高技术通讯 2013 年 第23 卷 第3 期:228 ~236

doi:10.3772/j.issn.1002-0470.2013.03.002

基于历史连接信息改进的单边加速 FAST TCP 算法①

陈晓龙② 彭志平

(广东石油化工学院计算机与电子信息学院 茂名 525000)

摘 要 针对 FAST TCP 存在难以选择协议参数的公开问题,根据 FAST TCP 单边加速应用的特点,提出了一种改进的2层结构的传输层算法。用此算法时,在下层,记录各活跃的 FAST TCP 连接小时间尺度收集最大排队延时等历史信息,上层算法充分利用历史连接能够提供的最大排队延时等信息,确定目标排队延时的控制范围,大时间尺度周期动态调整协议参数,通知给下层运行的 FAST TCP 连接,主动控制瓶颈链路的队列长度,避免队列溢出。NS-2 仿真实例证明,该改进算法可大幅提高系统的稳定性和利用率。

关键词 FAST TCP, 协议参数, 排队延时, 历史连接, 主动控制

0 引 言

FAST TCP[1,2] (Fast Active queue management Scalable Transmission Control Protocol,简称 FAST)是针对下一代高速网络提出的一种新型传输控制协议。在高速网络环境中,与基于分组丢失作为拥塞反馈信号的其他改进 TCP 协议相比,FAST 协议采用估计的排队延时信息计算 FAST 连接留在瓶颈链路缓冲区的分组个数,以期望留在瓶颈链路缓冲区分组的个数为平衡点,根据实际留在瓶颈链路缓冲区的分组个数距离平衡点位置的远近,非线性地调整发送窗口大小变化的快慢,不需要网络层中间节点参与,主动控制留在瓶颈链路缓冲区的队列长度,从而主动避免了缓冲区队列溢出和拥塞现象的出现,取得了更好的稳定性和更充分的瓶颈链路使用效率,其带宽利用率可达90%以上[1]。但其存在如何选择合适的协议参数这个公开问题。正是这个关键的公开问题阻扰了 FAST 协议在高速网络进一步应用和推广[1-4]。为此,本文提出了一种基于历史连接信息的单边加速 FAST TCP 改进算法,该算法可有效解决这个问题。

1 相关研究

FAST 系统主动控制留在瓶颈链路缓冲区的队列长度与 FAST 协议参数 α 和活跃的 FAST 连接数 n 有关[5-12]。协议参数 α 指 FAST 连接期望留在瓶颈链路端缓冲区的数据的分组个数。目前比较常用的是根据瓶颈链路带宽采用静态映射表选一次性指定协议参数 α[5,6]。但随着 FAST 连接的不断建立,n 会不断增大,这些活跃的连接主动控制留在瓶颈链路缓冲区的队列长度 $n\cdot\alpha$ 也会不断增加,当主动控制留在瓶颈链路缓冲区的队列长度 $n\cdot\alpha$ 超过瓶颈链路缓存容量 B 时,就会产生溢出。文献[7-12]表明出现这种情况时,FAST 系统会出现缓存溢出、丢失恢复、突发速率三种状态的循环往复,最终导致严重的报文段丢失和低质量的网络服务。

但由于在源端无法获得准确的经过瓶颈链路的 FAST 连接数 n 和瓶颈链路缓存容量 B,因此难以选择合适协议参数 α。文献[7]提出了在分钟数量级尺度下,根据排队延时和平均丢包速率动态调整调整协议参数 α 的策略。但该策略把焦点放在协议间的公平而非协议内部公平性上。文献[8]提出了根据当前 α 值与预计值 α_0 间的差值来动态调整参数 α 的策略,但该策略没有给出如何选取预计值 α_0 的方法。文献[9]提出了根据目标排队延时动态调整协议参数 α 的 AFAST TCP 算法,但算法也没有具体给出如何确定目标排队延时的方法。文献[10]注意到单靠 FAST 在源端是很难解决选择合适协议参

① 国家自然科学基金(61272382),广东省自然科学基金(s2011010003667),广东石油化工学院博士启动(511017)和茂名市科技计划(203634)资助项目。

② 男,1971 年生,副教授,博士;研究方向:网络拥塞控制,非线性控制;联系人,E-mail: xlycxl@126.com

(收稿日期:2012-09-25)

数 α 这个公开问题,因此提出在 FAST 源端系统上增加一层面向主干瓶颈链路的宏观指导功能,周期性(如数十秒或数分钟)地测量网络主干瓶颈链路的瓶颈带宽、往返时延等性能指标,根据结果推测竞争连接数量,然后由期望排队长度计算出每个连接的协议参数,统一通知相应 FAST 源端。文献[11]改进了上述根据测量结果推测竞争连接数量的方法,采用模糊控制技术指导 FAST 源端选择适当的协议参数 α。文献[12]进一步将上述方法整理,提出了一种 2 层结构的传输层解决方案,利用上层的性能服务感知内部网络状态,然后在此基础上形成基于冗余分组的下层控制目标。综上所述可知,文献[8,9]虽提出了相应的改进方案,但实际上只是将对协议参数 α 的设置转为对期望 α_0 值和期望排队延时的设置,没有从根本上解决设置协议参数 α 这个公开问题,并且,由于各连接无法直接通信,各连接无法同步调整协议参数,存在协议参数调整收敛速度慢的问题。文献[10-12]虽然解决了该公开问题,但需引入外部测量技术等方法,存在部署问题。总之,目前还没有很好的办法来解决 FAST 协议存在的这个公开问题,因此希望根据 FAST TCP 商业应用的实际情况来找到解决该方法。

