RED算法优化及仿真

曹志波
(河南理工大学 电气工程与自动化学院 复杂网络研究室,河南 焦作 454003)

摘 要:通过优化随机早期检测算法来降低网络的丢包率,提高吞吐量和降低延时。用网络仿真软件进行网络仿真:分析随机早期检测算法,找出随机早期检测算法在避免网络拥塞时存在的缺点;针对随机早期检测算法存在的缺点进行优化;将优化的随机早期检测算法在网络仿真软件上实现,在丢包率、吞吐量和延时上与随机早期检测算法进行比较,得到最终的实验结果。特色在于合理设置最大丢包率来控制路由器中的瞬时队列长度,进而实现网络资源的优化。仿真结果表明,相对于随机早期检测算法,改进后的算法能更好地降低网络的丢包率、提高吞吐量和降低延时。

关键词:随机早期检测算法;优化的随机早期检测算法;丢包率;吞吐量;延时
中图分类号:TP301.6 **文献标识码**:A **文章编号**:1673-629X(2010)08-0188-04

Simulation and Optimization of RED Algorithm

CAO Zhi-bo
(Complex Networks Lab. of College of Electrical Engineering & Automation,
Henan Polytechnic University, Jiaozuo 454003, China)

Abstract: Improve troughput and reduce packet loss rate and delay by optimizing RED algorithm. Simulation based on NS-2 is conducted: Firstly, analyze RED algorithm to find drawbacks of it; Secondly, optimize theoretically the drawbacks of RED algorithm and then get IM_RED algorithm; Finally, carry out IM_RED algorithm on NS-2, get results by comparing with RED algorithm on packet loss rate, troughput and delay. Reasonably setting maximum packet loss rate on router is main characteristic of text and acquire better network condition. The results show that IM_RED algorithm indeed has more advantages than RED algorithm in packet loss rate, troughput and delay.

Key words: RED algorithm; IM_RED algorithm; packet loss rate; troughput; delay

0 引 言

众所周知,因特网对数据的传输采用的是分组转发,并为每个接入因特网的主机提供尽力而为的服务[1],是不可靠的传输。随着加入的主机的增多,网络就会发生拥塞,传输效率变得低下,这就需要好的调度算法[2~5]来解决。如何设计好的调度算法来预防和避免网络拥塞[6~8]也就成了当前计算机网络的热门研究课题。

避免拥塞的关键是提前地预知拥塞和在合适的地方安装调度算法,路由器是分组转发的枢纽,所以也就成为安装调度算法的最佳的选择。文中针对随机早期检测算法RED[9~11]进行了优化,得出IM_RED算法,并在延时、吞吐量和丢包率方面比较了两种算法的优劣。最终验证在同样复杂的网络环境下,IM_RED算法取得了相对低的丢包率[12]、高的吞吐量[13]和较低的延时。然后又在改变udp数据流带宽的情况下验证了IM_RED算法的有效性。

1 RED随机早期检测算法及优化

1.1 RED算法

随机早期检测算法RED(Random Early Detection Algorithm)是由Sally Floyd和Van Jacobson提出的,用于路由器的拥塞避免控制。路由器工作在TCP/IP协议的网络层,它的主要功能就是为分组(IP数据包)选择合适的路径。采用RED算法的路由器,对于到达的但是超出路由器缓冲区的数据包,会在数据包上打上标记来通知发送端,让发送端减小发送窗口来避免拥塞的发生。因该算法对数据包的标记具有一定的概率性,故可以早期地避免拥塞的发生而命名。

收稿日期:2009-12-08;**修回日期**:2010-03-24
基金项目:河南省教育创新人才支持计划项目(2009HASTIT021);河南省基础与前沿技术研究计划项目(072300460050)
作者简介:曹志波(1985-),男,河北邯郸人,硕士,从事计算机网络研究。

RED算法通过定时地读路由器缓冲区的数据包队列的长度,利用指数加权移动平均算法(EWMA)计算出数据包的平均队列长度,如式(1):

$$Q_{t+1} = (1-\omega) \times Q_t + \omega \times q_{t+1} \tag{1}$$

其中 Q_t 为 t 时刻的平均队列长度,Q_{t+1} 为 $t+1$ 时刻的平均队列长度,q_{t+1} 为 $t+1$ 时刻的瞬时队列长度,ω 为权值取值范围[0,1]。

