**数据通信NS3作业-5&6**

姓名： 费扬 学号： 519021910917 日期： 2022.05.18

1. **实验名称及内容**

**Lab5 & Lab6：**

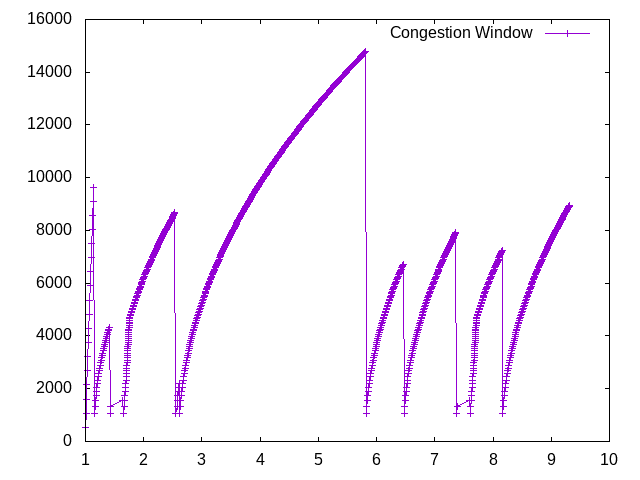
**Background：**

Fifth.cc示例演示了一个极其重要的规则，在使用任何类型的跟踪源之前您必须了解该规则: 即必须确保 Config::Connect 命令的目标存在，然后再尝试使用它, 这与说一个对象在尝试调用它之前必须被实例化没有什么不同。

任何 ns-3 脚本中都存在三个基本执行阶段。

* The first phase is sometimes called “Configuration Time” or “Setup Time,” and exists during the period when the main function of your script is running, but before Simulator::Run is called.
* The second phase is sometimes called “Simulation Time” and exists during the time period when Simulator::Run is actively executing its events.
* After it completes executing the simulation, Simulator::Run will return control back to the main function. When this happens, the script enters what can be called the “Teardown Phase,” which is when the structures and objects created during setup are taken apart and released.

**Demo:**



1. **实验过程和结果**

程序见压缩包内lab5和lab6的目录下。本次ns3的版本为3.30。

#include <fstream>

#include "ns3/core-module.h"

#include "ns3/network-module.h"

#include "ns3/internet-module.h"

#include "ns3/point-to-point-module.h"

#include "ns3/applications-module.h"

#include "ns3/packet-sink.h"

头文件包含内容如上。

**Lab 5:**

**Produce a figure showing TCP congestion window size, dropped packets and received packets, like in the following figure. Use the same network configuration as in mysixth.cc.**

**Sixth.cc is a script that writes the cwnd change and drop events developed in the example fifth.cc to disk in separate files. The cwnd changes are stored as a tab-separated ASCII file and the drop events are stored in a PCAP file.**

**The changes to make this happen are quite small. Actually this can be done in only 18 lines of code.**

**Hints:**

1. **Modify mysixth.cc**
2. **Use trace source**

[ns3::PointToPointNetDevice](https://www.nsnam.org/docs/release/3.28/doxygen/classns3_1_1_point_to_point_net_device.html)