2006 年成立于美国加州的 FastSoft 公司,推出了广域网单边加速产品 FastSoft E Series™,该产品采用了加州理工学院 2004 年研发出的突破性网络优化技术 FAST TCP,其网络工作拓扑如图 1 所示。

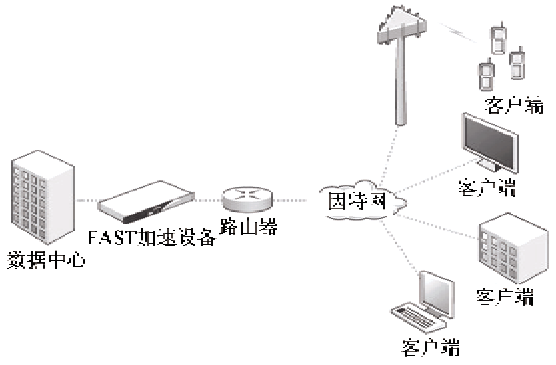

图 1　单边加速产品 FastSoft E Series™ 应用

由图 1 可知,只要在服务器端简单串联一个标准 1U 服务器大小 FastSoft E Series™ 设备,在客户端无需新增软件或浏览器插件,且在服务器端也无需修改配置或重写代码即可实现单边加速功能,在全球的任一地点访问该服务器均可享受到动态页面文件传输的 30% 到 500% 加速,主要应用领域:(1)软件即服务—云计算(software as a Service(SaaS)-part of cloud computing);(2)视频/游戏加速(video/gaming acceleration);(3)文件传输(file transfer);(4)内容分发网络(content delivery network,CDN)。主要客户包括 MySpace、Overstock. com、Limelight Networks、JWT、Thomson Technicolor、Honda、Siemens 等。

本文根据上述 FAST 单边加速的实际应用,提出了一种 2 层结构的传输层解决方案,下层仍使用 FAST 算法,但每个连接需要增加最大排队延时参数,并小时间尺度周期维护这个变量。当发生丢包时通知上层算法。上层算法发现下层有连接丢包后,读取下层各活跃连接[12]的最大排队延时,根据该最大排队延时确定目标排队延时控制范围,每隔一个分钟数量级尺度更新周期,自适应调整协议参数,通知给下层运行的 FAST 连接,主动控制瓶颈链路缓冲区队列长度在合理的范围,防止瓶颈链路缓冲区丢包,提供系统的稳定性和利用率。本文的最大创新点在于系统只要通过一次丢包,上层算法就可估计到瓶颈链路的最大排队延时,确定排队延时的目标控制范围,自适应调整协议参数,主动控制瓶颈链路的队列长度,避免了系统大量丢包行为,大大提高了系统的稳定性和利用率,仿真结果充分表明了这一点。

2　FAST 单边加速系统描述

根据图 1 的实际网络模型可知,只是实现服务器到各客户端的单边加速功能,故可将 FastSoft E Series™ 设备抽象成一个信源主机节点。信源主机和各信宿主机中建立的 FAST 连接中总会有一条瓶颈链路,因此可构建如图 2 所示的网络拓扑图。在图 2 中,假设 S_1 为信源主机节点,D_1、D_2、…、D_N 信宿主机节点。中间节点 L_1、L_2 组成瓶颈链路 l。信源主机、若干条相连链路和信宿主机组成一条路径如路径:S_1-L_1-L_2-D_1 包含信源主机 S_1、链路 L_1-L_2、和信宿主机 D_1,每条路径可建立多个 FAST 连接。

如图 2 所示,假设 FAST 单边加速网络系统建立了 N 条FAST连接(文后简称连接),共享唯一瓶

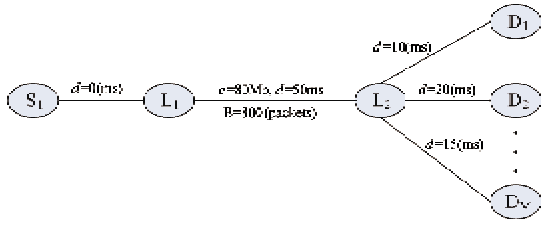

图 2　FAST 单边加速系统网络拓扑

颈链路 l，定义连接集合 $I = \{1,2,\cdots,N\}$。

对于任给的连接 $i \in I$，有：$W_i(t)$：发送端拥塞窗口大小(packet)；d_i：传播时延(s)；$q_i(t)$：排队时延(s)；$D_i(t)$：往还时延，其中 $D_i(t) = d_i + q_i(t)$(s)；D_i：平均往返延时(s)；$x_i(t)$：传输流量(packet/s)，其中 $x_i(t) = w_i(t)/D_i(t)$；α_i：协议参数(packet)；γ_i：控制律增益参数；c：瓶颈链路 l 的带宽(packet/s)；T：FAST 算法窗口更新周期(s)。

由文献[2,5,6]可知，各 FAST 连接初始协议参数 $\alpha_i = 50, \gamma_i = 0.5$。

其窗口按照算法 1 进行更新[2]。

算法 1：

每隔一周期 T，$w_i(k+1) = (1-\gamma_i)\cdot w_i(k) + \gamma_i\left(\frac{d_i}{d_i + q_i(k)}\cdot w_i(k) + \alpha_i\right)$，其中 $w_i(k)$ 为 $k \cdot T$ 时刻的源端发送窗口大小。

文献[13]提出了一种改进的 FAST 算法，该改进的 FAST 算法在决定本次窗口调整策略前，还考虑了前一往返周期的发送窗口大小，其窗口按照算法 2 进行更新。

