基于多宿SCTP的提高整体吞吐量的快速SACK策略

吕重霖 朱鸣华
(大连理工大学电子信息与电气工程学部 大连 116024)

摘 要 分析了在多宿SCTP的条件下,链路双方传输数据的几种算法,并基于多宿SCTP提出了一种从最快可抵达的路径发送SACK报文的策略,它能够消除只有发送端了解链路情况的弊端。仿真实验表明,在两条路径时延不均衡情况下,快速SACK策略可以极大地提高整体的吞吐量。

关键词 多宿,流控制传输协议,多路径同时传输,快速SACK
中国法分类号 TP393.0 **文献标识码** A

Fast SACK Scheme for Overall Throughput Enhancement Based on Multi-homed SCTP

LV Zhong-lin ZHU Ming-hua
(Faculty of Electronic Information and Electrical Engineering, Dalian University of Technology, Dalian 116024, China)

Abstract This paper analyzed several algorithms when transmitting data in multi-homed scenarios and proposed a fast SACK scheme based on multi-homed SCTP. The scheme eliminate abuse that only the sender grasps the link status. Simulation results show that the proposed scheme could improve the throughout performance of multi-homed SCTP hosts when the delay of two paths is not balanced.

Keywords Multi-homing, SCTP, CMT, Fast SACK

1 引言

流媒体控制传输协议(Stream Control Transmission Protocol,SCTP)是IETF于2000年10月制定的传输层协议标准。作为下一代的传输层协议,SCTP集合TCP、UDP的优点,克服了TCP固有的对头阻塞、Dos攻击,另外新增了许多其他功能,比如多穴、多流、Cookie安全机制等。RFC4960[1]是RFC中关于SCTP协议的最新规定,此规定默认仅用主路径进行数据传输,只有当主路径失效时,才更换到其他路径继续传输[2,3]。很明显,这种数据传输的方式没有充分利用SCTP的多穴特性,致使数据的传输效率不高。为了利用好多条路径,可以尝试在多条路径下进行传输[4]。实验表明,SCTP-CMT的传输性能明显优于传统TCP[5],其能够更充分利用网络资源。

Iyengar等人在多路径同时传输方面做了大量的研究和探索[6,7],并解决了算法在新条件下的不兼容情况:

1)发送端没有必要的快速重传问题:按照RFC4960规定,每当接收端收到非连续数据块,就立即发送SACK报文,通知发送端数据块可能已经丢失。发送端如果连续接收到4次同一数据块的SACK报文,就判定此数据块已经丢失,并立即重传此数据块。这在单路径传输的情况下是没问题的,但是在多路径同时传输的情况下,由于每条路径的时延不同,有可能导致数据块到达的顺序不同,而并非数据已经丢失,这样发送端的快速重传无疑加大了网络的拥塞状况。分裂重传算法(SFR)对每条路径上的丢失报文分别计数,当连续收到4个针对本路径的丢失报文时,才重传此丢失报文。

2)拥塞窗口更新太慢的问题:由于接收端接收到大量的非连续数据,导致累计sack大量减少;而按照以往的拥塞窗口扩大算法,只有接收到累计的sack信号,才增加拥塞窗口,这导致了拥塞窗口更新太慢。新的拥塞窗口更新算法(CUC)针对每条路径分别记录累积ACK,当发现某一条路径上的累积ACK增加时,立即扩大拥塞窗口。

3)大量SACK导致网络拥塞的问题:每当接收端收到非连续数据块,就立即发送SACK信号。在多路径条件下,由于路径的时延不同,导致大量的非连续数据块产生,从而大量的没有必要的SACK信号在网络下堆积,进而增加发送端没有必要的快速重传数据,导致整体性能的下降。延迟应答算法(DAC)总是延迟SACK的发送,并附带发送累积的非连续数据块个数。

通过上述论述可以看出,SFR算法和CUC算法完全没有修改接收端,DAC算法只对接收端做了很少的修改。这些方法研究的重点是对发送端的改进,使得发送端了解大量的路径信息数据,而接收端却不了解。文献[8]则提出了一种通过增加新的报文格式提高整体SCTP吞吐量的快速SACK算法。因此有必要改进接收端算法以提高两个节点协同工作的能力。