然后再利用算法内部的数据变量 $\max_p$ 与读缓冲区得到的平均队列进行运算得到丢弃概率 p,如式(2):

$$p = \begin{cases} 0 & 0 \leqslant Q < \min_{th} \\ \dfrac{Q - \min_{th}}{\max_{th} - \min_{th}} \times \max_p & \min_{th} \leqslant Q < \max_{th} \\ 1 & \max_{th} \leqslant Q < q_{\lim} \end{cases} \tag{2}$$

其中 p 为丢弃概率,Q 为数据包的平均队列长度,$q_{\lim}$ 为缓冲区容许的最大队列长度,$\min_{th}$ 为数据包长度的最小门限阈值,$\max_{th}$ 为数据包长度的最大门限阈值,$\max_p$ 为最大丢包概率。

另外第一次丢包概率和下一次丢包概率之间的分组数不易太长,因此随着两组间隔标记的分组之间的分组数增多,丢包概率应该适当地变化,如式(3):

$$p = \frac{p}{1 - \text{count} \times p} \tag{3}$$

式中的 count 为两次丢包之间未被标记的分组数目。

上面的分析图如图 1 所示。

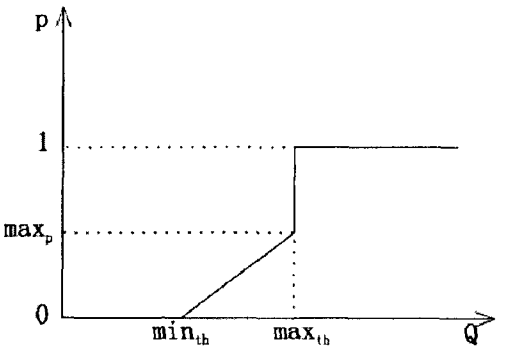

图 1　丢包概率与分组队列的关系

1.2　RED 算法的缺点

RED算法解决了 DropTail 丢尾算法的全局等待的问题,但是自身仍存在许多问题,其中包括:

(1)参数的配置问题:在不同的网络负载下,要使网络在丢包率、时间延迟和吞吐量达到良好的状态,需配置合适的参数。

(2)公平性问题:RED 算法没有能够有效地解决传输层中 tcp 流和 udp 流的公平竞争网络资源的问题。因为,tcp 流是一种可靠的传输流,在传输前会跟目的端先进行三步握手,然后再传输数据流,目的端对于每一个收到的数据包都会应答,RED 也正是利用这点来有效地控制 tcp 流的拥塞。但是 udp 提供的是不可靠的传输服务,目的端不会回应收到的数据包,因此 RED 算法也就不能有效地控制 udp 流的拥塞。当网络中的 udp 流逐渐增多时就会把采用拥塞避免算法的 tcp 流挤出去。

1.3　优化的随机早期检测算法 IM_RED

在 RED 算法中,$\max_p$ 随着平均队列 Q 在 $\min_{th}$ 和 $\max_{th}$ 之间增加而线性地增加,实验证明这样的设置很适合稳定的网络环境,但是对突变的网络环境不能快速地使网络稳定,导致网络拥塞,或是很长时间以后才使网络稳定。针对这一缺点的解决方法是:当平均队列 Q 超过 $\min_{th}$ 和低于 $\max_{th}$ 的一个选定的值 Q_{sel} 时,将 $\max_p$ 调到一个合适的值来避免拥塞的发生。

Q_{sel} 的选定:综合各种因素,文中选定 $Q_{sel} = \frac{1}{3} \times (\max_{th} - \min_{th})$。当平均队列 Q 小于 $\min_{th} + Q_{sel}$ 且大于 $\min_{th}$ 时调节 $\max_p$ 为当前值的一半,因为平均队列 Q 开始变小是 $\max_p$ 太大导致的;当平均队列 Q 大于 $\max_{th} - Q_{sel}$ 且小于 $\max_{th}$ 时调节 $\max_p$ 为当前值的一倍,因为平均队列 Q 开始变大是 $\max_p$ 太小导致的。

