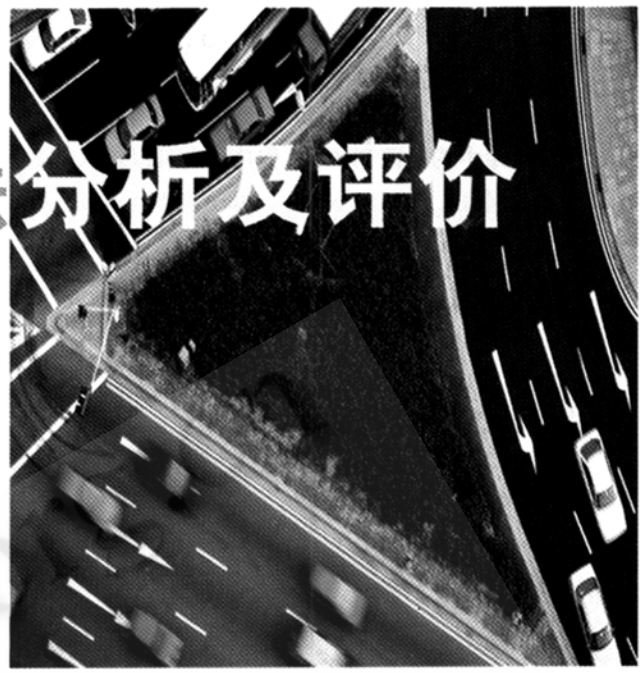
TCP 拥塞控制解决方法分析及评价



刘秋让 倪红波
(西安西北工业大学网络信息中心 710072)

摘要：在这篇文章中，我们探讨了网络拥塞出现的原因及其一般解决方法，并主要对Tcp的拥塞控制做了较深入的研究，阐述了在Tcp网络中处理拥塞的几种算法。通过它们原理的分析和性能的比较，指出了在目前的网络实现中可以优先采用的拥塞控制算法。

关键词：Tcp协议 拥塞控制 Tahoe算法 Reno算法 Sack (selective acknowledge)算法

当前和今后网络技术的目标就是在综合服务的网络上满足服务质量。目前在互联网应用最广泛的传输控制协议(Tcp,Transmission Control Protocol)，它可以用于3W服务(World Wide Web)，文件传输协议(Ftp,File Transmission Protocol)，远程登陆(Telnet)。Tcp能支持可靠传送用户数据，并且可以控制拥塞。但是由于所要传送的信息数据量很大，就需要网络的流量管理(Traffic Managerment)，其主要目标是控制网络拥塞，有效利用网络资源，并为用户提供协商的服务质量。这里我们阐述网络拥塞控制方面的问题，主要讨论出现拥塞的原因及其解决方法。

Tcp解决拥塞是采用基于窗口的(window-based)端到端(end-to-end)的算法，以前主要有两种：Tahoe和Reno。分别对通过重复应答所判断出的丢包现象进行不同的处理。后来随着网络技术的发展，在Reno算法的基础上，出现了New-Reno算法，下面就这几种算法做一概述。

1 TAHOE算法

早期的Tcp实现算法是基于积极响应并通过重传超时来重发丢失的数据，当丢包时，Tcp减小拥塞窗口，并重传被丢失的分组，然而在使网络拥塞达到最小方面，这些算法的性能很差。Tahoe Tcp参考了早期的实现方法，并增加一些算法，包括慢启动(Slow-Start)：窗口大小以指数速度增加；拥塞避免(Congestion Avoidance)：窗口的大小以一定的线性速度增加；快速重传(Fast Retransmit)：从一个丢包状态恢复而不需要等待重传定时器超时。Tahoe还包括对往返时间估计量的修改，这一参量的准确估计非常重要，因为它被用来设置重传超时定时器的基值。此外Tahoe引入快速重传机制，即当接受方收到几个对同一tcp报文的相同应答时，发送方就推断已经发生了丢包，而没有必要等到重传定时器超时，并且重传相应的包。这样以来提高了信道利用率。

2 RENO算法

Reno修改Tahoe的快速重传为快速恢复(指由三个重复应答判断有包丢失时，仅使拥塞窗口减半)，新的算法防止通信管道在快速重传之后变空，因而避免了慢启动在单包丢失之后重填。快速重传主要决定于收到的重复应答数目的初始门限值(一般设为三)。一旦达到门限值，发送方就重传一个包，同时使拥塞窗口减半，与Tahoe的慢启动不同，Reno的发送方用额外到达的应答来为后续包定时。发送方的可用窗口变为Min(Awin,Cwnd+Ndup)这里Awin是接受方的广播窗口，Cwnd是发送方的拥塞窗口，Ndup保持零值(在重复的应答的数目达到门限值之前)，因而在快速恢复中，发送方根据收到的重复应答来变动自己的窗口。相应的，每个重复应答表示有一些包已被移出网络并且现在已经到达接受方。在进入快速恢复并重传一个包之后，发送方就开始等待，直到一半窗口大小的重复应答被收到时，然后对应每个额外重复的应答传出一个新包，当接受到新数据的应答时，发送方退出快速恢复并设置Ndup=0。