RFC4960并未规定SACK的发送路径,当接收端接收到DATA数据后,默认按原路径回送SACK,而这条路径的时延情况未知。如果时延比较大,则SACK需要较长时间到达发送端。因此在路径的时延相差很大的情况下,改进SACK的发送策略,可以提高数据传输的整体性能。

2 快速SACK算法

2.1 算法说明

按照当前SCTP的设计,总是发送端占主动地位去探测路径的时延以及带宽,并决定从哪条路径进行下一个报文的数据

传输(或以前某一个报文的重新传输)。当接收端需要发送SACK数据时,默认使用最后一次收到数据包的路径。接收端由于不了解哪条路径是最优的发送路径,导致SACK的发送是盲目的。如果SACK使用时延比较大的路径发送,将会严重影响网络的整体性能。

如果接收端了解当前的几条路径的时延情况,则可以选择时延最小的路径发送SACK,这样发送端就可以得到最及时的报文确认,从而提高整体吞吐量。本文提出了一种思路,即可以在对SCTP协议修改较小的情况下,准确地判断出当前最小的路径时延。

本文论述中所用到的DATA和SACK报文格式及说明如图1和图2所示。

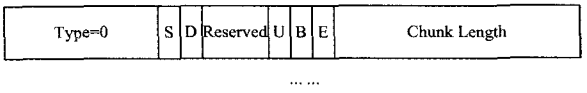

图1 修改后的DATA报文格式

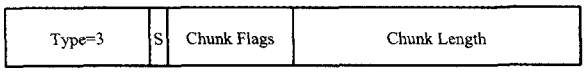

图2 修改后的SACK报文格式

将DATA报文中原保留位(Reserved)的第一个高字节位定义为S,$S=0$并且$D=0$代表普通DATA报文;$S=1$并且$D=0$代表特殊DATA报文;$S=0$并且$D=1$代表延误报文。将SACK报文中原Chunk Flags的第一个高位字节位也定义为S,$S=0$代表普通DATA报文,$S=1$代表特殊SACK报文。

2.2 算法描述及实现

图3给出了基于多宿SCTP的一种提高整体吞吐量的快速SACK策略的算法描述。

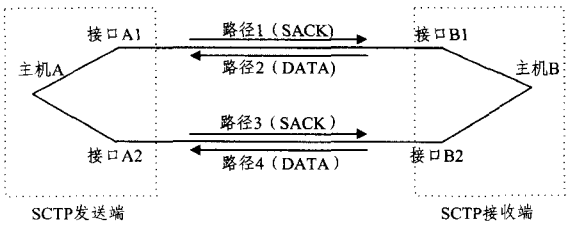

图3 快速SACK策略算法描述图

算法的实现步骤如下:

(1)收集数据

首先接收端选择发送端口列表的第一个端口作为发送路径,发送特殊SACK数据块,并记录这条发送路径(SendPath)和发送时间(SendTime);

发送端收到这个特殊SACK后,如果允许立刻发送DATA数据块,则将第一块要发送的数据块添加上特殊标记,并发送出去;如果不能立即发送DATA数据块,那么等到可以发送数据的时候,将第一块数据块添加上延误标记,再发送出去。

若接收端收到特殊DATA数据块,立即记录接收路径(ReceivePath)和接收时间(ReceiveTime),将时间间隔ReceiveTime-SendTime保存到相应的数据库中(即对应填入下表3)。到此为止,一次最快SACK路径探测完成,即一次收集数据完成。

若接收端收到延误DATA数据块,则表明这次收集数据失败。表1汇总了算法收集的数据,其中,-1是默认初始值,表明两个主机之间的关联刚刚建立。

表1 接收端数据库

	接收路径2	接收路径4
发送路径1	-1	-1
发送路径3	-1	-1

(2)比较发送路径的时延

按照接收到特殊DATA报文的时间顺序排序,从最近一次接收到特殊DATA报文的接收路径开始。若已经收集完针对此接收路径的所有发送路径的时间间隔(即对应此接收路径所有列的值),那么就可以通过这些时间间隔的大小来比较所有发送路径的时延大小,将时延最小的发送路径作为主SACK发送路径。若没有收集完全,那么继续对下一个接收到特殊DATA报文的接收路径进行比较。

