随机回退的TCP拥塞控制算法

姜文刚[1,2],孙金生[1],王执铨[1]
(1.南京理工大学自动化学院,江苏南京 210094; 2.江苏科技大学电子信息学院,江苏镇江 212003)

摘 要: 面对目前大量使用弃尾队列管理的网络中间节点,提出了一种随机回退的TCP拥塞控制算法TCP Njust,拥塞时在一定范围内随机减少拥塞窗口,拥塞避免时其拥塞窗口在一定范围内随机增加,以减缓TCP全局同步.该算法在Newreno基础上增加了一个随机函数,仿真表明该算法的性能优于Newreno,在网络突发数据流较多时,其性能优于Vegas,适合在Internet上使用.

关键词: 网络拥塞控制; 全局同步; 随机回退; 拥塞窗口; 弃尾
中图分类号: TP393 **文献标识码:** A **文章编号:** 0372-2112 (2011) 07-1689-04

A Random Back-Off TCP Congestion Control Algorithm

JIANG Wen-gang[1,2], SUN Jin-sheng[1], WANG Zhi-quan[1]
(1. *School of Automation, Nanjing University of Science and Technology, Nanjing, Jiangsu* 210094, *China*;
2. *School of Electronic and Information, Jiangsu University of Science and Technology, Zhenjiang, Jiangsu* 212003, *China*)

Abstract: Now a large number of intermediate nodes face a fact of using drop tail queue management, therefore a random back-off TCP congestion control algorithm named TCP Njust is proposed. When congestion occurs, randomly reduce the congestion window within a certain range, but when congestion avoid, congestion window is randomly increased within certain limits to reduce TCP global synchronization. This algorithm requires a random function on the base of Newreno. Simulation results show that this algorithm performs better than Newreno, and also better than Vegas when a large number of burst data stream occurs in the network, and it's suitable for use in the internet.

Key words: network congestion control; global synchronization; random back-off; congestion window; drop tail

1 引言

为增强TCP拥塞控制机制,Jacobson提出和式增加积式减少(AIMD)的"慢启动"、"拥塞避免"算法, Reno中增加了"快速重传"、"快速恢复", NewReno改进了"快速恢复".总体上TCP通过AIMD,对这4个算法设置不同的参数来实现不同TCP拥塞控制[1].

中间节点中最广泛使用的队列管理策略是"弃尾"(Drop Tail),即随着缓冲区的溢出而丢包,是一种被动队列管理机制."弃尾"的缺陷包括全局同步等.拥塞时,多个TCP发送端检测到丢包时,均减少发送速率,会造成中间结点从拥塞进入低利用率状态.而探测到空闲时,各发送端又增大发送速率,中间结点又发生拥塞,如此往复,产生了全局同步,导致链路带宽的利用率下降.主动队列管理(AQM)虽然可以有效地解决"全局同步"问题,但存在参数设置敏感,响应相对滞后等缺陷[2],没有在网络上广泛使用,文献[3]指出AQM的网络传输效率不如改进的弃尾队列管理.

对于采用弃尾的中间结点,如果TCP采用同样的方式改变拥塞窗口(Cwnd),全局同步不可避免,因此提出了一种TCP拥塞控制算法:TCP Njust,使TCP的Cwnd的增加和减少具有一定的随机性.拥塞时,有的Njust发送速度减少的多,有的减少的少,尽量避免发送速度减少过多导致效率下降;在拥塞避免阶段,各个Njust数据流Cwnd的增加也是随机的,防止发送速度过快地增长,延迟下次拥塞的时间,减少拥塞窗口波动,提高网络的传输效率.由于每个Njust Cwnd的增加和减少都是随机的,因此从总体上来说,各个Njust竞争是公平的.

2 现有的TCP拥塞控制算法

目前的TCP拥塞控制算法主要分为无带宽探测和有带宽探测两大类.无带宽探测的TCP是目前在Internet上应用最广的,包括Tahoe、Reno、Newreno、Sack、CUBIC、HSTCP、改进慢启动算法[4]等等,以上每种协议均采用相同Cwnd增加和减少机制,拥塞时减半减少Cwnd,增加时按同样的策略,造成严重的"全局同步".

