TCP 网络拥塞预防与控制算法仿真

姓名：程海鹏　学号：SA06011086

一、　实验目的

仿真 TCP 协议中用于拥塞控制的四种算法—慢开始（slow start）、拥塞避免(congestion avoidance)、快速重传(fast retransmit)和快速恢复(fast recovery)。比较快速重传和快速恢复（改进后的 TCP）对于慢开始和拥塞避免（传统的 TCP）的改进效果。

二、　实验方法

通过 OPNET 软件模拟 internet 中一个 FTP 应用上的 TCP 传输，在不同的网络丢包率和应用不同的 TCP 拥塞算法情况下，分别得到 FTP 服务器端拥塞窗口数据。从而，进一步根据实验数据比较快速重传和快速恢复（改进后的 TCP）对于慢开始和拥塞避免（传统的 TCP）的改进效果。

三、　实验内容一

在网络丢包率为 0％的网络中只使用 slow star 和 congestion avoidance TCP 拥塞控制算法，得到 FTP 服务器端拥塞窗口数据，并分析拥塞窗口数据变化曲线。

3.1、实验设置

3.1.1、网络拓扑

分别在 Paris 和 Stockholm 部署一个 FTP server、Router 和一个 FTP Client、Router。Paris 和 Stockholm 两地的 Router 都接入 European internet。