$\max_{xp}$ 的选定(为了区别 $\max_p$):根据 Q_{sel} 的选定可以很容易地得到 $\max_p$ 的选定公式,如式(4):

$$\max_{xp} = \begin{cases} 0 & Q < \min_{th} \\ \max_p \times 2 & \min_{th} \leqslant Q < \min_{th} + Q_{sel} \\ \max_p & \min_{th} + Q_{sel} \leqslant Q < \max_{th} - Q_{sel} \\ \max_p / 2 & \max_{th} - Q_{sel} \leqslant Q < \max_{th} \\ 1 & \max_{th} \leqslant Q \end{cases} \tag{4}$$

2　基于 NS－2 的仿真实验

2.1　NS－2 的扩展

NS－2[14,15]包括 C＋＋类和 Otcl 类,通过 Tclcl 类将两个类连接起来。RED 算法是在 C＋＋中实现的,它的类名是 REDQueue,优化的 IM_RED 在 REDQueue 类中增加一个 flag_ 变量和函数 Change-Maxp(double new_ave)即可,然后重新编译 NS－2,当 flag_ 为 0 时使用 RED 算法,flag_ 为 1 时使用 IM_RED 算法。

2.2　仿真过程及参数设置

在仿真中设置 10 条连接,其中 9 条 tcp 连接和 1 条 udp 连接:tcp 连接的带宽为 10Mbps,传输延时为 3ms;udp 连接的带宽为 100 Mbps,传输延时为 3ms。

两个路由器之间的带宽为 1 Mbps,传输延时为 4ms。仿真过程如图 2 所示。

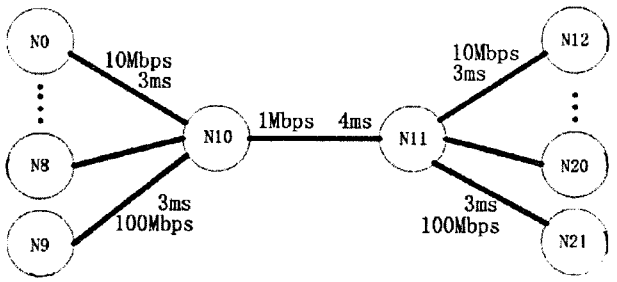

图 2 网络模拟图

其中 N0～N8 发送 tcp 连接,分别与 N12～N20 进行连接;N9 与 N21 建立 udp 连接。N0～N9 与路由器 N10 之间采用丢尾算法,路由器 N11 与 N12～N21 之间采用丢尾算法。两个路由器 N10 和 N11 之间采用 RED 算法或者 IM_RED 算法。其中 RED 的参数设置为:flag_ = 0 表示使用的是 RED 算法,queue_in_bytes_ = 0 表示平均队列的计算以数据包为单位,最小阈值 $\min_{th} = 10$,最大阈值 $\max_{th} = 20$,$\max_p$ 的初值为 0.01,权值 $\omega = 0.002$。IM_RED 算法只需更改 flag_ 的值为 1 即可。

2.3 RED 和 IM_RED 基于丢包率、吞吐量和延时的对比

在 NS-2 上运行使用 RED 算法的 moni.tcl 脚本和使用 IM_RED 算法的 moni1.tcl 脚本,产生两个数据文件 out_0.tr 和 out_1.tr。

(1)丢包率分析:利用 tcl 语言编写好的脚本程序 loss.awk 分别对 out_0.tr 和 out_1.tr 进行分析,得出采用 RED 算法的 tcp 数据流发送数为 2417,丢失数为 309,对 tcp 数据流的丢包率为 12.8%,udp 数据流的发送数为 1876,丢失数为 442,对 udp 数据流的丢包率为 23.6%;采用 IM_RED 算法的 tcp 数据流发送数为 2472,丢失数为 263,对 tcp 数据流的丢包率为 10.6%,udp 数据流的发送数为 1876,丢失数为 485,对 udp 数据流的丢包率为 25.9%。

由实验数据得出:相对于 RED 算法,IM_RED 算法有效地降低了对 tcp 数据流的丢包率,由原来的 12.8%降低到 10.6%;增加了对 udp 数据流的丢包率,由原来的 23.6%增加到 25.9%,使得网络资源对两种数据流的分配更加的公平。

(2)吞吐量分析:吞吐量是指网络传送二进制的速率,也称比特率或带宽,通常指比特率。把单位时间内在信道上成功传输的信息量定义为吞吐量,即指单位时间内成功传送的无差错通信量,即平均传输速率(Mbps)。