收稿日期:2010-09-29;修回日期:2011-03-11
基金项目:国家自然科学基金(No.60974129);江苏省自然科学基金(No.BK2009388);江苏省青蓝工程资助项目

有带宽探测的 TCP 主要包括 Vegas[5]、Westwood[6]、Rab[7]等等,通过先前的信息预测未来网络的带宽,来调节 Cwnd[8],在网络数据流平稳时,能达到较好的性能.文献[9,10]研究表明,Internet 的数据流具有突发性,预测难度大.Vegas 和未使用 Vegas 的 TCP 竞争带宽时存在严重的不公平[11].基于历史信息预测的 TCP 拥塞控制算法有一定的滞后,在网络流量波动大时,性能下降较多.随着大量嵌入式联网设备的出现,特别是物联网的发展,由于运算能力的限制,过于复杂的 TCP 拥塞控制算法较难实现,因此需要一种运算量小、改进的 TCP 拥塞控制算法,来提高网络的传输效率.

3 TCP Njust

3.1 TCP Njust 拥塞控制算法

Njust 拥塞控制算法主要改动 Newreno 协议的两个部分:发生拥塞时的 Cwnd 回退减少策略和拥塞避免时的 Cwnd 增加策略.其余部分和 Newreno 一样.

拥塞时 Njust 按如下方法减少 Cwnd:

$$Cwnd = Cwnd - Cwnd \times R(c,d) \tag{1}$$

$R(c,d)$为 c 到 d 之间平均分布的随机数,其中 $0 < c < d \leqslant 1$.

如果 Cwnd 的增加还是和 Newreno 一样每个往返时间(RTT)增加 1,则增加过快,缩短了到下次拥塞发生的时间,因此在拥塞避免阶段 Njust 按如下方法增加 Cwnd:

每收到一个新数据包 ACK 时,

$$Cwnd = Cwnd + \frac{1}{Cwnd} \times R(e,f) \tag{2}$$

$R(e,f)$为e到 f 之间平均分布的随机数,其中 $0 < e < f \leqslant 1$.

拥塞时 Njust 按照式(1)减少 Cwnd,避免瓶颈链路的队列减少过多.拥塞避免阶段,按式(2)随机增加 Cwnd,减缓"全局同步".Njust 只是比 Newreno 多了一个随机数函数,并没有增加过多的计算量.

3.2 TCP Njust 性能分析

文献[12]忽略了慢启动,得到采用 AIMD 策略的 TCP 平均发送速度如下:

$$T = \frac{\sqrt{2-b}\sqrt{a}}{\sqrt{2b}RTT\sqrt{p}} \tag{3}$$

在式(3)中 a 表示拥塞避免时 Cwnd 增加的量,b 表示拥塞时 Cwnd 减少的系数,RTT 为往返时间,p 为丢包率.对于线性增加减半减少的 TCP 协议,$a = 1, b = 0.5$.则 TCP 平均发送速度为:

$$T = \frac{\sqrt{1.5}}{RTT\sqrt{p}} \tag{4}$$

如果丢包率相同,a,b 满足下面式(5),则 TCP 的发送速度是一样的.

$$a = \frac{3b}{(2-b)} \tag{5}$$

式(1)中的 $R(c,d)$和式(2)中的 $R(e,f)$均为平均分布的随机数,平均后 $R(c,d)$近似为$(c+d)/2$,$R(e,f)$近似为$(e+f)/2$,即 $a = (e+f)/2, b = (c+d)/2$,所以 Njust 的平均发送速度 T_{Njust}为:

$$T_{Njust} = \frac{\sqrt{2-(c+d)/2}\sqrt{(e+f)/2}}{\sqrt{2(c+d)/2}RTT\sqrt{p_N}}$$
$$\Rightarrow T_{Njust} = \frac{\sqrt{4-(c+d)}\sqrt{(e+f)}}{\sqrt{2}\sqrt{(c+d)}RTT\sqrt{p_N}} \tag{6}$$

根据式(5),当 c、d、e、f 满足下面式(7)条件时,Njust 的发送速度应该和 Newreno 等协议是一样的.

$$(e+f) = \frac{6(c+d)}{4-(c+d)} \tag{7}$$

由于 Njust 采用随机回退和增加的策略,减缓了"全局同步",减少了数据包的丢失,所以在满足式(7)条件下,式(6)的丢包率 p_N 比式(3)的 p 要小,这样 Njust 的发送速度就比 Newreno 要快.式(3)没有考虑慢启动,由于 Njust 没有改变慢启动、快速恢复和快速重传算法,所以在这三个阶段的发送速度和 Newreno 是一样的.