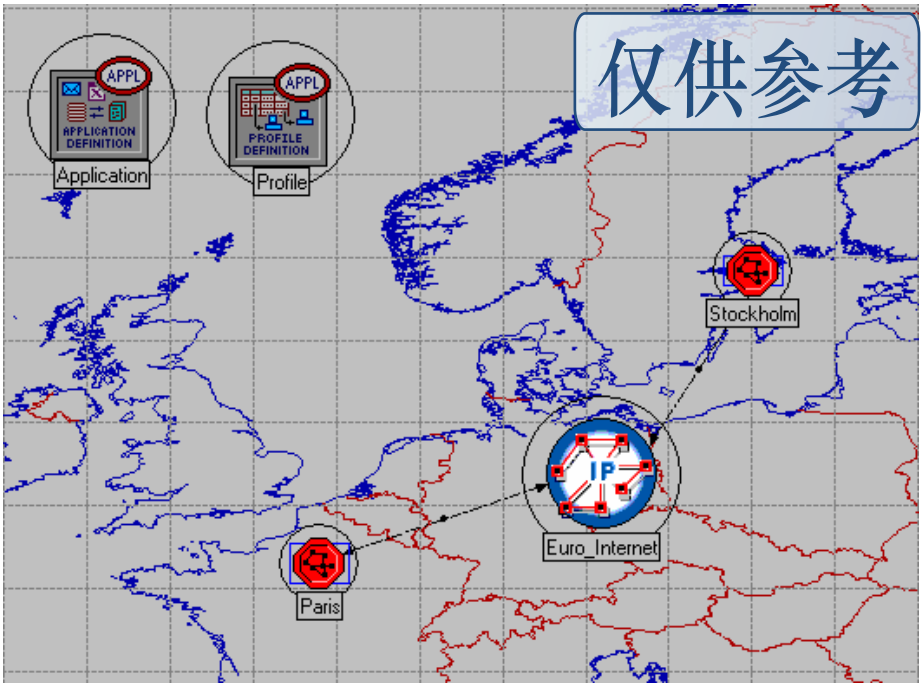

图 1 通过 Internet 的一个从 Paris 到 Stockholm 的 FTP 应用

在 Paris 的 FTP server、router 和在 Stockholm 的 FTP server、router 拓扑机构如图 2、图 3 所示。

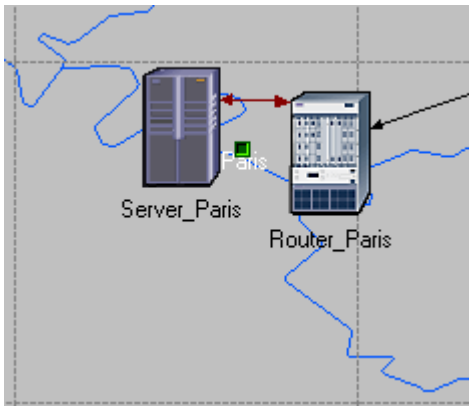

图 2 Paris 的 FTP server、router

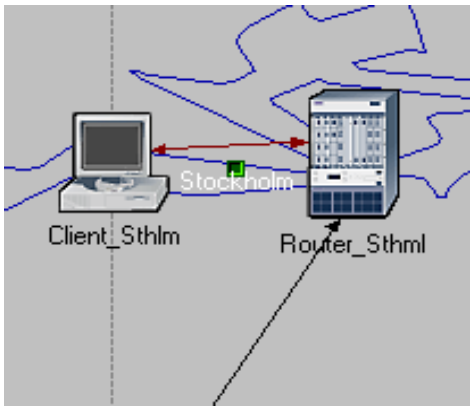

图 3Stockholm 的 FTP server、router

3.1.2、参数设置

[1]FTP 终端和 router 通过 100Base-T 连接，router 到 internet 则通过 PPP-DS3 连接；

[2]应用类型为 FTP（File Size=18M, ToS=Best Effort）；

[3]Client 设置为请求 FTP application，repeatability 设置为“once at start time”；

[4]IP clouds 丢包率为 0％，FTP server 端 TCP 不使用 Fast retransmit 和 Fast recovery。

[5]模拟网络工作时间设置为 5 minutes。

3.1.3、统计数据

[1]FTP 服务器端 TCP 连接的 congestion widow size(Bytes)。

3.2、实验数据

图 4，图 5 分别给出了统计数据的实验结果和 slow star threshold size 点。

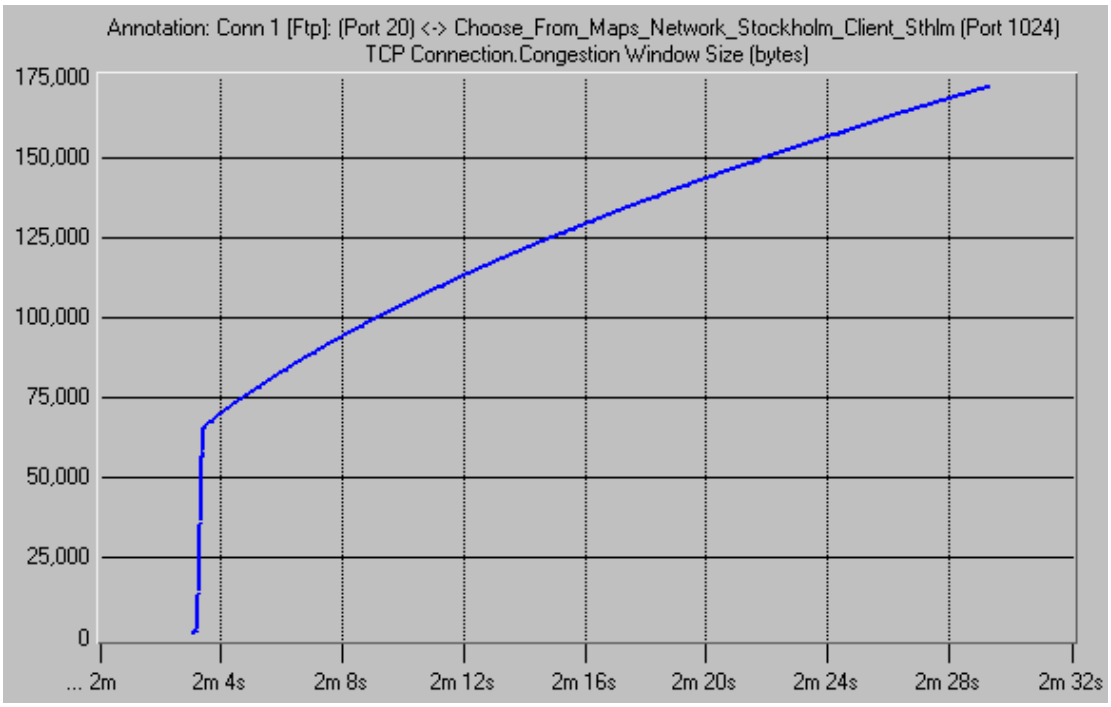

图 4 FTP Server TCP connect. Congestion window size (bytes)

