AIMD 算法公平性仿真验证与分析

何士钊

（电子科技大学信息与通信工程学院， 四川 成都 611731）

摘 要：本文第一部分引入了网络拥塞的概念，并指出了导致网络拥塞的三种直接原因。接着介绍了拥塞控制算法公平性的意义及评价指标。在第二部分，本文介绍了 TCP 所使用的拥塞算法的控制模型，列举了所涉及到的设计参数，并详细介绍了目前被广泛使用的 Reno 拥塞控制算法的四个阶段：慢启动、拥塞避免、快重传和快恢复。在第三部分，借助 NS3，仿真并模拟了两种场景下的 AIMD 算法的性能。通过对数据的分析，本文认为 AIMD 算法有较好的公平性。

关键词：网络拥塞；TCP；AIMD 算法；公平性；NS3

1 引言

1.1 网络拥塞的概念

拥塞崩溃最先在 1980 年提出，当网络中存的数据包超过了网络的处理能力时，会使得网络的性能下降，这种现象称为拥塞。在网络发生拥塞时，会导致吞吐量减小，如果用户仍不断的提出对网络资源的需求,更增加了网络的负担,严重时会发生“拥塞崩溃” (Congestion Collapse)现象[1]。

1.2 网络拥塞的产生原因

网络产生拥塞的根本原因在于端系统（或用户）提供给网络的负载大于网络资源容量和处理能力，具体表现为：数据包延时增大、丢包概率增大、上层应用系统性能下降。网络发生拥塞的直接原因主要有以下三点[2]:

1) 存储空间不足。当几个输入数据流需要同一个输入端口, 在这个端口就会建立排队队列，等待传输的数据包要经过长时间排队后才能通过路由器完成转发过程导致网络拥塞。
2) 带宽不足。低速链路对高速数据流的输入也会产生拥塞现象。在网络低速链路处会形成带宽瓶颈，当其满足不了所有信源带宽要求时，网络就会发生拥塞[2]。
3) 处理器能力弱、速度慢。如果路由器 CPU 的包处理速度跟不上高速链路，就会产生拥塞现象。同样，低速链路对高速 CPU 也会产生拥塞[2]。

综合上述三点直接原因，网络中拥塞现象发生的根本原因是：端系统的流量需求超出了现有的网络资源。但互联网是一个分布控制系统，无法控制用户对可用网络资源的需求。因此，需要拥塞控制算法实现为用户更合理地分配网络资源。

1.3 拥塞控制算法的公平性

由于拥塞控制算法对整个网络系统都有影响，因此在评估算法时应该从整个系统的角度出发进行考虑。其中一个非常重要的问题就是资源分配的公平性问题。

公平性是指在网络发生拥塞时各连接能公平地共享网络资源。网络拥塞的发生必然导致数据的丢失，进而导致各数据流之间争抢有限的网络资源。竞争能力强的数据流将得到更多网络资源，但这损害了其他流的利益，造成网络资源分配的不公平，并加重拥塞。因此拥塞算法的公平性是一个重要的评价指标。在本文中，对公平性的分析采用的是 Jain 公平指标计算公式：

$$F = \frac{\left(\sum_{i=1}^{n} \overline{x_i}\right)^2}{n\sum_{i=1}^{n} \overline{x_i}^2} \tag{0.1}$$

其中，F 表示公平性，n 代表用户的数量，$\overline{x_i}$ 代表用户 i 的平均吞吐量。F 取值在 0 到 1 之间，F 越大说明算法的公平性越好。

2 AIMD算法

TCP 是目前 Internet 上使用最为广泛的一种传输层协议。它为 Internet 提供了可靠的端到端通信。TCP 拥塞控制的目的是为了解决 Internet 的稳定性、异质性（接收端缓冲区大小、网络带宽及延迟等）、各数据流之间享用带宽的公平性以及网络资源的使用效率等问题。TCP 拥塞控制是确保网络性能的关键因素，也是各种网络管理控制机制和应用的基础[3]。