3.3 TCP Njust 公平性分析

Njust 协议之间的公平性可以用式(6)来比较,在同样的网络环境下,如果协议参数一样,丢包率相同,则它们的发送速度是一样的.如果 RTT 不一样,则存在 RTT 不公平性:

$$\frac{T_{Njust1}}{T_{Njust2}} = \frac{\frac{\sqrt{4-(c+d)}\sqrt{(e+f)}}{\sqrt{2}\sqrt{(c+d)}RTT1\sqrt{p_N}}}{\frac{\sqrt{4-(c+d)}\sqrt{(e+f)}}{\sqrt{2}\sqrt{(c+d)}RTT2\sqrt{p_N}}} \approx \frac{RTT2}{RTT1} \tag{8}$$

式(8)成立是在丢包率相同,忽略慢启动也就是不出现数据包超时的条件下成立的.

Njust 和 Newreno 同在一个网络传输数据时,不考虑慢启动,如果 Njust 满足式(7),且 RTT 相同,则两者的速率之比为:

$$\frac{T_{Njust}}{T} = \frac{\sqrt{p_N}}{\sqrt{p}} \tag{9}$$

在同一个网络,式(9)中 p_N 一般不会大于 p,所以 Njust 的发送速度不会低于 Newreno.

Njust 在和有带宽探测的 Vegas 等协议竞争网络带宽时,由于 Vegas 本身的原因,Vegas 发送速度过低.

4 仿真分析

在 NS2 下构建图 1 仿真拓扑,$R0$ 和 $R1$ 之间为瓶颈链路,采用弃尾队列管理,缓存为 30 packets;Si 均为持久性 FTP 源,向 Di 发送数据;Si 与 $R0$、Di 与 $R1$ 之间

的链路容量均为 *20*Mbps,延时 15ms;数据包均为 1040Byte(包头 40Byte).接收窗口设置足够大,使得 TCP 发送仅受 Cwnd 控制.

4.1 Njust 与 Newreno 性能比较

Floyd 在文献[13]中提出了"TCP 兼容协议"的概念,即该兼容协议能够响应网络拥塞而降低发送速度,在相同的条件(RTT,丢包率等)下,不占用其他标准 TCP 的带宽.在兼容的条件下,设置 Njust 参数为:$c=0.2$, $d=0.55$, $e=0.38$, $f=1.0$ 来验证其性能.在图 1 中发送节点分别采用 Njust 和 Newreno 协议,仿真在不同发送节点个数时,瓶颈链路 $R0$ 与 $R1$ 采用不同速度时,传输的有效数据包数目.Njust 的 4 个参数设置基本满足式(7),其发送速度为:

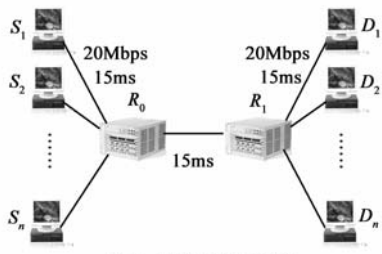

图1　网络拓扑结构

$$T_{Njust}=\frac{\sqrt{1.495}}{RTT\sqrt{p_N}} \tag{10}$$

理论上如果丢包率相同,则 Njust 发送速度略微小于 Newreno 式(4)得到的发送速度,但是由于 Njust 减缓了"全局同步",仿真结果显示 Njust 的发送速度要快于 Newreno.仿真时间为 100s,结果见图 2.图 2 中纵轴表示在同样的发送端数目和瓶颈链路速度情况下,Njust 和 Newreno 发送有效数据包的比值.在瓶颈链路速度低或发送节点数多时,发送端的 Cwnd 一直处于较小状态,随机回退并不能拉开发送端 Cwnd 的差距,也就不能有效减缓"全局同步",另外 Njust 并没有改变 Newreno 的慢启动算法,所以两者发送的有效数据包是一样的.但随着瓶颈链路速度增加或发送节点数减少,各发送端的 Cwnd 较大,随机回退能拉开发送端 Cwnd 的差距,所以能减缓"全局同步",Njust 发送的有效数据包明显超过 Newreno,提高了瓶颈链路的利用率.

