一种主动式队列管理算法——REDu,通过 CHOKe 命中和 RED 丢弃两种预选机制选出候选非适应流,利用热度升降机制筛选非适应流,综合队列长度和流热度计算丢弃概率对其进行惩罚.仿真实验表明,与 CHOKe 和 XCHOKe 相比,REDu 更加有效稳定地保护了适应流对网络资源占用,提高了网络的鲁棒性.

参考文献:

[1] V Ramaswamy, L Cuellar, S Eidenbenz, N Hengartner. Preventing bandwidth abuse at the router through sending rate estimate-based queue management schemes[A]. Proc IEEE ICC'07[C]. Glasgow: IEEE Press, 2007. 569 – 574.

[2] R Pan, B Prabhakar, K Psounis. CHOKe: A stateless active queue management scheme for approximating fair bandwidth allocation[A]. Proc IEEE INFOCOM '00[C]. Tel Aviv: IEEE Computer Society Press, 2000. 942 – 951.

[3] P Chhabra, A John, H Saran, R Shorey. Controlling malicious sources at internet gateways[A]. IEEE Int Conf. Communication[C]. Anchorage: IEEE Press, 2003. 1636 – 1640.

[4] P Abry, P Flandrin, M Taqqu, D Veitch. Wavelets for the analysis, estimation, and synthesis of scaling data[A]. Self-Similar Network Traffic Analysis and Performance Evaluation[C]. New York: Wiley, 2000. 39 – 88.

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

[6] The Network Simulator NS2 [OL]. http://www.isi.edu/nsnam/ns/, 2009 – 07 – 17.

[7] V Ramaswamy, et al. Light-weight control of mon-responsive traffic with low buffer requirements[A]. Proc of IFIP Networking'07[C]. Atlanta: Springer Berlin, 2007. 855 – 866.

[8] 吴春明,姜明,朱淼良.几种主动式队列管理算法的比较研究[J].电子学报,2004,32(3):429 – 434.
Wu Chun-ming, Jiang Ming, Zhu Miao-liang. A research on some active queue management algorithms[J]. Acta Electronica Sinica, 2004, 32(3): 429 – 434. (in Chinese)

作者简介:

黄 磊 男,1986 年生于江苏启东,硕士研究生,主要研究方向是可重构网络技术.
E-mail: suns.huanglei@gmail.com

吴春明 男,1967 年生于浙江萧山.博士,浙江大学计算机学院教授、博士生导师.主要从事网络服务质量、可重构网络技术、网络虚拟化等方向的科学研究工作.
E-mail: wuchunming@zju.edu.cn