算法 2：

每隔一周期 T，$w_i(k+1) = (1-\gamma_i)\cdot w_i(k) + \gamma_i\left(\frac{d_i}{d_i + q_i(k)}\cdot w_{i,old}(k) + \alpha_i\right)$，其中 $w_i(k)$ 同算法 1，$w_{i,old}(k)$ 为 $k \cdot T - D_i$ 时刻源端发送窗口大小。

文献[14]通过理论分析和实验仿真验证了该算法 2 比算法 1 具有更好的稳定性但响应速度要慢，并根据上述两种系统[2,13]的特性提出了进一步的改进算法 3。

算法 3：

每隔一周期 T，$w_i(k+1) = w_i(k) + \gamma_i\left(-(1-r)\cdot w_i(k) + \left(\frac{d_i}{d_i + q_i(k)} - r\right)\cdot w_{i,old}(k) + \alpha_i\right)$，其中 $w_i(k)$、$w_{i,old}(k)$ 同算法 2。r 为加权系数，当 r 为 0 时，算法 3 完全退化到算法 2，具有很好的稳定性但响应速度慢；当 r 取值较大时，上述算法的性能和算法 1 接近。文献[14]利用 Lyapunov-Razumikhin 定理证明了 $r \in [0,0.5)$ 时，上述改进 FAST 算法 3 系统全局稳定。

由文献[14]可知，当协议参数 α 取值较小时，系统的稳定性会下降，因此本文在动态调整协议参数 α 的同时，将同时调整加权系数 r，确保系统能在稳定性和响应速度等性能上取得一定的平衡。

3 动态调整协议参数算法

3.1 算法基本思想

构建如图 3 所示的 2 层算法。下层算法仍使用上述改进的 FAST 算法 3，但每个连接需要在小时间尺度周期维护最大排队延时，当发生丢包时要立即通知上层算法。上层算法的主要目的是每隔一大时间尺度更新周期，向下层改进的 FAST 源端提供一个合适的协议参数 α 和调整加权系数 r。其设计思想是：在初始状态时，若没有连接丢包，则直接采用静态映射表方法[5,6]选用协议参数。如发现下层有 FAST 连接丢包，则取丢包前一个小尺度周期内所有连接测到最大的瓶颈链路的排队延时作为瓶颈链路的最大排队延时，根据最大排队延时设置期望的目标排队延时控制范围，每隔一个大尺度更新周期进行协议参数的调整，使得瓶颈链路排队延时能够控制在期望的范围内。

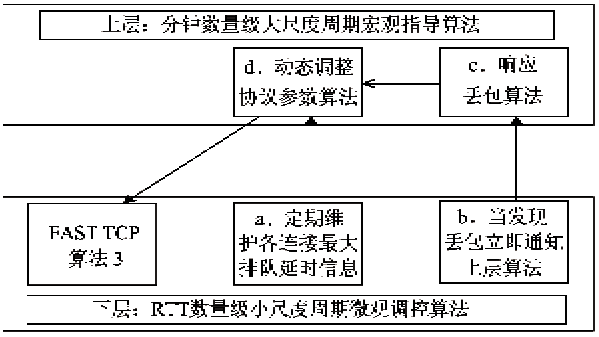

图 3 2 层结构的改进算法

3.2 下层改进算法

假设上层其小尺度更新周期为 T_b，大尺度更新周期为 T_α。

在原有算法 3 的基础上，对任给的连接 $i \in I$ 设置如下变量：

$q_{i,\max}$：每更新周期 T_b 内最大排队延时。

a. 定期维护各连接信息

每隔更新周期 T_b，重新设置 $q_{i,\max} = 0$。//只求每隔更新周期 T_b 的最大排队延时。

每收到一个确认帧 {

If $q_i(k) > q_{i,\max}$

$q_{i,\max} = q_i(k)$ //记录每个连接最大的排队延时。}

b. 若丢包通知上层算法

If(收到三个相同的确认帧,即发现有丢包现象)

通知上层算法,瓶颈链路发生丢包;

}

3.3 上层改进算法

建立一个缓存表,用来存放近期历史 FAST 连接参数信息[15]:

$maxalpha$:最大协议参数;$minalpha$:最小协议参数;$zmaxq$:小尺度周期内 T_b 所有连接的最大排队延时;$maxq(i)$:小尺度周期 T_b 内第 i 个连接的最大排队延时,算法每次启动时,获取该周期的最大排队延时,即设置 $maxq(i)=q_{i,\max}$;$lastmaxq$:上一个周期内所有连接的最大排队延时;$targetq$:目标排队延时;$targetq_{\min}$:目标排队延时最小值;$targetq_{\max}$:目标排队延时最大值;$alpha$:提供给下层的协议参数 α;$coefr$:提供给下层调整加权系数 r。

初始状态时,设置 $targetq_{\min}=0$,$targetq_{\max}=0$,$targetq=0$。

c. 响应丢包算法

Proc repondloss(){

If 没有发现有 FAST 连接丢包,则不启动动态调整协议参数算法。

If(收到下层有连接发出的丢包信号)

对每个连接:

$maxq(i)=q_{i,\max}$;//读取其最大排队延时。

If $maxq(i)>zmaxq$ $zmaxq=maxq(i)$;//求出所有连接的最大排队延时。

$targetq_{\min}=\theta_1*zmaxq$;