为了尽量避免慢启动阶段,选择在瓶颈链路速度较快,为 12Mbps 时,测试只有一个发送节点时,Njust 和 Newreno 发送的有效数据包,分别是:119682 packets 和 123487 packets,Njust 单个节点的发送速度低于 Newreno,这也基本符合式(10)的结果.在 4 个发送节点时观察链路 $R0-R1$ 缓存队列的变化情况.图 4 中,采用 Newreno 协议时,链路队列为长度 0 的次数多于图 3 中采用 Njust 算法时的次数.瓶颈队列长度为 0 意味着链路空闲,效率下降.这种状况就是"全局同步"造成的,可见 Njust 能有效减缓"全局同步".

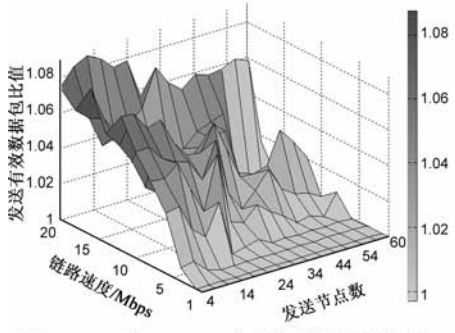

图2　Njust与Newreno传输有效数据包比较

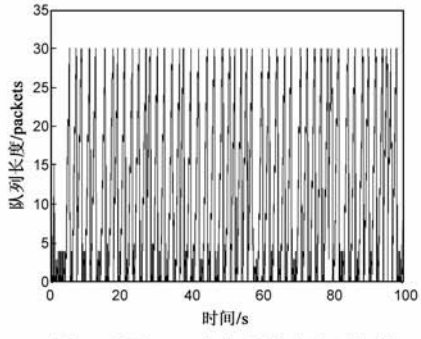

图3　采用Njust时,瓶颈链路队列变化

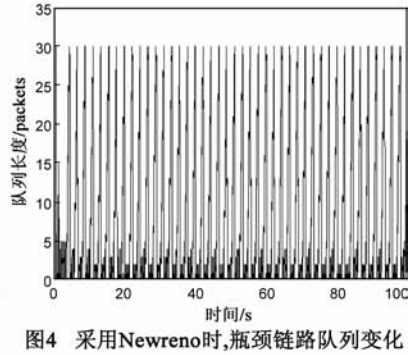

图4　采用Newreno时,瓶颈链路队列变化

4.2 Njust 与 Vegas 性能比较

在图 1 网络中加入 2 个速度一样随机突发的数据流,在 NS2 中用 Pareto ON/OFF 对象实现,突发和空闲时间均为 100ms,数据包为 1000Byte,Pareto 的分布成形参数为 1.5,观察在突发数据流不同的发送速度下各算法的性能.TCP 发送节点为 8 个,分别均采用 Njust、Newreno 或 Vegas 算法,其它参数不变如图 1,统计瓶颈链路发送有效数据包的个数(不包括突发流的数据包).仿真 100s,在图 5 中可以看到,突发数据流速度为 2Mbps 时,突变不剧烈,Vegas 发送的有效数据包超出 Njust 6.2%,但随着突发数据流速度增加,3 个算法的发送有效数据包不同程度地减少,而 Vegas 下降更多,突发数据流速度为 8Mbps 时,Vegas 发送的有效数据包只有 Njust 的 85.1%.所以基于带宽探测的 Vegas 在网络传输不平稳时,性能下降较多.

4.3 混合通讯时的性能

在图 1 网络中,发送节点数为 2,分别是 Njust 和 Newreno,统计 100s 内 2 个节点发送的有效数据包,Njust 为 67488 packets,Newreno 为 61849 packets.如果 2 个发送节点都采用 Newreno,平均每个节点发送的有效数据包为 62174 packets,下降 0.5%,所以 Njust 采用这组参数:$c=0.2$, $d=0.55$, $e=0.38$, $f=1.0$,对 Newreno 基本上是兼容的.Njust 多发送的有效数据包是随机回退的作用.两种算法的 Cwnd 变化如图 6 所示,如果将 Njust 应用于 Internet 上与现有的 TCP 协议比不会处于弱势.将上述 Newreno 协议改为 Vegas 时,统计 100s 内 2 个节点发送的有效数据包,Njust 为 96139 packets,Vegas 为 34662 packets,Vegas 与 Njust 竞争完全处于劣势.仿真结果表