(3)更新数据

每当接收端收到特殊DATA数据块或延误DATA数据块后,立即从发送端口列表中选择下一个端口作为发送端口,继续发送特殊SACK数据块,从而及时更新数据表中的数据。

在两次最快SACK传输路径的探测之间使用主SACK路径发送普通SACK报文,这样既可以保证数据表中数据的实时性,又可以较快地发送普通SACK报文。

3 仿真实验结果及分析

本文使用ns-2[9]作为仿真实验环境(以下实验统一使用图4所示的拓扑图)。

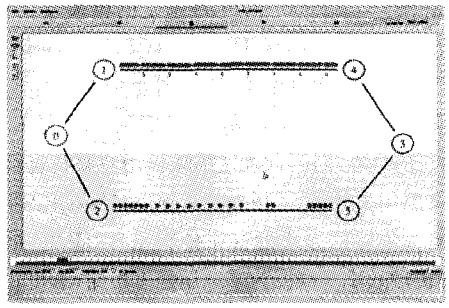

图4 仿真实验拓扑图

说明:0节点表示主机,1、2节点代表0节点的两个网络接口,节点3、4、5同理;

节点1到节点4之间路径规定为路径1,带宽2M,往返时延都为50ms;

节点2到节点5之间路径规定为路径2,带宽2M,往返时延都为250ms;

主机0往主机3共发送8M数据。

3.1 算法正确性的验证

当接收端(主机3)接收到DATA数据后,如果需要发送SACK数据包,就会从同一路径回复。

如图5所示,在未使用快速SACK算法的情况下,接口4由于收到的DATA数据包比较多,发送SACK的机会也就更多。这样就可能导致时延小的路径空闲,而时延大的路径拥塞,严重影响整个网络的吞吐量。

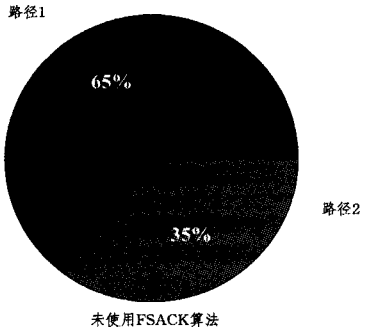

图 5　SACK 数据包比例图(未使用快速 SACK 算法)

如图 6 所示,在使用快速 SACK 算法的情况下,一旦发现路径 1 的时延比较小,就可以优先选择路径 1 发送 SACK,这样发送端就可以在最短的时间里接收到 SACK,并发送后面的 DATA 数据包,从而提高发送端的响应速度。

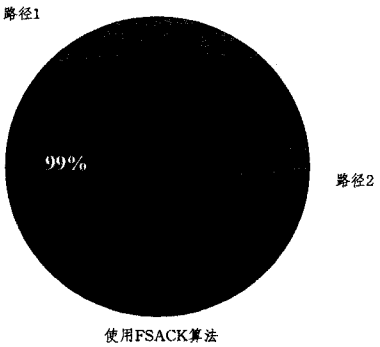

图 6　SACK 数据包比例图(使用快速 SACK 算法)

表 2 显示了在使用快速 SACK 算法的情况下,特殊 SACK 和普通 SACK 的比例。

表 2　SACK 比例表

	路径 1	路径 2
普通 SACK	2819	2
特殊 SACK	41	27

由于需要及时地更新路径的时延,因此系统一旦更新了路径的时延,就立即从刚接收的 DATA 数据的路径回复 SACK,进行下一次时延的更新。

由表 2 知,路径 2(时延大)只在刚开始两条路径的时延都未知的情况下,发送了 2 个普通 SACK 数据包,其它的 27 个特殊 SACK 都是为了更新路径时延(表 1 数据的更新)。

3.2　性能比较

图 7 显示了在其他参数不变,路径 2 时延从 50ms 到 500ms 变化的情况下,数据传输所需时间的统计结果。

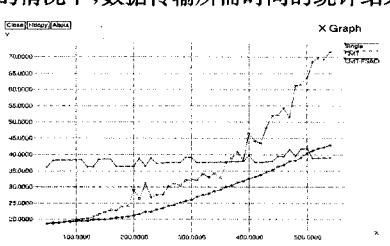

图 7　系统吞吐量统计图

其中,X 轴:路径 2 的时延,单位为毫秒(ms);Y 轴:传输 8M 数据所需的时间,单位为秒(s);红线显示的是只使用路径 1(时延为 50ms)进行单路径数据传输的情况;绿线显示的是未使用快速 SACK 算法进行多路径数据传输的情况;蓝线显示的是使用快速 SACK 算法进行多路径数据传输的情况。