Reno的理想情况是在一个窗口中单包丢弃时，Reno发送方在每个往返时间(Rtt)中最多重传一个包。但是它

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn

在同一窗口中出现多包丢弃时可能出现问题。

3 NEW RENO 算法

这里探讨对Reno的一些改进，New Reno做了一个变化，即当多包丢弃时，去掉了Reno的等待重传定时器。在快速恢复中，当发送方收到一个部分应答来表征一些包而不是所有包，在这个阶段(即 Fast Recovery)的起始时间没有被成功发送。在Reno中，部分包通过减少可用窗口至拥塞窗口的大小以使TCP退出快速恢复。在New Reno中，部分应答不能使TCP退出快速恢复，反之，部分应答表示跟随它的所应答的包已经丢失，需要重传。New Reno的恢复不需要重传超时，每个往返时间重传一个包直到所有的丢包全被传完。New Reno保持在快速恢复状态，直到在快速恢复阶段初始化时未成功传送的数据全被响应。

前面这几种算法，在单包丢弃时，效果是不错的，但如果在同一数据窗口中出现多包丢弃的情况下，它们的性能都有较大的局限性。后来出现了基于选择应答(Sack,Selective Acknowledge)的算法，它较好的解决了同一数据窗口中多包丢弃的问题，这种算法的基本原理是这样的：

Sack 算法中，有一个称作选择域(Option)的数据段，Sack的选择域包含许多Sack块，它们报告一组已经到达和排序的非拥塞的数据，第一块需要报告接受方最近接受的数据，其他块重复最新报告过的数据块。一般来说，每一个Sack选择域包含三个Sack块。当sack选择域和高性能tcp扩展处理的时间戳一起使用时，它仅有三个Sack块；当Sack选择域同时和时间戳(Timestamp)与Tcp扩展处理T/TCP(Tcp Extentions for Transactions)一起使用时，选择域空间仅有两个 sack 块。

在这里的Sack Tcp拥塞控制算法，是对Reno算法的保守扩展，因此它们用同样的增加和减少拥塞窗口的算法，并且对其他的拥塞算法做最小改动。Sack Tcp保存Tahoe和Reno算法在出现乱包时的健壮性，并用重传超时作为重排序的恢复方法。Sack 和Reno的最大不同在于一个数据窗出现多包丢弃。

如同在Reno中，Sack算法当接收方收到门限值(Tcprexmtthresh)个重复的应答时，发送方进入快速恢复阶段，它这时重传一个包，并把拥塞窗口减半，在快速恢复阶段，Sack维护一个叫做管道(Pipe)的变量，它表示估计的在该键路尚未成功发送的包数目(这是和Reno不同的一点)。当估计的包数目小于拥塞窗口的大小时，发送方仅仅发送新包或重传；在发送方发送一个新包或者重传一个旧包时，管道变量就加一，反之，当收到一个报告新数据已被接受的应答时减一。

管道变量分离了何时和哪一个包要发送，发送方维护一个叫做分数板的数据结构，它记忆以前的Sack选择项的应答。当发送方允许发送一个包时，它从包列表中重传下一个包，这些包用来推测在接收方已经丢失；如果没有这样的包并且接收方的广播窗口是足够大，发送方发送新包。当重传的包自己丢失时，那么Sack算法就用重传超时来检测丢包，重传丢失的包然后进入慢启动。

如果发送方收到这样一个应答帧，它用来通告所有在进入快速恢复时没有完成的数据已经完成发送，那么，发送方退出快速恢复阶段。发送方对部分应答(是在快速恢复阶段收到的，但不把发送方带出快速恢复)进行特殊的处理。对部分应答来说，发送方每次减少两个包而不是一个包。当快速恢复初始化时，管道为每个假定丢失的包减一，而为每一个重传的包增加一。因而当接收到第一个部分应答时就把管道减少两个包，从某种意义上说，有些欺骗的意味：因为部分包仅仅代表一个包离开了管道，然而对后继的部分包而言，当重传包进入时管道要增加，但对那些被假定丢失的包管道从不减少。因而当后继的部分包到达时，事实上是两个包离开了管道：原始包(假定已经丢失的包)和重传包。因为发送方为部分包减少二而不是一，所以Sack发送方不可能比慢启动更慢。如同前面所说的，最大爆发量(Maxburst)这个变量限制了要发送的用来响应单个应答包的包数目，避免了爆发流量(Burst Traffic),Maxburst是实验性质的，不必要在实现中应用。

在这几种算法的仿真试验中[4]，如果是单包丢失，Tahoe需要从慢启动中恢复，而其他的三种算法都可以比较平滑的从快速恢复从丢包中恢复。换句话说，正如前边所讨论的，在丢失一个包时，这几个算法的差别不是很明显的，尤其是New-Reno和Sack几乎是相同的。但如果是多包丢失(比如四个包)，Tahoe经过慢启动从丢包中恢复，Reno出现严重的性能问题，必须等待重传定时器超时从丢包中恢复，而New-Reno需要四个往返时间来恢复和重传四个丢失的包，最优的是Sack-Tcp，它的恢复比较迅速，平滑。总之，在同一数据窗口中多包丢弃时，其他算法和Sack的差别很明显，使我们足以相信Sack是较优的选择。■

©《计算机系统应用》编辑部 http://www.c-s-a.org.cn