采用 RED 算法时,tcp 数据流的平均吞吐量为 596.76bps, udp 数据流的平均吞吐量为 760.47bps;采用 IM_RED 算法时,tcp 数据流的平均吞吐量为 641.96bps,udp 数据流的平均吞吐量为 745.61bps。

由实验数据得出:改进的 IM_RED 算法提高了 tcp 数据流的平均吞吐量,降低了 udp 数据流的平均吞吐量,使两种流更加公平地占有网络资源。

两种算法下 udp 数据流的吞吐量对比如图 3 所示。

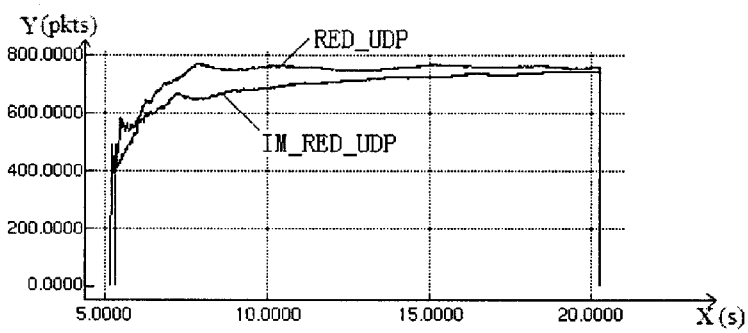

图 3 基于 RED 和 IM_RED 的吞吐量对比

(3)延时分析:利用瞬时延时分析脚本 end_delay.awk,分析 out_0.tr 和 out_1.tr 得出文件 ed0 和 ed1,再利用平均延时分析脚本 ave_delay.awk 对结果文件 ed0 和 ed1 分析得出平均延时。利用 RED 算法的脚本得出的平均延时为 0.11333s;利用 IM_RED 算法得出的平均延时为 0.11245s。

由实验数据得出:相对于 RED 算法,IM_RED 算法有效地减小了数据传输时的时延,充分地利用了有限的网络资源。

以上的分析如表 1 所示。

表 1 基于 RED 和 IM_RED 的结果分析(udp=100 Mbps)

网络性能	RED 算法		IM_RED 算法	
(udp 带宽 100 Mbps)	tcp 数据流	udp 数据流	tcp 数据流	udp 数据流
丢包率(%)	12.8	23.6	10.6	25.9
吞吐量(bps)	596.76	760.47	641.96	745.61
延时(s)	0.11333		0.11245	

2.4 改变 udp 数据流的带宽

通过改变 udp 数据流占有的带宽,更好地验证 IM_RED 算法相对于 RED 算法在三种网络性能下的有效性。整个过程只需要改变两个模拟脚本中 udp 数据流的带宽即可,选取 udp 数据流的带宽为 150Mbps,结果如表 2 所示。

在增大 udp 带宽时,丢包率和延时变化不是很明显,只有吞吐量有变化,这是因为网络带宽的占用已接近上限,增大发送端的发送速率,已无法再争用更多的

网络资源。但相对于 RED 算法,IM_RED 算法仍然有效地抑制了 udp 流对带宽的占用,使得带宽的分配更加的公平。

表 2　基于 RED 和 IM_RED 的结果分析(udp=150 Mbps)

网络性能 (udp 带宽 150 Mbps)	RED 算法		IM_RED 算法	
	tcp 数据流	udp 数据流	tcp 数据流	udp 数据流
丢包率(%)	12.8	23.6	10.5	25.9
吞吐量(bps)	596.76	760.47	614.96	745.61
延时(s)	0.11333		0.11245	

3　结束语

仿真结果表明:相对于 RED 算法,优化后的 IM_RED 算法提高了 udp 数据流的丢包率,降低了 tcp 数据流的丢包率;降低了 udp 数据流的吞吐量,提高了 tcp 数据流的吞吐量;降低了两种数据流的平均延时。IM_RED 算法使得数据流对网络资源的占有更加公平,充分地利用了有限的网络资源。

文中针对 RED 算法参数配置难和公平性差进行算法上的优化,一定程度上弥补了算法在丢包率、吞吐量和延时上的缺点。不足之处是没有根本上解决公平性问题,有待更深入的优化以解决公平性的问题。

参考文献:

[1] 刘　轶,肖凯平,刘楚达,等. QoS 接纳控制算法性能分析与比较[J]. 计算机工程,2005,31(20):113－116.