**PhyRxEnd**: Trace source indicating a packet has been completely received by the device

**Simulation：**

**$./waf --run scratch/lab5**

**Check the .cwnd file, and use gnuplot to draw a figure.**

**$ gnuplot:**

*set terminal png size 960, 720*

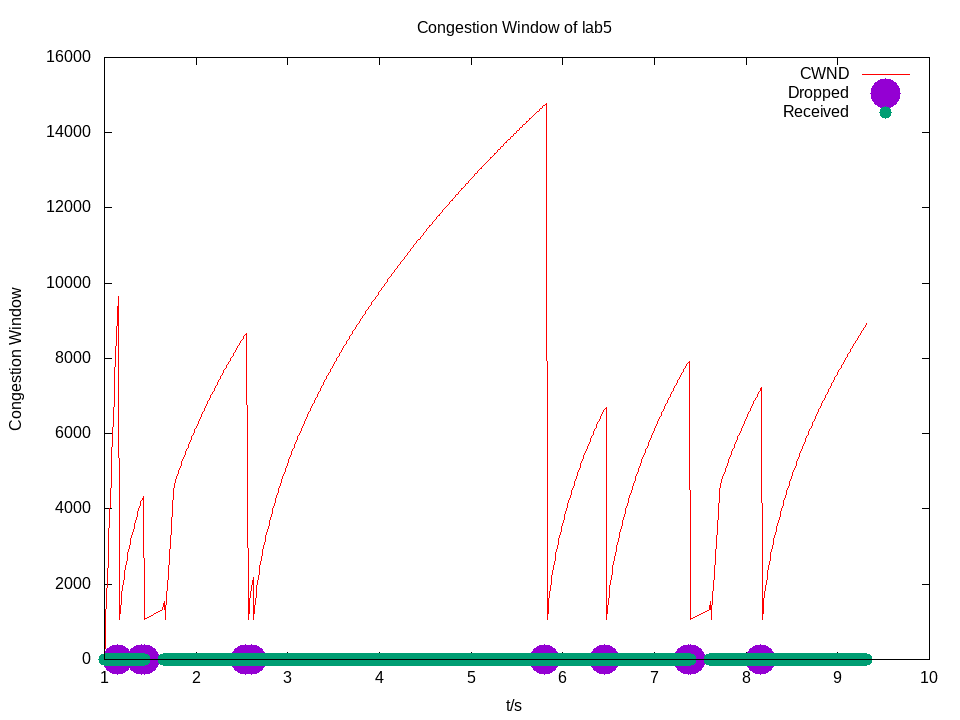
*set output "lab5.png"*

*set xlabel "t/s"*

*set ylabel "Congestion Window"*

*set title "Congestion Window of lab5"*

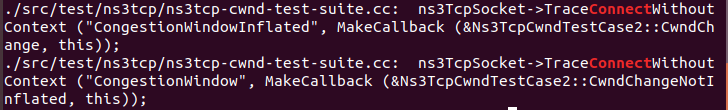
*plot "lab5.cwnd" using 1:2 title "CWND" with lines lc rgb "red" , "lab5\_drop.cwnd" using 1:2 title "Dropped" with points lt 1 pt 7 ps 5, "lab5\_recv.cwnd" using 1:2 title "Received" with points lt 2 pt 7 ps 2*



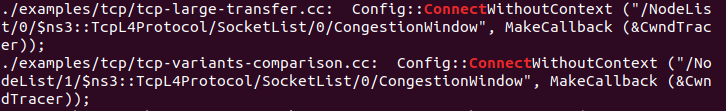
**Lab 6:**

有两种方法可以将跟踪接收器连接到 CongestionWindow。

* **Method 1: using SocketObject->TraceConnect(……)**



* **Method 2: using Config::Connect(…… )**



**我们在lab5中使用了方法1，在lab6.cc中使用了方法2来得到TCP拥塞窗口的值。**

**Hint:**

1. Use a two-node point-to-pint topology. Use BulkSendApplication on 1 node, and PacketSink on the other node. (refer to examples/tcp/tcp-bulk-send.cc)
2. Use the link error model as in mysixth.cc
3. To use Config::Connect (……) , you need to create a simulator event that is run after the dynamic object is created and hook the trace when that event is executed. Refer to /examples/tcp/tcp-variants-comparison.cc for more details.

**Config：：connectWithoutContext方法：**

static void

TraceCwndChange(std::string traceName)

{

  Ptr<OutputStreamWrapper> stream = asciiTraceHelper.CreateFileStream (traceName.c\_str());

  Config::ConnectWithoutContext (

    "/NodeList/0/$ns3::TcpL4Protocol/SocketList/0/CongestionWindow",

    MakeBoundCallback (&CwndChange, stream));

}

static void

TraceDrop(std::string dropName)

{

  Ptr<OutputStreamWrapper> DropFile = asciiTraceHelper.CreateFileStream (dropName.c\_str());

  Config::ConnectWithoutContext (

    "/NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/PhyRxDrop",

    MakeBoundCallback (&RxDrop, DropFile));

}

static void

TraceRecv(std::string recvName)

{

  Ptr<OutputStreamWrapper> RecvFile = asciiTraceHelper.CreateFileStream (recvName);

  Config::ConnectWithoutContext (

    "/NodeList/1/DeviceList/0/$ns3::PointToPointNetDevice/PhyRxEnd",

    MakeBoundCallback (&RxEnd, RecvFile));