2.1 TCP拥塞控制的基本方式

从控制理论的角度来看，拥塞控制的方法可以分为两类：开环和闭环控制。当流量特征可以准确规定、性能要求可以事先获得时，适合开环控制；而当流量特征不能准确描述或者当系统不提供资源预留时，适合使用闭环控制[4]。对于 Internet 这样不断变化的复杂系统，显然应当选择闭环控制。

通常情况下，闭环的拥塞控制分为以下 3 个阶段[5]：

1) 检测网络中拥塞现象的发生；
2) 将拥塞信息报告到拥塞控制点；
3) 拥塞控制点根据拥塞信息进行调整以消除拥塞。

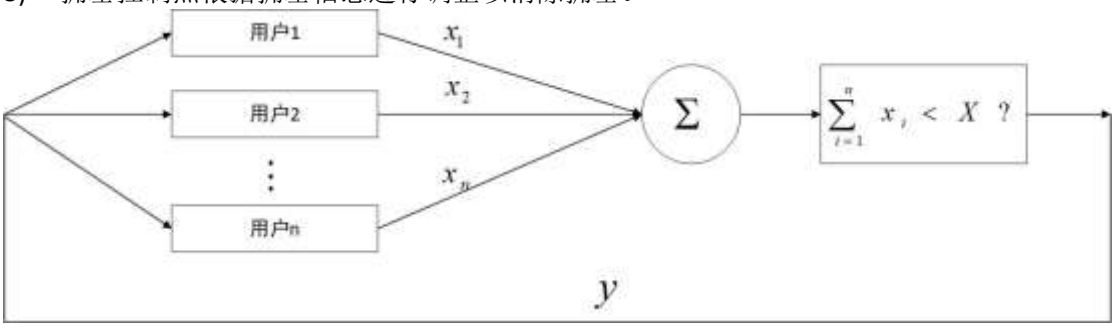

图 1 n 个用户共享同一网络时闭环拥塞控制模型

闭环拥塞控制的模型如图 1 所示。其中 n 表示网络中用户的数量；x_i 表示在时刻 t，第 i 个用户的传输负载；$\sum_{i=1}^{n} x_i$ 表示进入网络的总负载；X 表示网络的承受能力；y 表示系统反馈函数。如果总负载不大于网络的承受能力，则表示所有用户的请求都会被完全接受。此时，x_i 也同时表示每个用户实际分配到的网络资源。在应用中，系统会通过反馈控制函数 $y(t)$ 实时地改变用户传送负载的大小。令对第 i 个用户的改变量为 $u_i(t)$，则该用户的传送负载变为 $x_i(t+\Delta t) = x_i(t) + u_i(t)$ [6]。

基本的 TCP 拥塞控制算法都是通过控制一些重要参数的改变实现的，用于拥塞控制的参数主要有:

1) 拥塞窗口(*cwnd*):拥塞控制的关键参数，描述了发送端在拥塞控制情况下一次最多能发送数据包的数量。
2) 通告窗口(*awin*):是接收端给发送端预设的发送窗口大小。
3) 发送窗口(*win*):是发送端每次实际发送数据的窗口大小。
4) 慢启动阈值(*ssthresh*):是拥塞控制中慢启动阶段和拥塞避免阶段的分界点。
5) 往返响应时间(*RTT*): 从发送端发送一个数据包开始，到发送端收到接收端确认该数据包的时间间隔。
6) 重传定时器(*RTO*):描述数据包从发送到失效的时间间隔。

2.2 AIMD算法的四个阶段

目前应用最广泛的 TCP 拥塞控制算法是 Reno 算法，该算法的核心是 AIMD 算法。AIMD 主要由四个阶段（算法）组成：慢启动、拥塞避免、快重传、快速恢复。

2.2.1 慢启动阶段

TCP 应用在启动一个连接时会向网络中发送大量的数据分组，这样做容易导致网络发生拥塞和 TCP 连接的吞吐量急剧下降。由于 TCP 源端无法知道网络当前的资源利用状况和拥塞程度，因此新建立的 TCP 连接不能一开始就发送大量数据，而应逐步增加每次发送的数据量，以避免上述现象的发生[7]。