$targetq_{\max}=\theta_2*zmaxq$;//求出目标排队延时控制范围。$0<\theta_1<\theta_2<1$

adapitvepara();//一旦丢包就启动动态调整协议参数算法。

}

该改进算法一旦收到下层有 FAST 连接发生丢包,就马上启动动态调整协议参数算法。为避免频繁调整协议参数,在响应丢包进入到动态调整协议参数算法后,一个更新周期 T_α 内,不会再响应丢包启动上述算法。

d. 动态调整协议参数算法

Proc adapitvepara(){

每更新周期 T_α 或因为丢包执行如下算法:{

若本次算法因为丢包而触发,则重新计时,直到隔一个周期 T_α 后才可以调用本算法,以保住调整的分钟数量级尺度。

If $targetq_{\min}>0$ {//有连接发生丢包,获得了最大排队延时和目标排队延时的范围。

$lastmaxq=zmaxq$;$zmaxq=0$;//记录上个周期的最大排队延时。

对每个连接: {

$maxq(i)=q_{i,\max}$;$q_{i,\max}=0$;//读取在该更新周期内的最大排队延时。

If $maxq(i)>zmaxq$

$zmaxq=maxq(i)$;//求出本更新周期的最大排队延时。

}

$zmaxq=(1-\theta)lastzmaxq+\theta zmaxq$;//平滑处理

If($zmaxq>targetq_{\min}$ and $zmaxq<targetq_{\max}$)

{不用采取措施,目标已满足期望排队延时范围;}

Else if $zmaxq\leqslant targetq_{\min}$

{ $\eta_1=(targetq_{\min}-zmaxq)/targetq_{\min}$

//确定离期望范围远近位置,根据位置远近确定算法调整幅度

$targetq=targetq_{\min}+\eta_1(targetq_{\max}-targetq_{\min})$

//根据位置确定目标排队延时

}

Else if $zmaxq\geqslant targetq_{\max}$

{

$\eta_2=\theta_2(zmaxq-targetq_{\max})/targetq_{\max}$

//确定离期望范围位置,根据位置远近确定算法调整幅度

$targetq=targetq_{\max}-\eta_2(targetq_{\max}-targetq_{\min})$

//根据位置确定目标排队延时

}

$diff=targetq/zmaxq$;//调整因子为本更新周期内求出的最大排队延时与目标排队延时比值

$alpha_{new}=diff*alpha$;//自适应调整协议参数

If $alpha_{new}>alpha_{\max}$ $alpha=alpha_{\max}$

else if $alpha_{new}<alpha_{\min}$ then $alpha=alpha_{\min}$

else $alpha=alpha_{new}$ //规范化处理

$\eta=(alpha-alpha_{\min})/(alpha_{\max}-alpha_{\min})$

// 求协议参数 $alpha$ 在最大取值的比值

$coefr=0.5+0.5\cdot(e^{\kappa*\eta}-1)/(e^{\kappa}-1)$

$(\kappa<-1)$ //

}

将协议参数 $alpha$ 和调整加权系数 $coefr$ 通知给

下层。

由算法中 η 和 *coefr* 的计算公式可知，*alpha* 取值越大，η 越大，*coefr* 取值也越大。这是因为若 *alpha* 取值大，则系统稳定性好。因此，在确保系统稳定的情况下，可考虑取较大的 *coefr*，确保系统的快速响应性。

4 仿真验证

通过 NS2.31 仿真来验证算法的有效性，仿真采用的网络拓扑如图 2 所示。假设 M = 3，即有 3 个信宿主机 D_1、D_2 和 D_3，构成路径 S1-L1-L2-D1（文后简称路径 S1-D1，其余类推）、S1-L1-L2-D2 和 S1-L1-L2-D3。假设 S1-L1、L2-D1、L2-D2 和 L2-D3 都有足够的带宽和链路缓存容量，不会拥塞和丢包，但 L2 到 3 个信宿主机传播延时各不相同，分别是 10ms、20ms 和 15ms。瓶颈链路 L1-L2 的带宽 c = 80Mb/s，传播延时为 50ms，瓶颈链路缓存容量 B = 800packet。假设 1packet = 1000byte = 8000bit，因此 c = 80Mb/s = 10000packet/s。

实验 1：验证当采用上述算法 1，当连接数增加，缓冲溢出时，系统的稳定性差，利用率低。

根据瓶颈链路的带宽 c = 80 Mb/s，采用静态映射表方法确定各连接协议参数 α_i = 50，γ_i = 0.5[5,6]。仿真时间 1000s，3 条路径各建立 10 个 FAST 连接。路径 1 在 0s 时建立连接，1000s 时结束；路径 2 在 150s 建立连接，600s 结束。路径 3 在 400s 建立连接，1000s 结束。假设各连接都能够获得准确的传播延时，不存在公平性问题。仿真结果如图 4 至图 6 所示。

