shell

```
vagrant@guest1:~$ ftp -n < src/wireshark/ftp_conf.txt
```

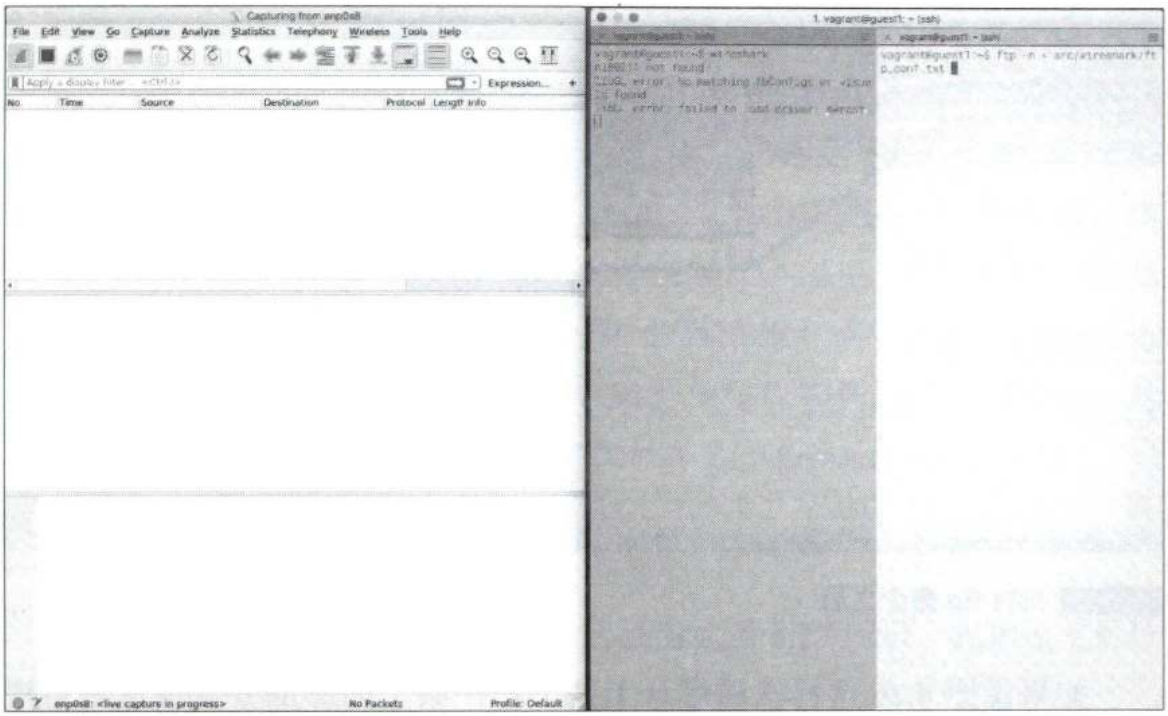

图 4.24 运行 ftp 命令之前

上述 shell 命令在 ftp 命令上加上了 -n 的参数选项，这样就可以不启动登录会话操作，直接自动运行 ftp_conf.txt 中保存的 ftp 命令。ft_conf.txt 是如下文所示的批处理文件。

ftp_conf.txt

```
open 192.168.33.20 ←open {域名}
user vagrant vagrant ←user {用户名} {密码}
prompt ←关闭提示模式
put tempfile ←上传tempfile
```

刚才的 ftp 命令是否正确运行了呢？ ftp 命令运行后的状态如图 4.25 所示。从图中可以看出，为了发送 100 MB 的文件，在约 0.6024 秒的时间内，guest1 和 guest2 之间收发了 5697 个数据包。