在慢启动阶段，TCP 源端按照拥塞窗口大小发送数据，每收到一个 *ACK* 确认，拥塞窗口就增加一个数据分组发送量。拥塞窗口将随着往返时延(*RTT*)呈指数增长，源端向网络发送的数据量将急剧增加。为了防止拥塞窗口急速增长引起网络拥塞，还需要设定 *ssthresh*。在慢启动期间，当 *cwnd* 超过 *ssthresh* 或者检测到拥塞时(如果 TCP 源端发现超时或者收到 3 个相同 *ACK* 时，即认为网络发生了拥塞)，将停止执行慢启动算法，转而进入拥塞避免阶段。

2.2.2 拥塞避免阶段

在此阶段，*cwnd* 将采用线性增长（AI）的方式。在拥塞避免期间，源端在收到所有期待的非重复 ACK 之后 *cwnd* 将增加一个数据分组发送量。如果在该阶段，源端再次判定发生了拥塞，则会将 *ssthresh* 设置为当前 *cwnd* 一半，这一操作即乘法减小（MD）；此外，还会将 *cwnd* 的大小设置成 1 个数据分组发送量。

2.2.3 快重传和快速恢复阶段

当接收端收到一个失序的数据报时，会立即发回一个重复的 *ACK*，从而告知发送端接收端发生失序及失序的报文序号。如果连续收到 3 个相同的 *ACK*，则很大程度上说明报文已经在网络中丢失。为了减少接收端等待时间及避免不必要的网络负担，此时源端将会直接重传对应报文，而不用关心是否超时。这一动作就叫做快重传。此时，源端会：

1) 将 *ssthresh* 的大小设置为当前 *cwnd* 大小的一半；
2) 将 *cwnd* 的大小设置为 *ssthresh* 的大小（或设置为 *ssthresh*+3）；
3) 进入拥塞避免阶段。

根据 *cwnd* 加性增加（AI）、乘法减小的性质，我们可以得到同一网络中的多个 TCP 流的 *cwnd* 会遵从同比例减少、同量增加的规律。依据这一规律，从理论层面分析可以得到 AIMD 算法会使多个流最终收敛到公平点，从而保证算法的公平性。

3 仿真验证与结果分析

NS2(Network Simulator，version 3)是由 UC Berkeley 开发而成一种面向对象的网络仿真器，本质上是一个离散事件模拟器。NS3 有一个虚拟时钟，所有的仿真都由离散事件驱动的。目前 NS3 可以用于仿真各种不同的 IP 网络。NS3 中已经实现了 TCP 所使用的 Reno 算法，所以我们直接 NS3 进行仿真。

3.1 仿真模型

本文采用如图 2 所示的网络拓扑进行仿真。该网络拓扑中，A 与 C、B 与 D 分别构成 2 个 TCP 流，记为 TCP1 和 TCP2。其中，A 与 B 是源端，C 与 D 是宿端。E 与 F 是两个路由器，它们之间的 P2P 链路构成该拓扑的主干网瓶颈链路，记为 L0。L1-L4 则为其他主机与路由器相连的接入链路。

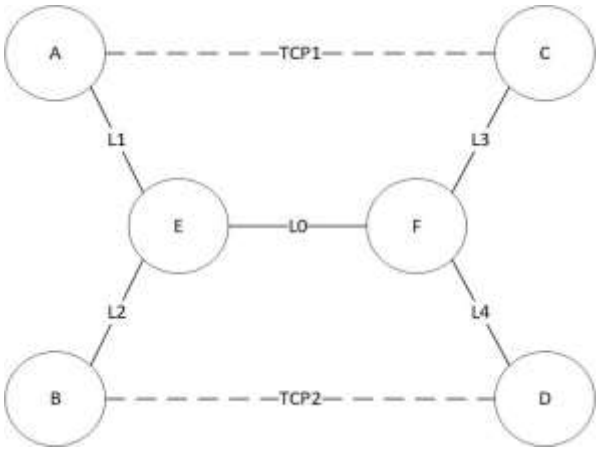

图 2　仿真网络拓扑图

在本文中仿真了 2 个场景，分别是：TCP1 与 TCP2 的流量之和小于瓶颈链路 L0 的传输容量；TCP1 与 TCP2 的流量之和大于瓶颈链路 L0 的传输容量。仿真中，先启动 TCP1 的传输，30ns 之后启动 TCP2 的传输。具体的带宽设置如表 1 所示：