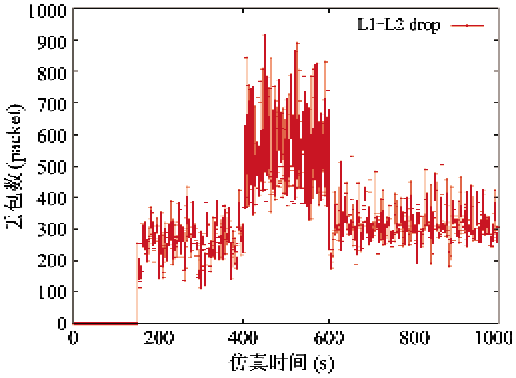

图 4 瓶颈链路缓冲区丢包个数

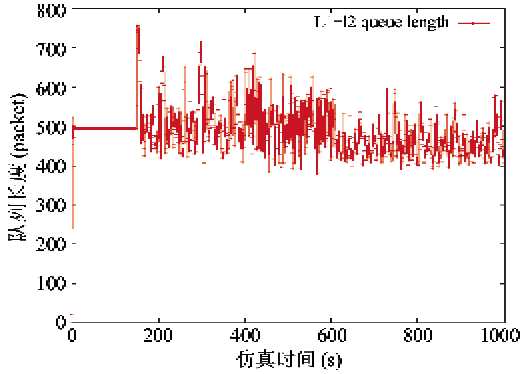

图 5 瓶颈链路缓冲区队列长度

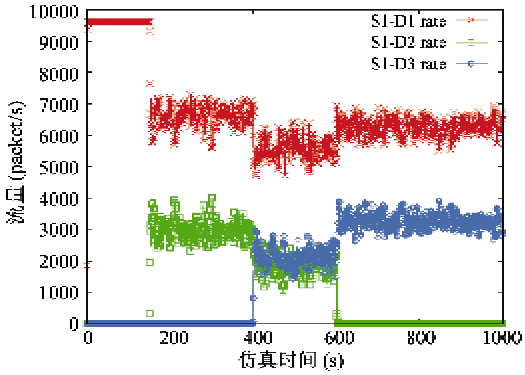

图 6 三条路径分配的传输流量

仿真结果分析：由图 4 至图 6 可知，在前 150s，当只有路径 S1-D1 的 10 个活跃的 FAST 连接时，n = 10。因此 $n \cdot \alpha < B$，没有丢包情况发生，瓶颈链路队列长度稳定，系统利用率高。在 150s 时，由于路径 S1-D2 建立了 10 个 FAST 连接，此时 n = 20，因此 $n \cdot \alpha > B$，瓶颈链路缓冲区溢出，开始出现丢包，瓶颈链路队列长度开始出现振荡。当到 400s 时，路径 S1-D3 又建立了 10 个 FAST 连接，n = 30，因此，瓶颈链路缓冲区更加拥挤，丢包更加剧列。

实验 2：在和实验 1 的仿真环境、参数设置相同的情况下，但源端不采用上述 FAST 算法 1，采用本文提出的动态调整协议参数算法。设置 *maxalpha* = 50，*minalpha* = 3，T_b = 0.5s，T_α = 60s，θ_1 = 0.4，θ_2 = 0.6。仿真结果如图 7 至图 12 所示。

仿真结果分析：

由图 7 至图 12 的仿真结果可知，在 150s 时，由于路径 S1-D2 建立了 10 个 FAST 连接，瓶颈链路缓冲区溢出，开始出现丢包，下层连接发现丢包，会马上通知上层算法。上层收到丢包信息，马上启动动态调整协议算法，调整相应的协议参数 *alpha* 和调整加权系数 *coefr*，并及时通知给下层。下层算法根据新的协议参数进行运算。由图7至图9可知，各连