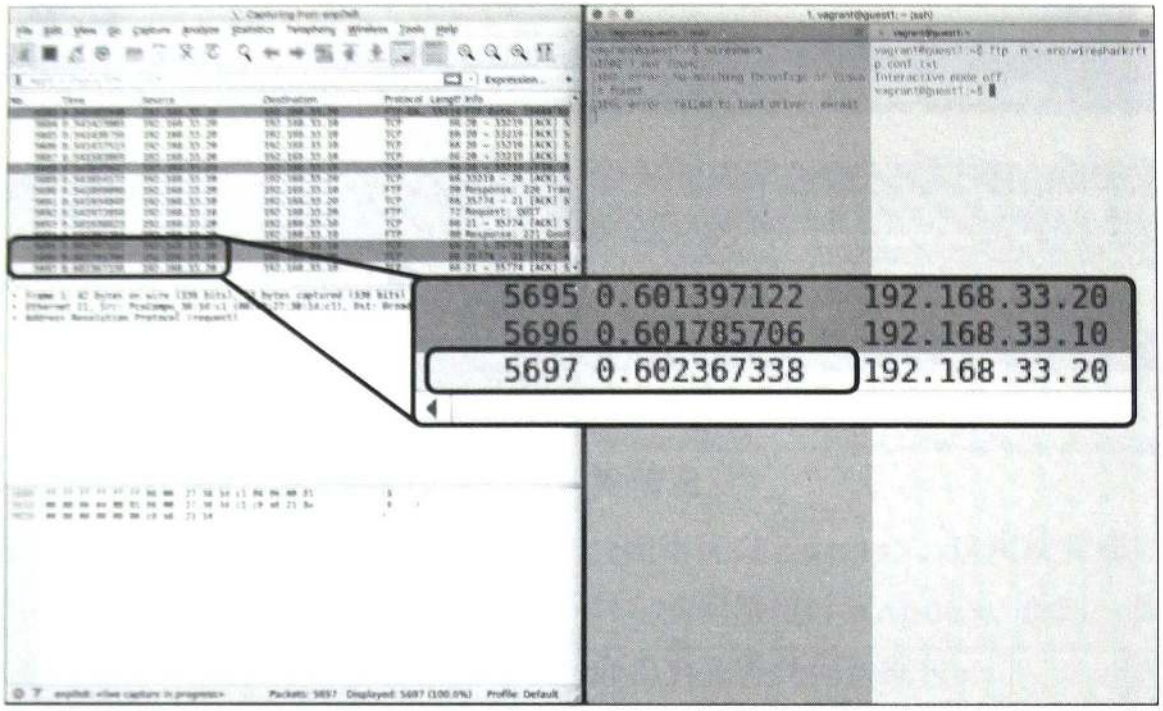

图 4.25 运行 ftp 命令之后

如果像图 4.26 这样按顺序从上往下扫一眼上面那部分数据包总览视图，可以看到 FTP 协议数据传输的具体流程：首先客户端（client）通过 ARP（详见 1.1 节）获取到 192.169.33.20 的 MAC 地址，然后进行 3 次握手，与服务端（sever）建立 TCP 连接。

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	PcsCompu_38:1d:c1	Broadcast	ARP	42	Who has 192.168.33
2	0.000267529	PcsCompu_99:00:8c	PcsCompu_38:1d:c1	ARP	60	192.168.33.20 is a
3	0.000273192	192.168.33.10	192.168.33.20	TCP	74	35774 → 21 [SYN] S
4	0.000468759	192.168.33.20	192.168.33.10	TCP	74	21 → 35774 [SYN, A
5	0.000483268	192.168.33.10	192.168.33.20	TCP	66	35774 → 21 [ACK] S
6	0.002601696	192.168.33.20	192.168.33.10	FTP	86	Response: 220 (vsF
7	0.002643806	192.168.33.10	192.168.33.20	TCP	66	35774 → 21 [ACK] S
8	0.002785309	192.168.33.10	192.168.33.20	FTP	80	Request: USER vagr
9	0.002831046	192.168.33.20	192.168.33.10	TCP	66	21 → 35774 [ACK] S
10	0.002910336	192.168.33.20	192.168.33.10	FTP	100	Response: 331 Plea
11	0.002945613	192.168.33.10	192.168.33.20	FTP	80	Request: PASS vagr
12	0.041300035	192.168.33.20	192.168.33.10	TCP	66	21 → 35774 [ACK] S
13	0.051493823	192.168.33.20	192.168.33.10	FTP	89	Response: 230 Logi
14	0.051631793	192.168.33.10	192.168.33.20	FTP	72	Request: SYST
15	0.051934929	192.168.33.20	192.168.33.10	TCP	66	21 → 35774 [ACK] S