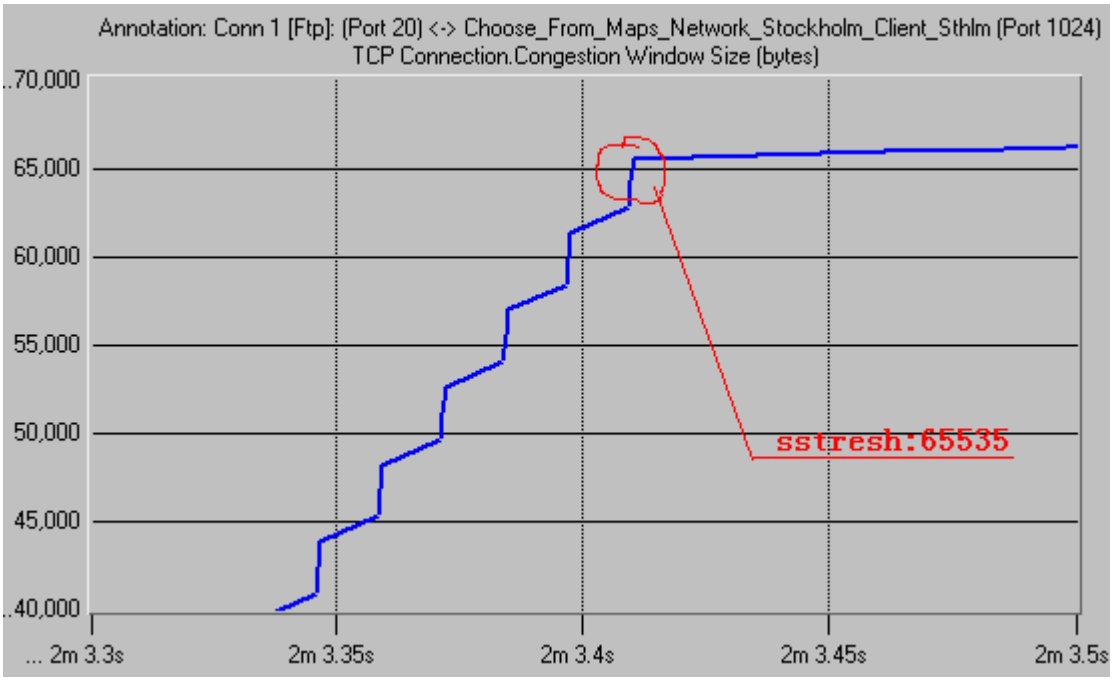

图 5 the value of the slow star threshold size

3. 3、实验数据分析

[1]sstresh 之前 TCP 连接的 cwnd 平均增加速度为 65535/0.4 = 260K (bytes/sec)。

[2]sstresh 之后 TCP 连接的 cwnd 平均增长数据为 106489/26 = 3.9K (bytes/sec)。

3. 4、问题讨论

为了避免拥塞崩溃，对于每个 TCP 连接，TCP 都记录 receiver 的通报窗口大小。同时，为了控制拥塞现象，TCP 使用另一个限制值－congestion window。

Slow start : allowed_window = min (receiver_advertisement, congestion_window)

当一个新的连接建立后，拥塞窗口只被初始化为一个报文段大小（一般为 512 字节）。随着 sender 每收到一个从 receiver 返回的 ACK，拥塞窗口的大小呈指数级增长（$N = 2^x$）。当 Router 的容量达到饱和时，Router 便会开始丢包。网络开始丢包则是说明拥塞窗口过大，需要降低其大小，因为降低 congestion window 的值就会降低 TCP 注入该连接的流量。

拥塞避免或拥塞预防，是一种在拥塞变的严重之前预防网络发生丢包的拥塞窗口变化策略。从 sender 的角度来看，以下两种情况意味着包被丢失：[1]超时；[2]收到重复 ACK's。在实际应用中，慢启动和拥塞避免通常会一起使用，具体算法操作如下：

Step 1. Initialization: cwnd = bytes of one segment; sstresh = 65535; //cwnd->"Congestion window", sstresh->"slow start threshold size".