}

**Simulation：**

**$./waf --run scratch/lab6**

**Check the .cwnd file, and use gnuplot to draw a figure.**

**$ gnuplot**

*set terminal png size 960, 720*

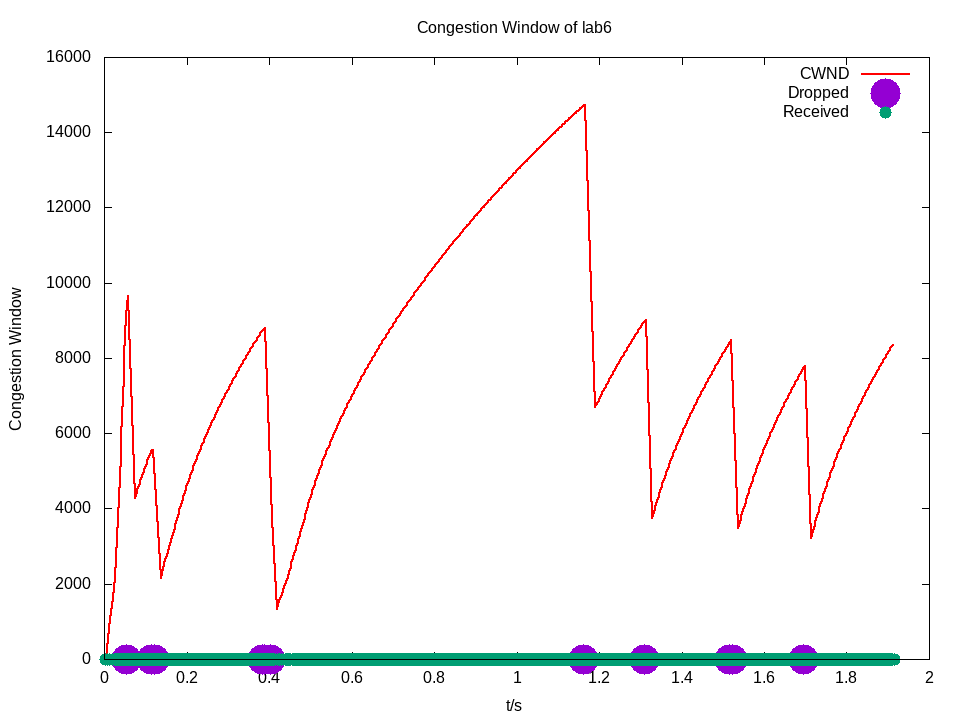
*set output "lab6.png"*

*set xlabel "t/s"*

*set ylabel "Congestion Window"*

*set title "Congestion Window of lab6"*

*plot "lab6.cwnd" using 1:2 title "CWND" with lines lc rgb "red" , "lab6\_drop.cwnd" using 1:2 title "Dropped" with points lt 1 pt 7 ps 5, "lab6\_recv.cwnd" using 1:2 title "Received" with points lt 2 pt 7 ps 2*



1. 实验思考：

* **关于cwnd的计算机网络控制算法：**

拥塞窗口（cwnd）：

技术解释：TCP流量控制方法之一，如果说通告窗口是通过接收数据的一端进行的流量控制，那么拥塞窗口就是发送端进行的流量控制，发送端发送数据包的字节数不会超过拥塞窗口的大小。

1. 慢启动算法

设计背景：

当发送方和接收方在一个速率较慢的链路时，如果一开始发送的数据太多，可能导致耗尽路由器的缓存空间，从而引发网络崩溃，所以tcp在进行数据发送时，选择逐步增加分组的数量来避免此类问题。

具体算法：

1. 当TCP握手完成后，拥塞窗口会被初始化为一个报文段大小，也就是每次只能发送一个报文段

2. 每收到一个ACK，拥塞窗口的大小就会增加1个报文段，以此类推

3. 最终数据发送的字节数，将根据通告窗口和拥塞窗口的较小值作为上限

4. 当到某一个阈值后，可能在网络上出现丢包，此时将会进入拥塞避免算法

1. 拥塞避免算法

设计背景：

1. 认定当分组丢失的时候，网络上就一定发生了拥塞

2. 不考虑移动信号不稳定的情况

3. 分组损坏的可能性较低

具体算法：

1. 当进行拥塞避免时，cwnd每次收到ack后不在进行指数倍增加，具体增加的方法为 cwnd = cwnd + 1/cwnd
2. 当拥塞发生时，发送数据包丢失，例如超时确认或者重复的确认包，ssthress将会被设置为当前窗口（拥塞窗口和通告窗口的最小值）的一半
3. 如果是因为超时确认引起的丢包，那么cwnd将会被初始化为1个报文段

* **关于使用Bulk类：**

此流量生成器只是尽可能快地将数据发送到 MaxBytes，或者直到应用程序停止（如果 MaxBytes 为零）。一旦下层发送缓冲区被填满，它就会等到空间有空来发送更多数据，基本上保持恒定的数据流。仅支持SOCK\_STREAM和SOCK\_SEQPACKET套接字。例如，可以使用 TCP 套接字，但不能使用 UDP 套接字。

可以在属性中设置发送大小，最大字节数等参数以控制传输速率。