[2] Yamaguchi T, Takahashi Y. A queue management algorithm for fair bandwidth allocation[J]. SienceDirect/Computer Communication, 2007, 30(9): 2048－2059.

[3] Abbasov B, Korukoglu S. Effective RED: An algorithm to improve RED's performance by reducing packet loss rate[J]. SienceDirect/Journal of Network and Computer Applications, 2009, 32(3): 703－709.

[4] Manfredi S, di Bernardo M, Garofalo F. Design, validation and experimental testing of a robust AQM control[J]. SienceDirect/Control Engineering Practice, 2009, 17(3): 394－407.

[5] 杨吉文,张卫东. 基于 NS2 的主动队列管理仿真研究[J]. 计算机工程,2006,32(17):189－191.

[6] 吴　忠,范君晖. TCP 流拥塞窗口的非线性动力学行为仿真分析[J]. 计算机仿真,2005,22(9):280－284.

[7] 谌新年. D－RED:一种改进的路由器拥塞控制算法[J]. 计算机工程与科学,2007,29(5):45－49.

[8] Floyd S, Fall K. Promoting the Use of End to End Congestion Control in the Internet[J]. IEEE/ACM Transaction Networking, 1999, 7(4): 458－472.

[9] Floyd S, Jacobson V. Random Early Detection Gateways for Congestion Avoidence[J]. IEEE/ACM Transaction Networking, 1993, 1(4): 397－413.

[10] 文　宏,唐玉华,朱培栋. RED 簇主动管理算法研究[J]. 计算机工程与科学,2006,28(5):66－69.

[11] 任丰原,林　闯,王福豹. RED 算法的稳定性:基于非线性控制理论的分析[J]. 计算机学报,2002,25(12):1302－1307.

[12] 葛志辉,陈志刚. 一种新的基于 RTT 的丢包率估计算法[J]. 计算机工程与应用,2005,41(19):26－27.

[13] 赵　金,王晓辉. 网络瓶颈带宽测量建模分析[J]. 系统仿真学报,2004,16(4):859－861.

[14] 徐雷鸣,庞　博,赵　耀. NS 与网络模拟[M]. 北京:人民邮电出版社,2003:57－72.

[15] 王晓燕,郑明春. 基于 NS2 的网络仿真研究与应用[J]. 计算机仿真,2004,21(12):128－131.

(上接第 187 页)

发框架。

参考文献:

[1] 田海利. 第三方物流仓储管理系统的设计与实现[D]. 长沙:国防科学技术大学, 2006.

[2] 白　英. 基于 StrutS 架构的库存管理系统设计与实现[D]. 大连:大连海事大学,2008.

[3] 隋　永,周家纪. MVC 在 J2EE 框架中的应用研究[J]. 计算机技术与发展,2006,16(12):119－120.

[4] 金　焱,董英茹. JavaEE 架构及 EJB3 在企业级开发中的应用[J]. 工程科技,2009(6):174－175.

[5] 蒋琴芬,蔡玉华,刘　婕. 一种基于 Struts＋EJB 框架的信息服务系统的设计与实现[J]. 东华大学学报:自然科学版,2006,32:56－57.

[6] 丁振兰,杨宝祝,吴华瑞,等. 基于 Struts＋EJB 拓展框架的小城镇电子政务[J]. 计算机技术与发展,2006,16(2):39－40.

[7] 贾　伟. JSP 环境下基于 Struts 的 MVC 模式在仓储管理系统中的应用[J]. 价值工程,2007(2):98－99.

[8] SUN. Remote Method Invocation Home[EB/OL]. 2001. http://java.sun.com/javase/technologies/core/basic/rmi/index.jsp.

[9] 苏林凤,郭跟成. EJB 性能优化设计方案[J]. 计算机工程,2007,33(22):102－103.

[10] 朱岸青,王会进,高河福. J2EE 系统的 EJB 技术性能测试和优化[J]. 暨南大学学报,2006,26(3):369－370.

[11] Vasiliev Y. Introducing EJB 3 and the Java Persistence API [M]. [s.l.]:Apress.2008.

[12] Beschorner K, Rosenstiel W. Untersuchungen zur effizienten Kommunikation in EJB － Systemen [J]. Informatic － Forschung und Entwicklung, 2004, 18(2): 78－80.