由图 7 可知,在两条路径时延相差不大的情况下,使用默认的多路径同时传输算法能极大地提高整体的吞吐量;但是在两条路径时延相差 8 倍以及 8 倍以上的情况下,默认的多路径同时传输反而不如只在单路径下进行传输的情况,其主要的原因在于默认的 CMT 算法只对发送端进行了改进,并未对接收端进行改进。

如图 7 所示,使用快速 SACK 算法后,整体性能得到大幅度提升,且在两条路径时延相差小于 10 倍的情况下,其吞吐量高于只用单路径传输的吞吐量。

3.3　仿真实验分析

图 8 显示了在路径 2 时延为 500ms 的条件下,3 种不同情况的统计结果。

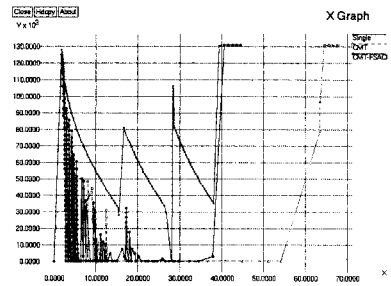

图 8　接收端接收窗口统计图

其中,X 轴:时间,单位毫秒(ms);Y 轴:接收端的接收窗口 rwnd 的大小。

由图 8 可以看出,默认的多路径同时传输的情况下,在 12～55ms时间内出现接收端缓冲区阻塞的问题[10]。这是由于发送端在路径 2 上发送的数据需要较长时间才能到接收端,而路径 1 虽然空闲,但仍不能发送数据,这样导致多路径同时传输的性能严重下降,其甚至小于单路径传输。

图 9 显示的是在 24.981888 时刻发送端的发送窗口,其中灰色底纹为已经接收到的数据块,白色底纹为仍未接收到的数据库。显然此时已经出现头端阻塞,发送端急需接收到端头报文的确认,SACK 报文的传送速度影响着整体的性能,而此时接收端已经通过快速 SACK 算法检测出路径 1 的时延比较小,优先选择路径 1 发送 SACK 报文。

图 9　发送窗口

结束语　本文在 SFR 算法、CUC 算法和 DAC 算法的基础上,提出了一种接收端可以感知链路时延状况的算法,消除了以往只有发送端了解链路情况的弊端。在快速 SACK 策略下,接收端可以选择最快的一条链路发送 SACK 报文,这样在两条路径时延不均衡时,新策略可以极大地提高性能。

在真实情况下,链路两端都要发送数据,接收端在发送 SACK 报文的同时也在发送 DATA 报文。接收端的感知链路时延状况的算法有待改进。

参 考 文 献

[1]　Stewart R, et al. Stream Control Transmission Protocol [S]. IETF RFC4960,2007

(下转第 122 页)

运算，各节点参与运算的概率是相同的。但是由分析可知，在进行均值运算时，每个节点对操作的贡献度是不一样的。那些与周围节点时钟差值较大的节点贡献更大。同时，在移动环境中扩散算法会优先在本地实现时钟同步，那些运动速度较快的节点有更大机会离开本地区域到达较远的区域，因此其与新的邻居节点具有较大的差值的机会更多。为了进一步提高收敛速度、降低通信成本，一种可行的方法是使不同的节点具有不同的概率参与到均值运算中来。

4.1 改进算法 1

为了提高收敛速度，可以让不同的节点以不同的概率来进行均值运算。第一种方法是使节点参与运算的概率与此节点在上次均值计算中时钟变化的绝对值成正比例。即 $p_i \propto |H_{i-1}-H_{i-2}|$。这些节点具有比邻居节点更大的时钟差值。通过增加其参与运算的概率可以提高收敛速度，这可以从图 5 中看出。图 5 表示了改进前后的同步效果，横坐标是总操作次数。

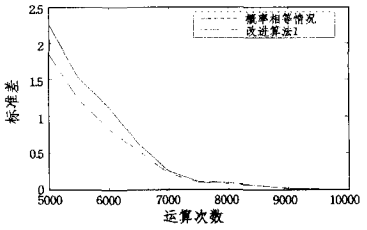

图 5 改进算法 1 的效果

4.2 改进算法 2

在节点移动的环境下，由于距离较近的节点比较容易实现局部同步，而较远的节点有更大可能具有较大的时钟差值，因此可以使节点进行均值运算的概率与节点速度成正比。仿真结果如图 6 所示，改进算法可以提高时钟收敛速度。