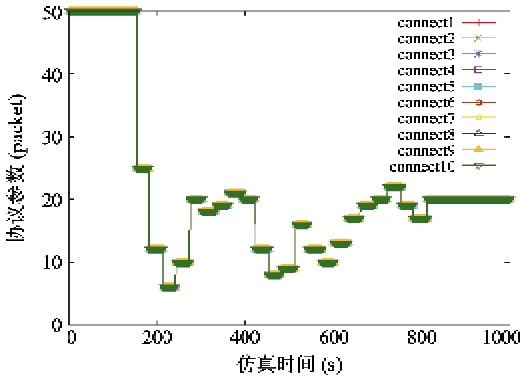

图 7　路径 S1-D1 各连接协议参数变化情况

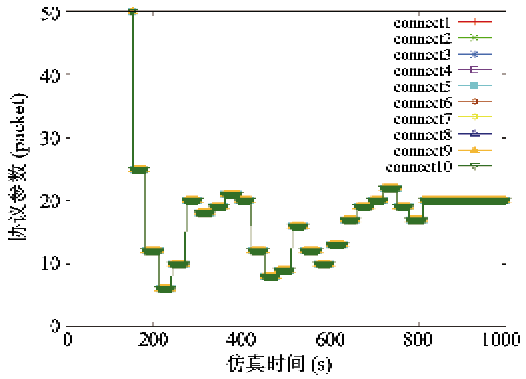

图 8　路径 S1-D2 各连接协议参数变化情况

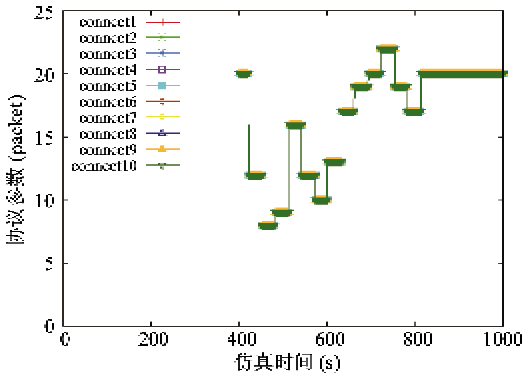

图 9　路径 S3-D3 各连接协议参数变化情况

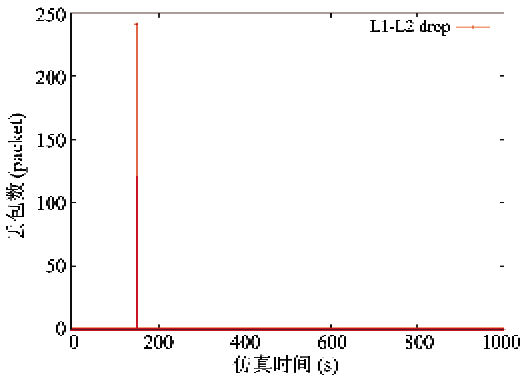

图 10　瓶颈链路缓冲区丢包个数

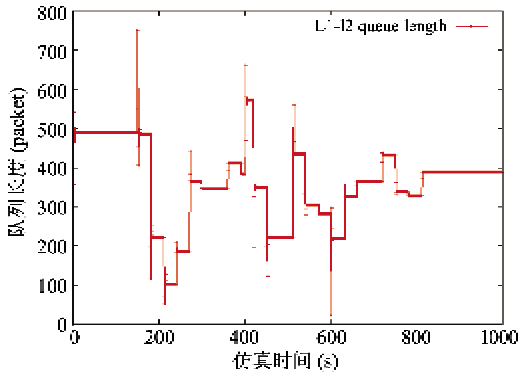

图 11　瓶颈链路缓冲区队列长度

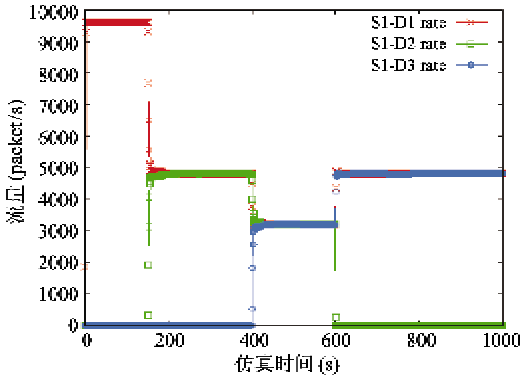

图 12　三条路径分配的传输流量

接都从上层算法大时间尺度获得了统一的协议参数 $alpha$。对比图 4 和图 10 可知,由于采用了动态协议参数调整算法,各连接只发生了一次丢包。对比图 5 和图 11 可知,瓶颈链路队列长度和排队延时能够控制在期望的范围内,并表现出较好的稳定性。对比图 6 和图 12,各路径公平地平分了瓶颈链路带宽,系统呈现出好的稳定性和高利用率。

实验 3:模拟现实环境下,瓶颈链路发生变化,即验证在瓶颈链路缓冲容量 B 发生变化的情况下,采用动态调整协议参数算法的有效性。

与实验 2 仿真环境参数不同之处如下:

(1) 瓶颈链路缓冲区 B 在 0 ~ 200s 内为 1200 (packet),200 ~ 400s 设置为 800, 400 ~ 1000s 设置为 600。

(2) 路径 S2-D2 改为在 0s 建立 10 个连接,在 800s 结束。其他仿真环境、参数设置与算法 2 相同。

仿真结果如图 13 至图 15 所示。

仿真结果分析:

由图 13 至图 15 的仿真结果可知, 在 200s 时, 活跃的连接数 $n = 20$, 瓶颈链路缓冲由 1200packet 变为 800packet,因此有 $n \cdot \alpha > B$, 瓶颈链路缓冲区

溢出，开始出现丢包，动态调整协议参数算法启动，自适应调整协议参数 α。当到达 400s 时，又有新的连接建立，到 600s 时，瓶颈链路缓冲进一步变小，800s 连接释放等情况发生，但本文提出的算法都能自适应调整。由图 14 和图 15 可知，能够把队列控制在适当的范围内，减小了丢包情况发生，各路径也能够公平地分配带宽。

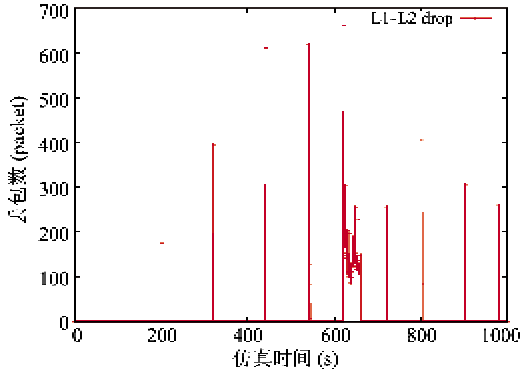

图 13 瓶颈链路缓冲区丢包个数

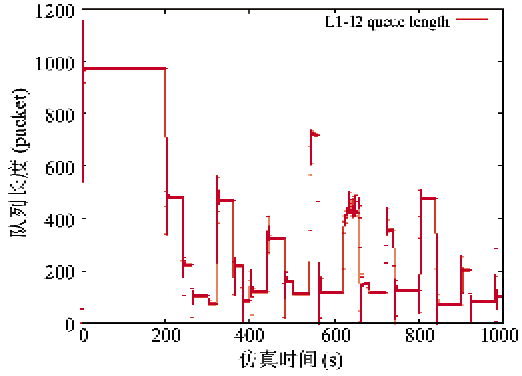

图 14 瓶颈链路缓冲区队列长度

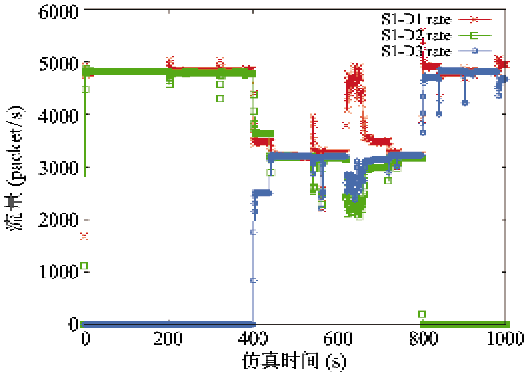

图 15 三条路径分配的传输流量

实验 4： 验证各路径随机建立连接情况下算法 4 的有效性。

与实验 2 仿真环境参数不同之处如下：

假设各路径 TCP 连接数不固定，但已统计到各路径 FAST 连接到达服从参数 ν 的泊松过程、其传送文件尺寸服从参数 u 指数分布随机变量。假设参数 $\nu = 1/6$（connect/s），$u = 1/20000$（packet），实际是统计意义下平均每 6s 建立一个连接，每个连接的统计平均大小是 20000packet，其他仿真环境参数设置与实验 2 相同。图 16 至图 18 是在源端采用算法 1 时的仿真结果。图 19 至图 21 是在源端采用本文提出算法时的仿真结果。表 1、表 2 是分别采用算法 1 和本文提出的算法各做 5 次实验下的统计结果。