图 4.26 FTP 文件传输时最开始的几个数据包

观察拥塞控制算法的行为 TCP Stream Graphs 功能

接下来，使用 Wireshark 的 Statistics 菜单中的 TCP Stream Graphs 功

能，来观察拥塞控制算法的行为。不过请注意，只有从在上半部分视图中选择的发送方（IP 地址、端口号）到目的地（IP 地址、端口号）的统计结果会被显示出来。

FTP 会建立两条 TCP 连接，其中一条用于控制，而另一条用于数据传输。前者是从客户端（guest1）到服务器端（guest2）的连接，使用端口号 21；后者是从服务器端（guest2）到客户端（guest1）的连接，使用端口 20。本次模拟关注的重点是数据传输的 TCP 连接，因此这里首先找一下发送方 IP 地址是 192.168.33.10，目的地 IP 地址是 192.168.33.20，目的地端口号是 20 的数据包。虽然不同的环境下会有所不同，不过从上往下数，在约第 20 行便可以找到对应的数据包。选中这个数据包之后，点击 Statistics 菜单中的 TCP Stream Graphs 选项。

——可绘制的 5 种曲线图

TCP Stream Graphs 功能启动后的画面如图 4.27 所示。此功能支持绘制以下 5 种曲线图。

- Time Sequence（时间序列，Stevens）
- Time Sequence（时间序列，tcptrace）
- Throughput（吞吐量）
- Round Trip Time（往返时延）
- Window Scaling（窗口扩大）

Time Sequence（Stevens）是描述发送序列号随时间产生的变化的曲线（图 4.28）。由于可绘制的曲线图与 W. 查理德·史蒂文斯（W. Richard Stevens）所著的“TCP/IP 详解”系列图书中出现的曲线图一样，因此它被称为 Stevens。