Step 2. Sender's allowed_window = min (receiver_advertisement, congestion_window);

Step 3. if (TCP connection is available){

if (congestion){

```
        ssthresh = cwnd/2;
        If (timeout) cwnd=bytes of one segment;
        }
    if (a new data is acknowledged by receiver) {
        If (cwnd <= ssthresh) TCP is in slow start;
        Else TCP is in congestion avoidance; // cwnd += segsize*segsize/cwnd
        }
    }
    goto step 2;
```

四、 实验内容二

在网络丢包率为 0.5％的网络中使用以上四种 TCP 拥塞控制算法，得到 FTP 服务器端拥塞窗口数据，并分析拥塞窗口数据变化曲线。

4.1、实验设置

4.1.1、网络拓扑

同实验一。

4.1.2、参数设置

[1] IP clouds 丢包率为 0.5％，FTP server 端 TCP 使用 Fast retransmit 和 Fast recovery。

[2]其他参数设置同实验一。

4.1.3、待统计数据

[1]FTP 服务器端 TCP 连接的 congestion widow size(Bytes)。

4.2、实验数据

图 6 中从上至下三部分分别给出网络没有丢包、0.5％丢包率网络中只使用 Fast retransmit 和同时使用 Fast retransmit 和 Fast recovery 算法情况下，FTP server 端 congestion window size 数据。

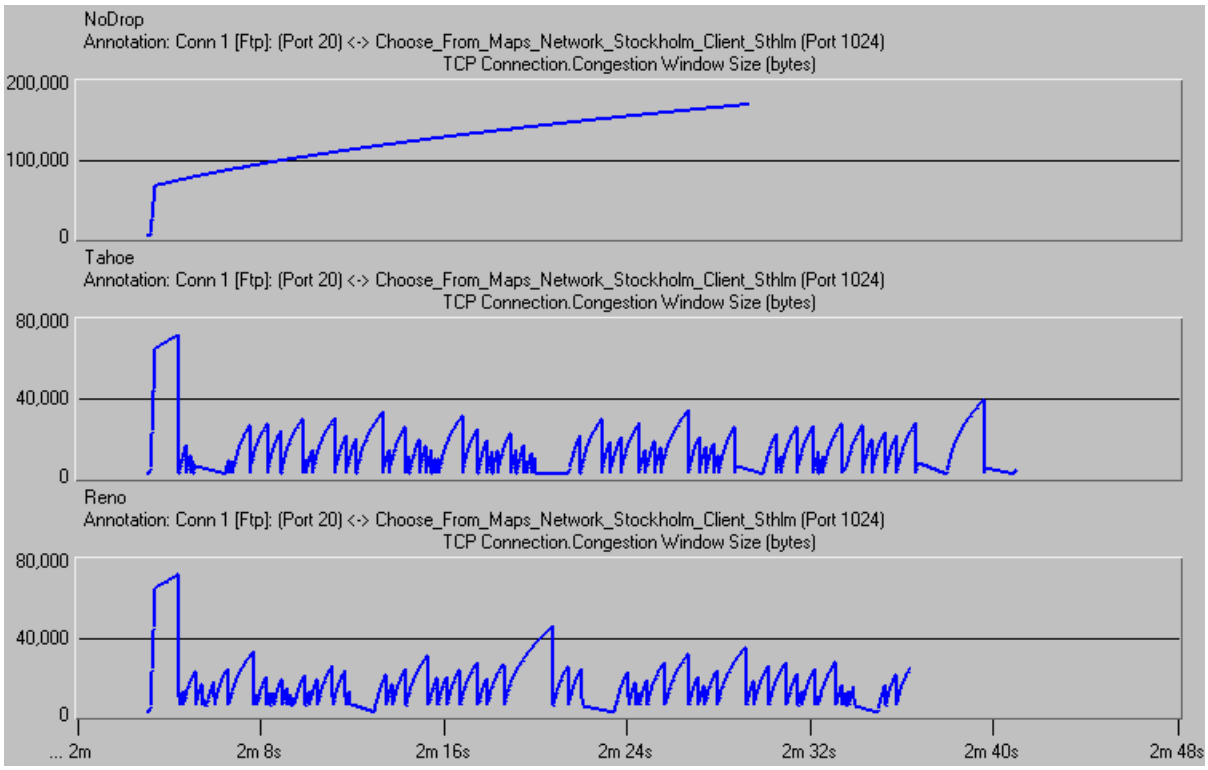

图 6 在 0.5％网络丢包率下单独使用 Fast retransmit 和同时使用 Fast retransmit，Fast recovery 算法 FTP Server 端 cwnd 数据