仿真结果分析：

图 16 至图 18 和表 1 是采用算法 1 的情况下得到的运算结果。图 19 至图 21 和表 2 是在本文提出的算法的情况下得到仿真结果。对比这两种仿真结果可以发现，在采用算法 4 的情况下，瓶颈链路丢包大幅下降，瓶颈链路队列长度能够主动控制在合理范围内，各路径获得了较好的公平性。表 1 和表 2 列出了 5 组数据，反映了各路径完成的连接数、每连接平均发送时间和所有连接的平均发送时间。对比表 1 和表 2 可知，平均每连接发送时间由 58.1s 降低到 39.9s，发送速度提高了 31.3%。

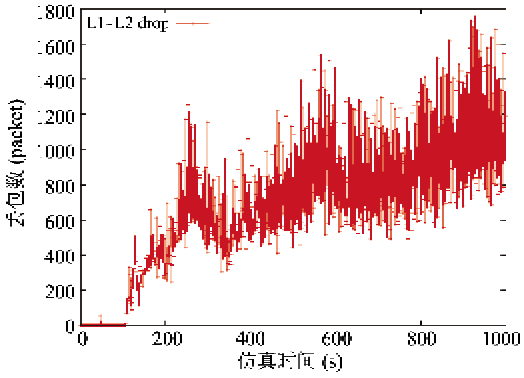

图 16 瓶颈链路缓冲区丢包个数

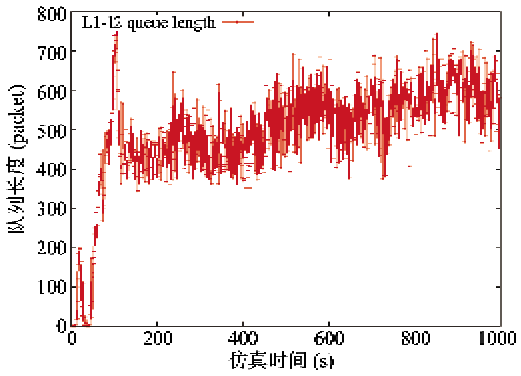

图 17 瓶颈链路缓冲区队列长度

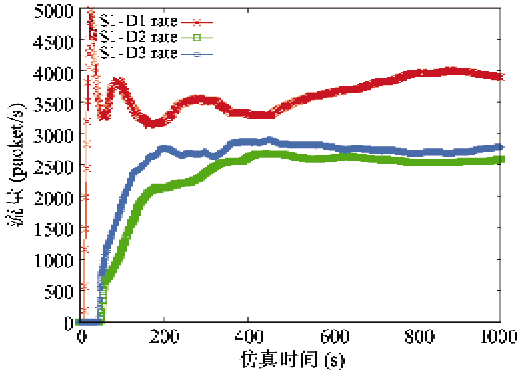

图 18　三条路径分配的传输流量

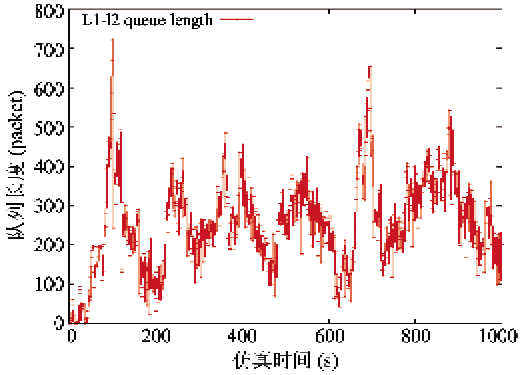

图 20　瓶颈链路缓冲区队列长度

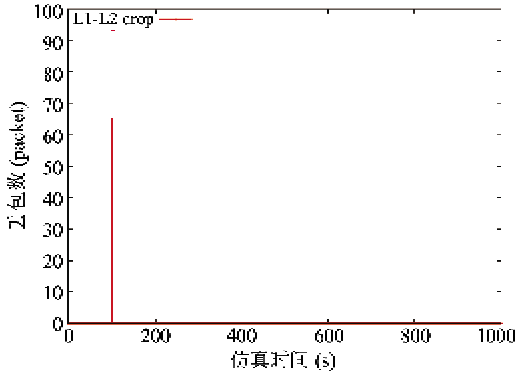

图 19　瓶颈链路缓冲区丢包个数

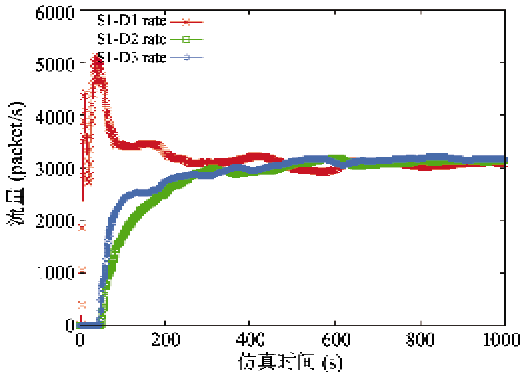

图 21　三条路径分配的传输流量

表 1　采用算法 1 各路径完成发送 FAST 连接情况(时间单位:s)

序号	S1-D1 完成连接数	S1-D1 每连接平均发送时间	S1-D2 完成连接数	S1-D2 每连接平均发送时间	S1-D3 完成连接数	S1-D3 每连接平均发送时间	总完成连接数	总平均发送时间
1	163	30.2	122	83.6	143	52.1	428	52.7
2	182	45.2	107	99.6	133	75.8	422	68.7
3	158	33.5	118	86.5	143	56.7	419	56.4
4	174	31.0	118	78.1	140	51.1	432	50.4
5	164	31.1	121	100	146	65.0	431	62.1
平均	168.2	34.2	117.2	89.6	141	60.1	426.4	58.1