表 1 仿真参数设置

场景	L0 带宽（Mbps）	L1 带宽（Mbps）	L2 带宽（Mbps）	L3 带宽（Mbps）	L4 带宽（Mbps）	A 发送速率（Mbps）	B 发送速率（Mbps）	仿真时间（ns）
1	10	3	3	3	3	3	3	80
2	5	3	3	3	3	3	3	80

Reno 算法中的其他参数均为 NS3 中的默认值。

3.2 仿真数据

根据得到的仿真数据，本文分别绘制了节点A和B的*cwnd*变化趋势以及吞吐量变化趋势。并在仿真过程中，对节点F连接节点E的接口进行抓包，利用wireshark的统计工具，得到了IO Graph。具体图像如下图所示：

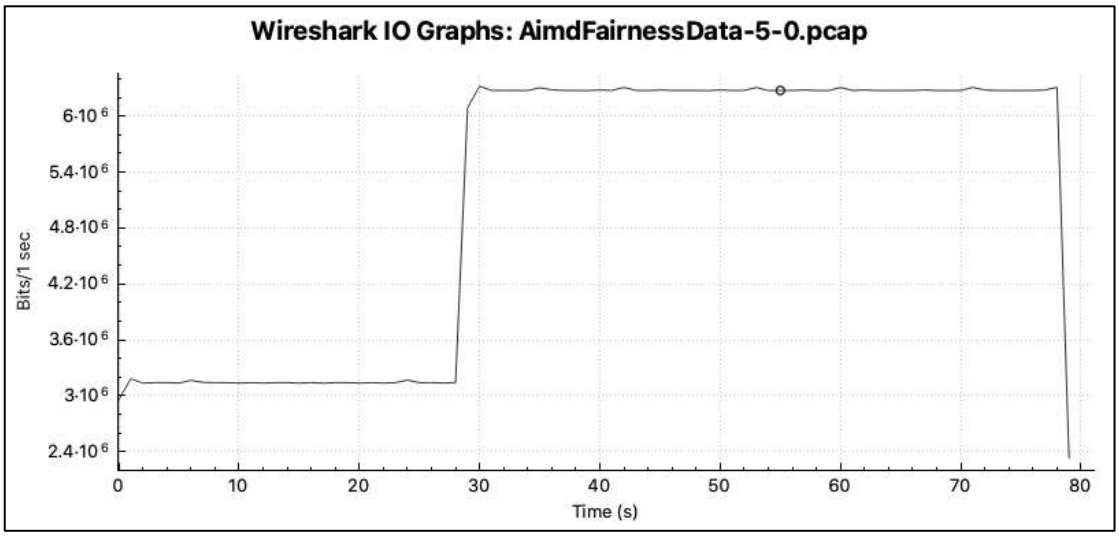

图 3　场景 1 的 IO Graph

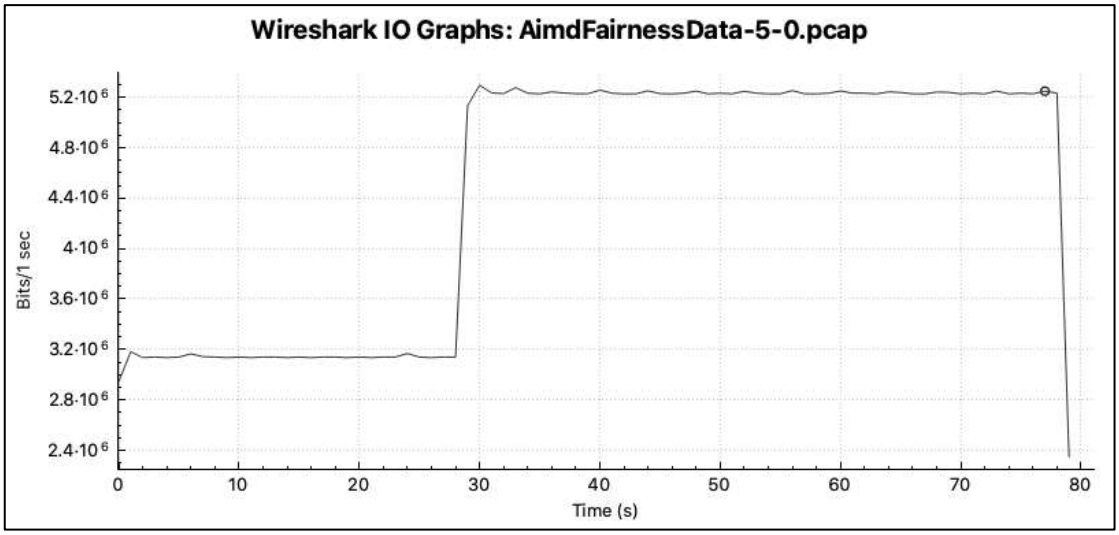

图 4　场景 2 的 IO Graph

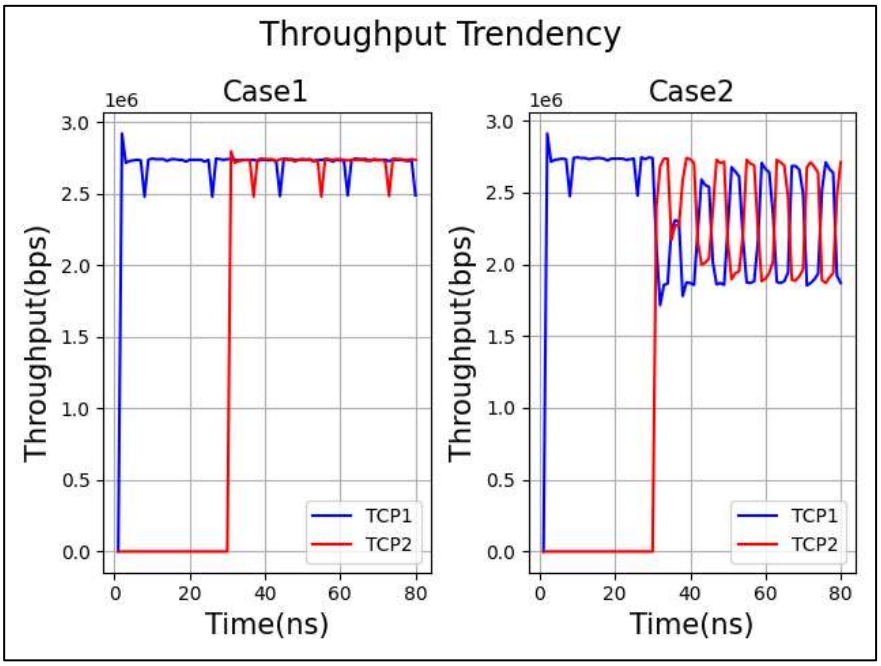

图 5　吞吐量变化趋势

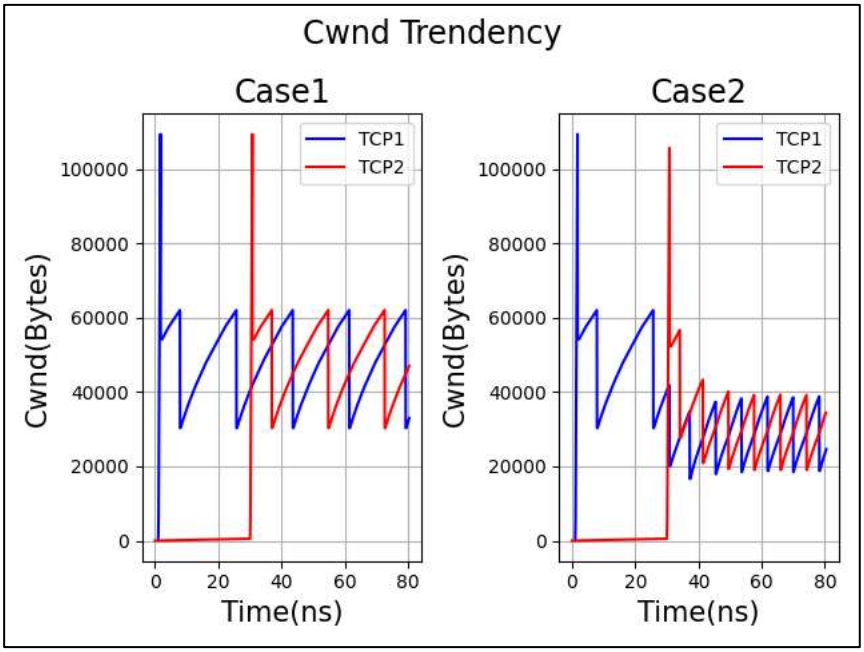

图 6　*cwnd* 变化趋势

3.3　结果分析