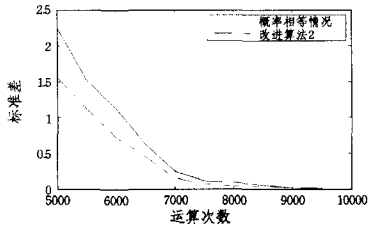

图 6 改进算法 2 的效果

4.3 改进算法分析

在移动无线传感器网络中，通过调整节点参与均值运算的概率，可以在网络总计算量不变的前提下，提高网络的同步速度。两种方法的本质都是使与网络时钟均值有较大偏差的节点更多地参加计算；同时节点对自身上次时钟变化值和自身移动速度的测定都是本地完成的，不需要节点间的通信，因此其附加能量消耗很低。改进方法在性能和效率间取得了较好的平衡。

结束语 时钟同步是分布式系统中非常重要的问题，在无线传感器网络中要同时考虑同步精度和能量效率两方面。在节点移动的情况下，这一问题更加复杂。本节对移动无线传感器网络中的时钟同步问题进行了介绍。着重介绍了一种基于扩散的同步算法，分析了其在移动环境下的性能，说明了移动性可以提高此种算法的同步速度，并进一步提出了两种加快时钟同步速度的方法。

参 考 文 献

[1] kyildiz I F, Su W, Sankarasubramaniam Y, et al. A survey on sensor networks [J]. IEEE Communications Magazine, 2002, 40 (8): 102-105

[2] 王波，叶晓慧，赵玉亭，等. 自组织网络时钟同步研究综述[J]. 计算机科学，2010，37(5)：30-33

[3] Hoffmann W, Lichtenegger B, Collins H, et al. GPS theory and practice [J]. Geophysicaet Montanistica Hungarica, 1995, 30 (1): 107

[4] Jeremy E, Lewis G, Deborah E. Fine-grained network time synchronization using reference broadcasts [C]// 5th Symposium on Operating Systems Design and Implementation. Boston, Massachusetts, USA: 2002: 147-163

[5] Ganeriwal S, Kumar R, Srivastava M B. Timing-sync protocol for sensor networks [C]// 1st International Conference on Embedded Networked Sensor Systems. Los Angeles, CA, USA, 2003: 138-149

[6] Sivrikaya F, Yener B. Time synchronization in sensor networks: A survey [J]. IEEE Network, 2004, 18(4): 45-50

[7] Li Q, Rus D. Global clock synchronization in sensor networks [J]. IEEE Transactions on Computers, 2006, 55(2): 214-226

(上接第 119 页)

[2] 夏云，孙力娟，叶晓国，等. SCTP 协议分析与仿真研究[J]. 计算机技术与发展，2009，19(11)：27-30

[3] 孙楠，张载龙，孙雁飞. SCTP 性能的仿真分析[J]. 计算机技术与发展，2010，20(12)：103-106

[4] 朱桂勇，吴庆波. 基于 SCTP 多宿特点的多路径同时传输研究[J]. 计算机技术与发展，2007，17(3)：5-9

[5] 鄢欢，高德云，宋飞. 基于 SCTP 多路径并行传输的性能评估[J]. 计算机技术与发展，2010，20(11)：29-32

[6] Iyengar J, Amer P, Stewart R. Concurrent multipath transfer using SCTP multihoming over independent end-to-end paths[J]. IEEE/ACM, 2006, 14(5): 951-964

[7] Iyengar J, Amer P, Stewart R. Retransmission policies for concurrent multipath transfer using SCTP multihoming[J]. IEEE ICON, 2004, 2(16): 713-719

[8] Cui Lin, Koh S J, Lee W J. Fast Selective ACK Scheme for Throughput Enhancement of Multi-Homed SCTP Hosts[J]. IEEE, 2001, 14(6): 587-589

[9] [EB/OL]. http://www.isi.edu/nsnam/ns/

[10] Iyengar J, Amer P, Stewart R. Receive Buffer Blocking In Concurrent Multipath Transfer [J]. IEEE, 2005, 1(28): 121-126