图 7 和图 8 则分别给出 Fast retransmit 算法，Fast retransmit + Fast recovery 算法检测到 congestion 后 congestion window size 的数据。

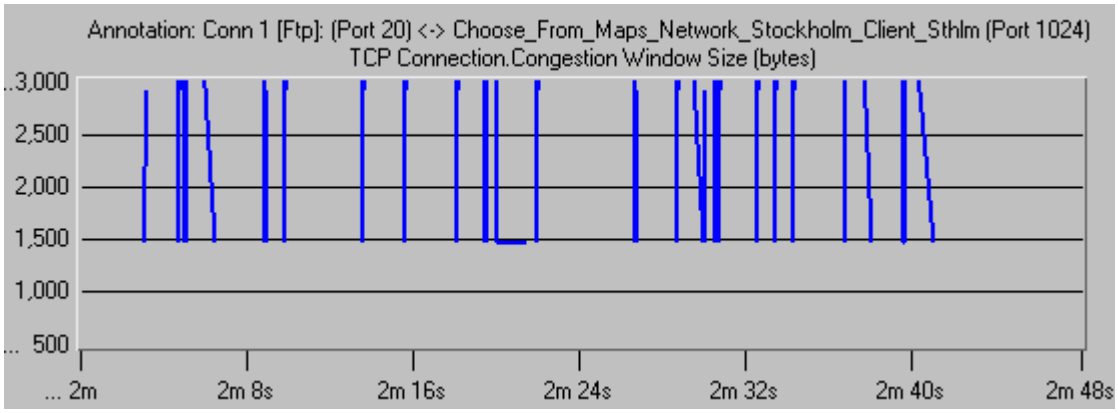

图 7 在 0.5%丢包率网络中，使用 Fast retransmit 算法在网络发生拥塞(timeout)时 cwnd 数据

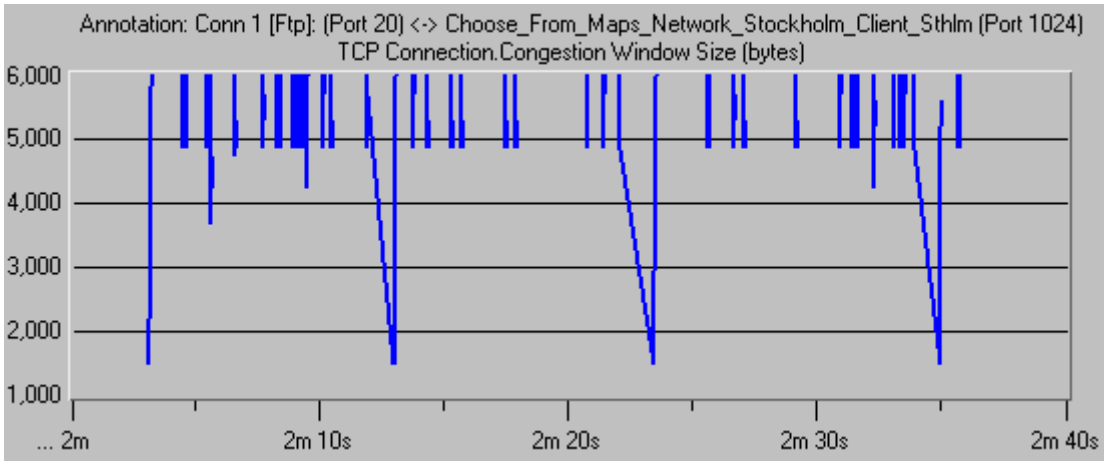

图 8 在 0.5%丢包率网络中，使用 Fast retransmit 和 Fast recovery 算法在网络发生拥塞(timeout)时 cwnd 数据

4.3、实验数据分析

[1]从图 6 数据可以直观的看到：在 0.5％网络丢包率情况下，同时使用 Fast retransmit 和 Fast recovery 算法网络性能最优（即 FTP 传输时间最短）。