从图 3 和图 4 中可以看出，在平稳状态时，第一种场景下的瞬时流量是 2 个流的瞬时流量之和；第二种场景下的瞬时流量则是瓶颈链路的最大传输速率。根据这 2 幅图，结合 1.2 可以判断：场景 1 没有发生网络拥塞，场景 2 发生了网络拥塞。

根据得到的数据，可以分别计算出 2 个场景中 2 个流 30-80ns 内各自平均吞吐量；再利用（0.1）可以计算出公平性指标 F 。结果如表 2 所示：

表 2　平均吞吐量及公平性指标

场景	TCP1（Mbps）	TCP2（Mbps）	F
1	2.72	2.20	0.999
2	2.72	2.33	0.999

可以看出，在两种场景下，AIMD 算法都具有很好的公平性。这一点从图 5 中也可以看出：2 个场景下，2 个流的吞吐量的总体趋势都是相同的，只是“步调”不一致。同时，也可以看到，当 2 个流的流量之和大于瓶颈链路传输容量时，会发生更频繁、明显的抖动。

从图 6 可以看到：在场景 1 下，在 TCP2 启动前后，TCP1 的 *cwnd* 的变化规律保持一致，且 2 个流的变化趋势稳定后相同；在场景 2 下，TCP2 启动后，TCP1 的 *cwnd* 变化曲线的峰值减小为原来的一半左右，且 2 个流的变化趋势稳定后相同。从 2 个流最终变化趋势相同，也可以验证 AIMD 算法具有较好的公平性。

4　结束语

随着 Internet 网络规模的不断扩大，网络中设备的不断增加，网络拥塞这一问题将会在未来变得更有挑战性。这也意味着对应用最广泛的可靠传输协议 TCP 性能提出了更苛刻的要求。基于此，本文分析并仿真模拟了 TCP 中最经典的拥塞控制算法 AIMD 算法。从理论分

析和最后的仿真结果，都可以得到 Reno 算法中的 AIMD 算法具有较好公平性。

参考文献：

[1] 付忠.TCP拥塞控制算法性能研究[D].南京：南京邮电大学, 2001：1.

[2] 付忠.TCP拥塞控制算法性能研究[D].南京：南京邮电大学, 2001：2.

[3] 付忠.TCP拥塞控制算法性能研究[D].南京：南京邮电大学, 2001：6.

[4] 付忠.TCP拥塞控制算法性能研究[D].南京：南京邮电大学, 2001：6.

[5] 付忠.TCP拥塞控制算法性能研究[D].南京：南京邮电大学, 2001：6.

[6] 付忠.TCP拥塞控制算法性能研究[D].南京：南京邮电大学, 2001：7.

[7] 付忠.TCP拥塞控制算法性能研究[D].南京：南京邮电大学, 2001：10.