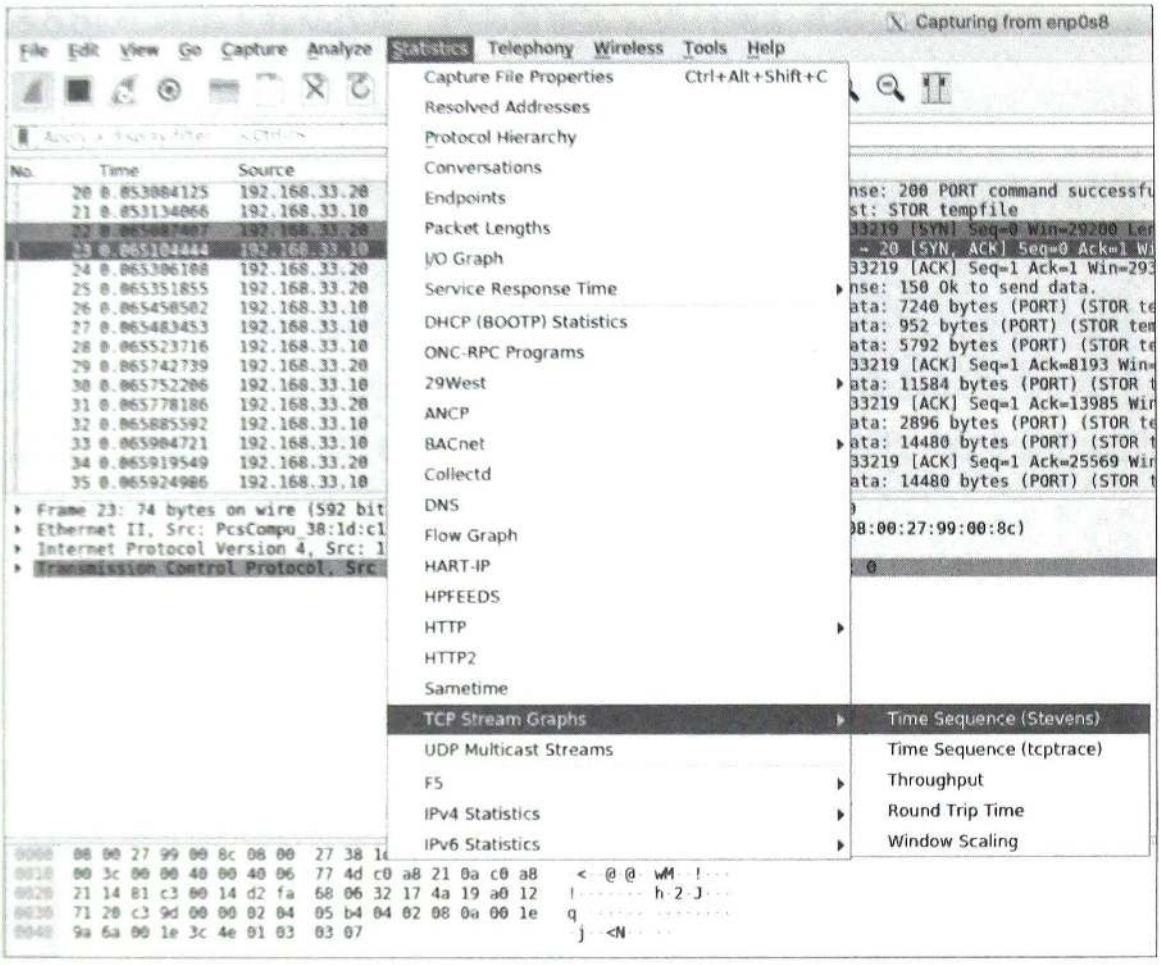

图 4.27 Wireshark 的 TCP Stream Graphs 功能

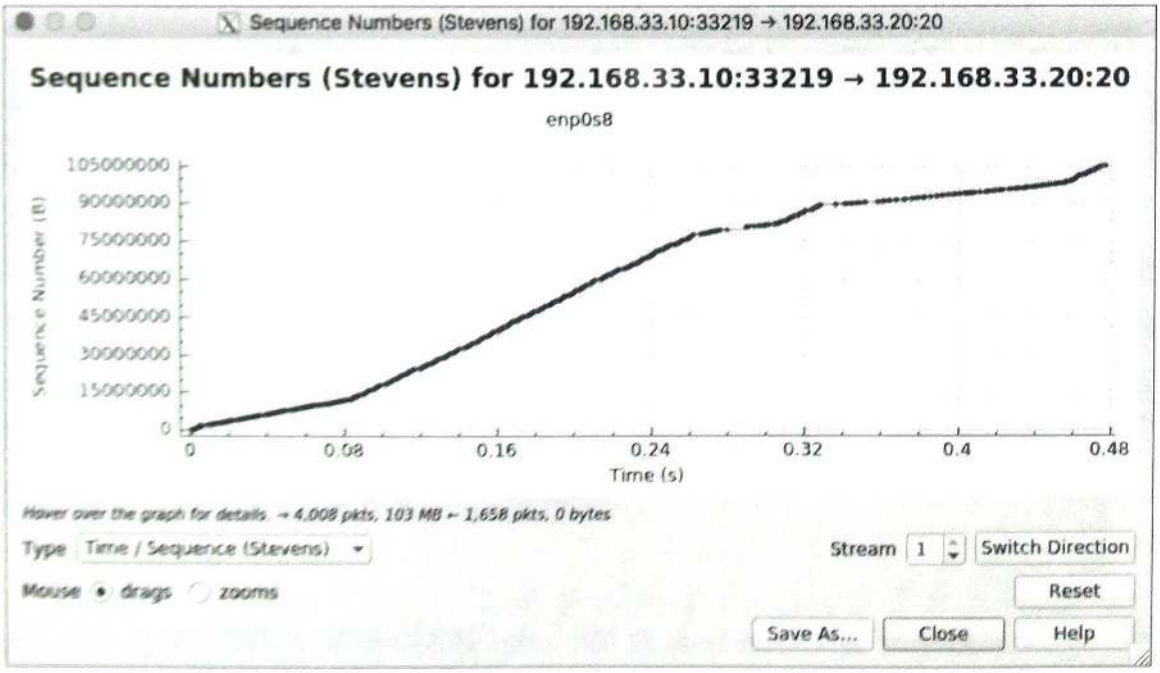

图 4.28 TCP Stream Graphs 的 Time Sequence（Stevens）

Time Sequence（tcptrace）如图 4.29 所示，曲线图上不仅有发送序列号，还有 ACK、SACK 等随时间变化的曲线。

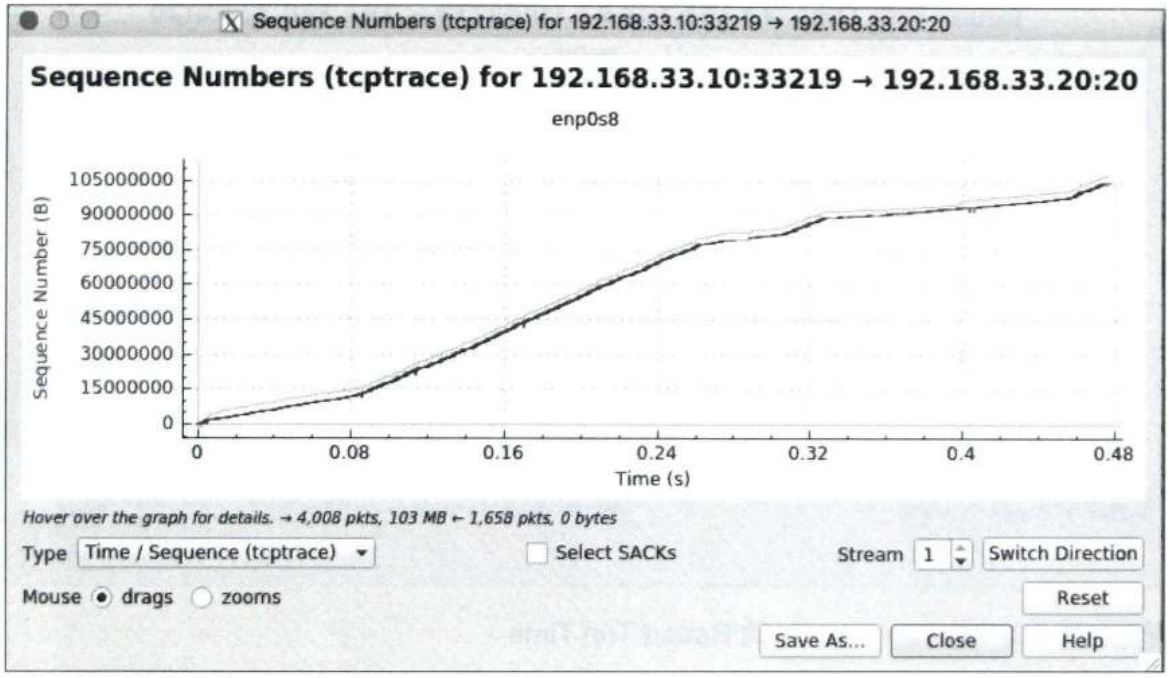

图 4.29 TCP Stream Graphs 的 Time Sequence（tcptrace）

Throughput 如图 4.30 所示，是 TCP 段长度和平均吞吐量随时间变化的曲线图。