[2]比较图 7 和图 8 数据可以看到：只使用 Fast retransmit 算法（图 7）相比较同时使用 Fast retransmit 和 Fast recovery 在遇到拥塞时前者的拥塞窗口小于后者。

4.4、问题讨论

Fast retransmit 是对 congestion avoidance 的改进算法。TCP sender 利用 fast retransmit 根据重复的 ACK’s 来检测和修复丢包，即当 TCP sender 收到 3 个重复 ACK’s 之后(这期间没有其他的 ACK 到达)，就不再等待重传记时器超时而是直接重新发送丢失的报文段。

在 Fast retransmit 之后将执行 Congestion avoidance。但在 congestion avoidance 之后将不再是 slow start（在大窗口，中度拥塞的情况下将较大的提供网络性能），取而代之的是 Fast recovery。Fast retransmit 和 Fast recovery 常常一起使用，具体过程如下：

```
Step 1. if (the third duplicate ACK is received){
            if(cwnd/2>=2) ssthresh = cwnd/2;
            else ssthresh=2;
            retransmit the missing segment;
            cwnd = ssthresh+3*megsize;
        }
Step 2. if (another duplicate ACK arrives) {
            cwnd += megsize;
            if (allowed by cwnd and advertised window) transmit a segment;
            }
Step 3. if (a new ACK arrives){
            ssthresh = cwnd;
            cwnd += megsize*megsize/cwnd;
            }
            goto step1;
```

显然，在网络存在丢包情况时，使用 Fast retransmit 和 Fast recovery 更为有效。

五、 实验内容三

分别在网络丢包率为 0.5%、1％的网络中使用以上四种 TCP 拥塞控制算法，得到 FTP 服务器端拥塞窗口数据，并分析拥塞窗口数据变化曲线。

5.1、实验数据

图 9 由上至下分别给出在网络丢包率为 0.5%情况下，使用 slow star、fast retransmit 以及 fast retransmit + fast recovery 算法，FTP server 端 congestion window size 数据。