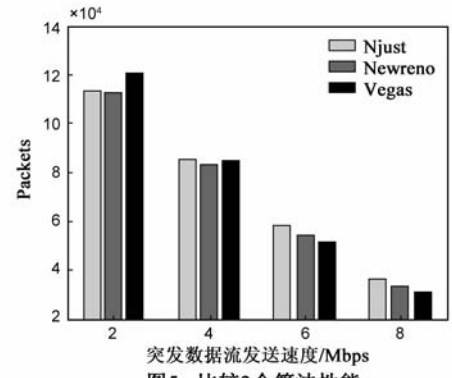

图5 比较3个算法性能

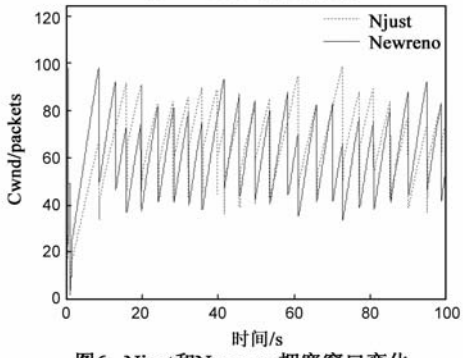

图6 Njust和Newreno拥塞窗口变化

明 Njust 能够有效减缓“全局同步”,提高网络传输效率,可以在目前多种 TCP 协议共存的 Internet 上使用.

5 结论

Njust 拥塞控制算法,只是在 Newreno 的基础上增加了一个随机数,并没有过多的计算量,能够降低全局同步,减少丢包,提高网络的性能.仿真是在满足式(7)基础上的,因此可以进一步研究改变四个参数时的 TCP Njust 性能,该算法还可以推广到其它 TCP 协议,来提高网络传输效率.

参考文献

[1] Kevin F, Floyd S. Simulation-based comparisons of tahoe, teno, and SACK TCP[J]. ACM Computer Communication Review, 1996, 26(3): 5 – 21.

[2] Chen C K, Hung Y C, et al. Design of robust active queue management controllers for a class of TCP communication networks[J]. Information Science, 2007, 177(19): 4059 – 4071.

[3] Zheng Changyong, Dai Yuehua, Chen Junning. Is current active queue management really necessary [A]. First International Workshop on Education Technology and Computer Science [C]. Wuhan, 2009. 538 – 541.

[4] CHEN Zhigang, DENG Xiaoheng, et al. A new parameter-config based on slow-start mechanism[J]. Journal of Communication and Computer, 2005, 12(5): 56 – 62.

[5] SING J, BEN S. Improving congestion window growth in large bandwidth delay product networks[A]. 15th IEEE International Conference on Network[C]. Adelaide, 2007. 382 – 387.

[6] Tekala M. TCP westwood with limited congestion window[A]. International Conference on Advanced Computer Control[C]. Singapore, 2009. 687 – 692.

[7] TANG Xu-hong, LIU Zheng-lan, ZHU Miao-liang. TCP-Rab: a receiver advertisement based TCP protocol[J]. Zhejiang Univ SCI 2004, 5(11): 1352 – 1360.

[8] 蒋翊,吴春明,姜明.一种拥塞感知的 TFRC 协议慢启动算法[J].电子学报,2009,37(5):1025 – 1029.
Jiang Yi, Wu Chun-ming, Jiang Ming. A congestion aware Slow-Start algorithm for TFRC protocol[J]. Acta Electronica Sinica, 2009, 37(5): 1025 – 1029. (in Chinese)

[9] Crovella M, Bestavros A. Self-similarity in world wide web traffic: Evidence and possible causes[J]. IEEE/ACM Transactions on Networking, 1997, 5(6): 835 – 846.

[10] Wen Y, Zhu G X. A compound prediction of self-similar traffic with heavy tailness[A]. Proceedings of ICSCA2006[C]. China, 2006. 846 – 850.

[11] Jeonghoon M, Richard J L. Analysis and comparison of TCP reno and vegas[A]. Proceedings of IEEE INFOCOM'99[C]. New York, 1999. 1556 – 1563.

[12] FLOYD S, HANDLEY M, PADHYE J. A comparison of equation-based and AIMD congestion control[R]. ACIRI, May 12, 2000.

[13] Floyd S, Handley M, Padhye J, et al. Equation-based congestion control for unicast applications[J]. ACM Computer Communication Review, 2000, 30(4): 43 – 56.

作者简介

姜文刚 男,博士生,副教授,研究方向为网络控制、伺服控制.
E-mail: a_1_2_3@163.com

孙金生 男,博导,教授,研究方向为网络拥塞控制、容错控制.