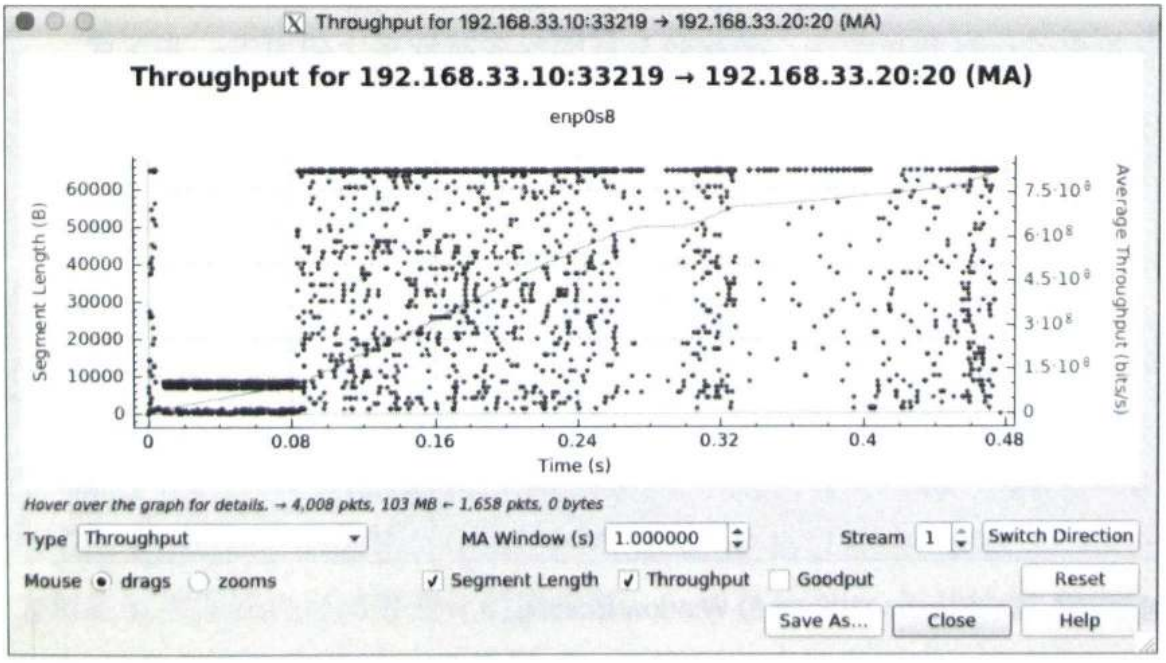

图 4.30 TCP Stream Graphs 的 Throughput

Round Trip Time 如图 4.31 所示，是 *RTT* 随时间变化的曲线图。

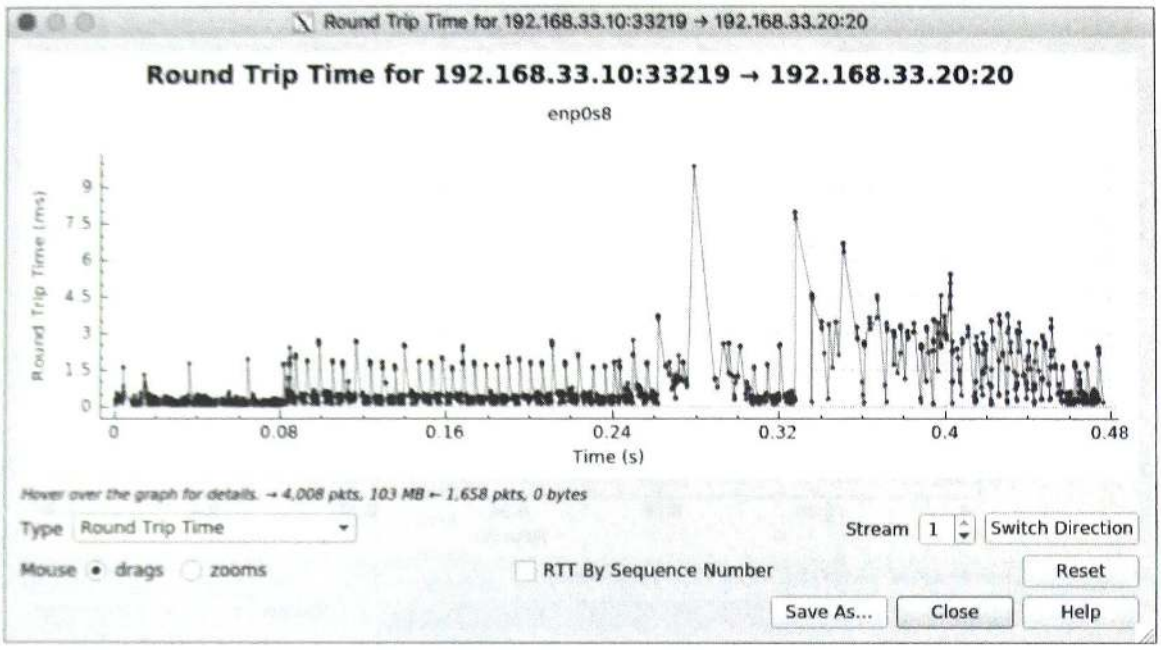

图 4.31 TCP Stream Graphs 的 Round Trip Time

Window Scaling 如图 4.32 所示，是接收窗口大小（*rwnd*）和发送中的数据量（*swnd*）随时间变化的曲线图。

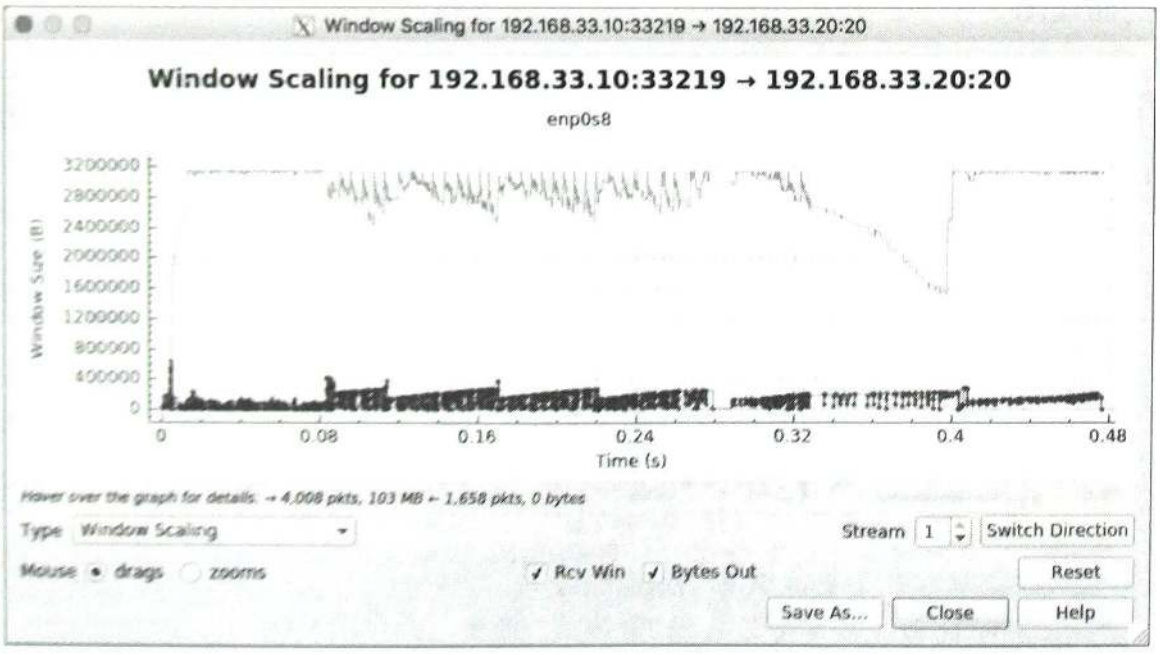

图 4.32 TCP Stream Graphs 的 Window Scaling