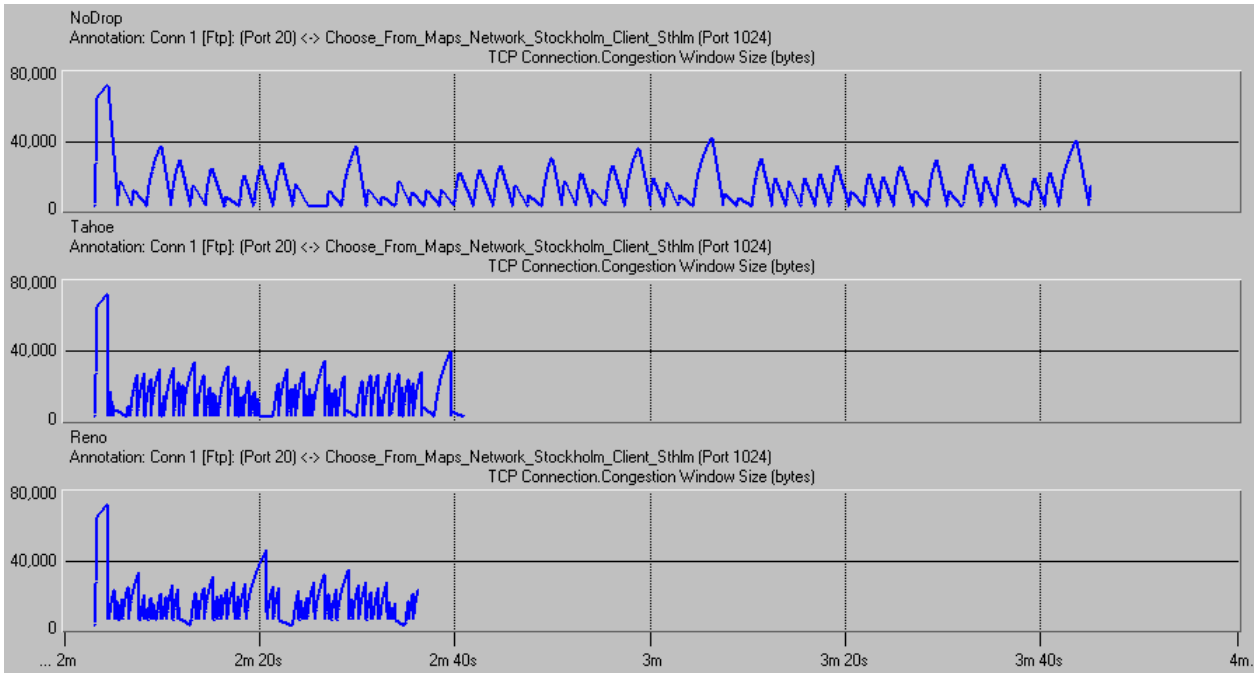

图 9 在 0.5％网络丢包率下使用 slow star、Fast retransmit 和 Fast recovery + Fast recovery 算法 FTP Server 端 cwnd 数据

图 10 给出在网络丢包率为 1%情况下，使用 slow star、fast retransmit 以及 fast retransmit + fast recovery 算法，FTP server 端 congestion window size 数据。

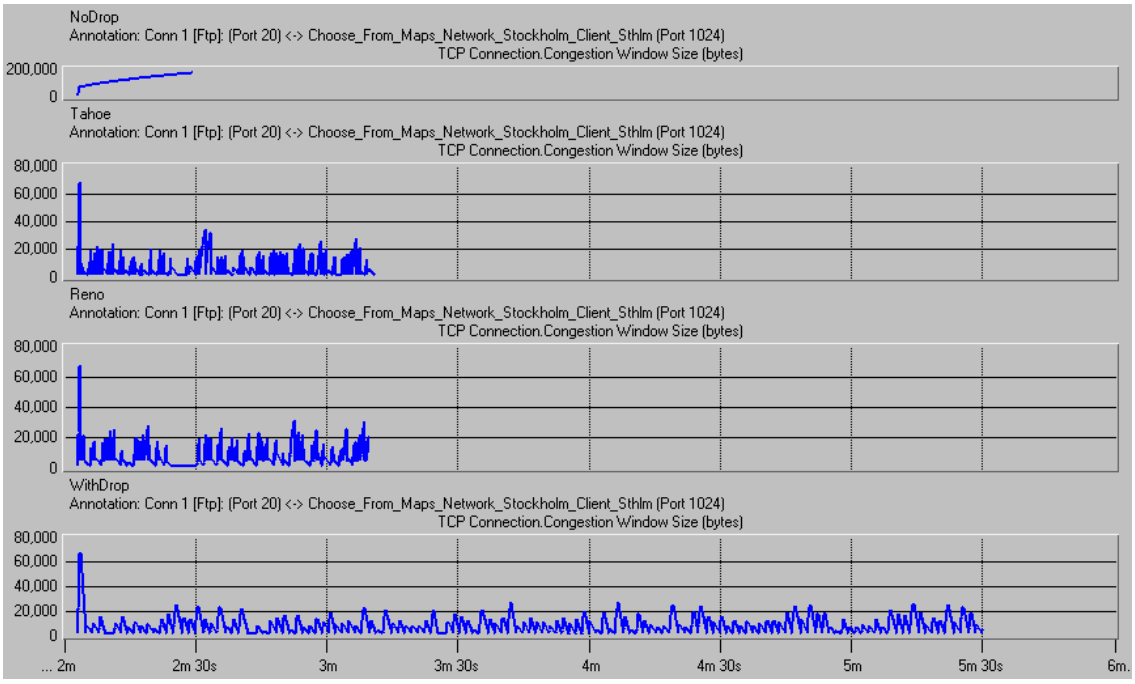

图 10 在 1％网络丢包率下使用 slow star、Fast retransmit 和 Fast recovery + Fast recovery 算法 FTP Server 端 cwnd 数据