表 2　采用动态调整协议参数算法各路径完成发送 FAST 连接情况(时间单位:s)

序号	S1-D1 完成连接数	S1-D1 每连接平均发送时间	S1-D2 完成连接数	S1-D2 每连接平均发送时间	S1-D3 完成连接数	S1-D3 每连接平均发送时间	总完成连接数	总平均发送时间
1	165	38.0	123	36.4	141	32.9	429	35.9
2	147	38.9	129	41.3	148	38.4	424	39.5
3	163	31.8	131	40.6	150	35.0	444	35.5
4	152	44.0	125	49.2	146	46.8	423	46.5
5	147	39.7	133	46.6	151	40.1	431	42.0
平均	154.8	38.5	128.2	42.8	147.2	38.6	430.2	39.9

5 结 论

目前,FAST TCP 的单边加速确实可提高服务端到客户端的传输效果,提高客户服务满意度,从而带来好的经济效益,但当连接数增加时,系统的稳定性和利用率会下降。因此,本文根据 FAST TCP 单边加速应用特点,提出了一种 2 层结构的传输层改进算法。下层算法仍采用 FAST 算法,但需小时间尺度周期记录各连接最大排队延时等历史信息;上层算法利用下层各连接提供的相关历史信息,大时间尺度周期下协同合作,自适应调整协议参数,通知给下层活跃的 FAST 连接。上述改进算法分别在各路径传播延时不同、瓶颈链路缓冲队列发生变化、各路径随机建立 FAST 连接的不同仿真环境下进行了仿真实验。仿真结果表明,该改进算法能够主动适应环境变化,动态调整协议参数,主动控制瓶颈链路中缓冲队列长度,大幅提高系统的稳定性和瓶颈链路带宽的利用率,这有助于高速网络的应用和 FAST TCP 单边加速设备的推广。

参考文献

[1] 梁伟,张顺颐,宁向延,徐苏磊. 基于稳定性的 FAST TCP 参数 γ 调整. 通信学报,2010,31(7):53-59

[2] David X W, Cheng J, Low S H. FAST TCP: motivation, architecture, algorithms, performance. *IEEE/ACM Transactions on Networking*, 2006, 14(6):1246-1259

[3] Choi Y J, Ko J W, Wook S. Improved global stability conditions of the tuning parameter in FAST TCP. *IEEE Communications Letters*, 2009, 13(3):202-205

[4] 陈晓龙,章云. 非线性时变时滞 FAST TCP 系统低保守全局稳定性. 控制理论与应用,2012,29(4):477-485

[5] 黄小猛,林闯. 高速传输协议研究进展. 计算机学报,2006,29(11):111-120

[6] Gu., Grossman R L. SABUL: A transport protocol for grid computing. *Journal of Grid Computing*, 2003,1(4): 377-386

[7] Tang A, Wei D, Low S H. Heterogeneous congestion control: efficiency, fairness and design. In: Proceeding of the 14th IEEE International Conference on Network Protocols, St. Barbara, USA, 2006. 127-136

[8] 朱小松. 解决 FAST TCP 缓存溢出相关问题的改进 pacing technique 算法和参数调整算法. 信息通信技术,2012,15(2):60-65

[9] Zhang H Y, Zhang X Y. Adaptive FAST TCP. In: Proceeding of the 2nd International Conference on Future Networks, Sanya, China, 2010. 114-119

[10] 宋丽华,陈鸣,张睿. 一种基于测量的 TCP Fast 改进方案. 北京邮电大学学报,2005, 28(4):27-32

[11] 宋丽华,王海涛,陈鸣. 基于网络测量和模糊控制技术的拥塞控制机制. 华南理工大学学报,2006, 34(6): 89-95

[12] 宋丽华,王海涛. 基于性能服务的高速网络运输层拥塞控制解决方案. 解放军理工大学学报(自然科学版),2012,13(3):259-265

[13] Choi J Y, Koo K, Low S H. Global exponential Stability of FAST TCP. In: Proceedings of the 45th IEEE Conference on Decision and Control, San Diego, USA, 2006. 639-643

[14] Kyungmo Koo, Joon-Young Choi, Jin S. Lee. Two different models of FAST TCP and their stable and efficient modification. *Network Control and Optimization*, 2007, 44(1): 65-73

[15] 陈晶,郑明春. 一种基于历史链接的网络拥塞控制算法及其性能分析. 计算机研究与发展,2003,40(10): 1470-1475

An improved unilateral acceleration FAST TCP algorithm based on history connection information

Chen Xiaolong, Peng Zhiping

(School of Computer and Electronics Information, Guangdong University of Petrochemical Technology, Maoming 525000)

Abstracts

Aiming at the open problem that it is difficult for FAST TCP to choose proper protocol parameters, a novel improved two-layer algorithm for the transport layer is proposed according to the application characteristics of unilateral accelerated FAST TCP systems in the paper. It is described below. In the lower layer, the history information of the active FAST TCP flows, such as the maximum queue delay, is recorded at the small time scale period. By making full use of the history information provided by the lower layer, the range of target queuing delay is obtained and the proper protocol parameters of the FAST TCP flows are dynamically tuned by the upper layer algorithm in the large time scale. So the queuing length of the bottleneck link is actively controlled and the queue overflow problem is solved. It is proved by the NS-2 simulation results that the stability and utilization can be improved while this improved two-layer algorithm is used.

Key words: FAST TCP, protocol parameter, queuing delay, history connection, active control