图 11、图 12 分别给出 FTP server 端瞬时输出流量和时间平均流量数据。

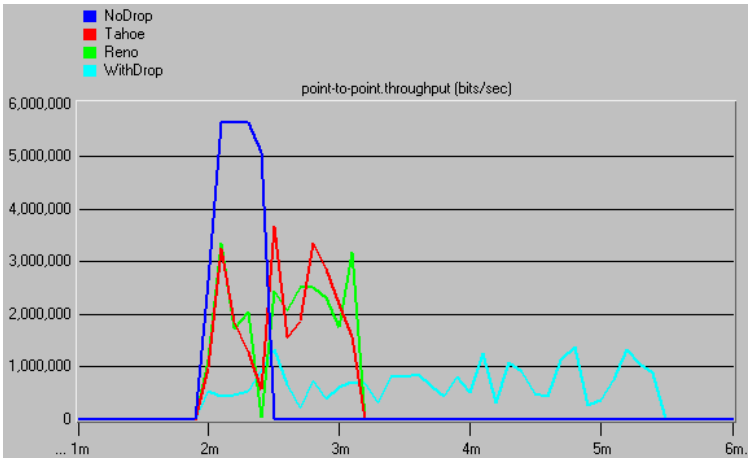

图 11 FTP server 端瞬时输出流量

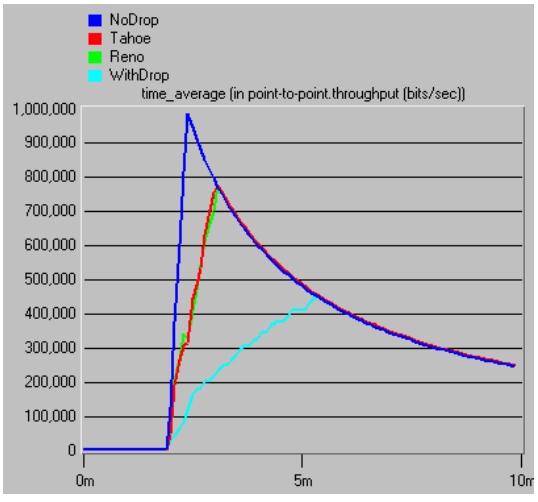

图 12 FTP server 端时间平均输出流量

5.2、实验数据分析

[1]从图 9 数据可以得到：在 0.5％网络丢包率情况下，使用 slow star 和 congestion avoidance 算法的性能最低，改进后的拥塞控制算法（使用 fast retransmit 和 fast recovery）性能明显提高；同时使用 fast retransmit 和 fast

recovery 性能较好于只使用 fast retransmit。

[2]从图 10 数据可以得到：在 1％网络丢包率情况下，改进后的拥塞控制算法性能提高更加明显；而同时使用 fast retransmit 和 fast recovery 性能与使用 fast retransmit 区别很小，但还是略好。

[3]图 11 和图 12 在网络丢包率为 1％时，使用 fast retransmit 使 FTP server 输出流量增大。

5.3、问题讨论

[1]当网络存在丢包时，使用 fast retransmit 改进算法能提高网络性能；使用 fast recovery 可以使网络流量平稳。

[2]随着网络丢包率的提高，使用改进后的 TCP 拥塞控制算法其性能优势更加明显。

[3]在网络丢包率较高时，使用 fast recovery 与 slow star 性能相当。即当 timeout 出现频繁时，fast recovery 恢复传输 cwnd 值较小，接近于 slow star 的一个报文段 cwnd 大小。

六、 结论

[1]网络存在因为拥塞而发生丢包时，使用 fast retransmit 和 fast recovery 算法来改进 TCP 可以提高网络吞吐量同时平稳网络性能。

[2]无线链路上运行 TCP 协议时最好采用改进后的 TCP。这是因为无线链路上的通信受限于窄带宽，高延迟，高误码率以及经常发生的暂时中断，从而所产生的丢包现象就会无必要的导致 TCP 窗口缩小，定时器复位，最终带来长延迟和低吞吐量。此时若采用 fast retransmit 算法就能通过重复的 ACK 来探测包丢失并立刻重传。从而把无线链路的质量对终端隐藏起来，提高网络的吞吐量。