——更改拥塞控制算法 sysctl 命令（Ubuntu）

如果采用其他的拥塞控制算法，而非 Reno，它们又会有什么样的表现呢？ Ubuntu 可以通过以下的 `sysctl` 命令更改拥塞控制算法。

shell
```
vagrant@guest1:~$ sudo sysctl -w net.ipv4.tcp_congestion_control=bic
```

上面的 shell 命令使用了 BIC 作为例子，它其实也可以修改为其他的拥塞控制算法。请随意地修改为自己喜欢的算法，并观察其行为。

本章使用 Wireshark 对使用 FTP 发送文件时发送序列号、ACK 的行为变化进行了观察。通过此项模拟，我们虽然可以确认到拥塞控制算法的外部行为表现，却无法掌握 *cwnd*、*ssthresh* 等内部变量的变化情况。

因此，下一章我们将使用网络模拟器 ns-3 对拥塞控制算法内部的变量进行研究。

4.4 加深理解：网络模拟器 ns-3 入门

拥塞控制算法的观察 ❷

为了进一步加深大家对拥塞控制算法的理解，本节将使用离散事件驱动型网络模拟器 ns-3，来观察 *cwnd*、*ssthresh* 等内部变量的情况。同上一节一样，笔者已经准备好了虚拟环境的设置文件，请大家务必亲自动手去探索拥塞控制世界的奥妙。

ns-3 的基本情况

ns-3（Network Simulator 3）是以网络研究和教育为目的的离散事件驱动型网络模拟器。离散事件驱动型网络模拟器是指以数据包的收发等事件为契机驱动系统进行离散变化的模拟器。ns-3 是从 2006 年开始基于 GNU GPL v2 协议开源开发的，它提供了一个实际实现极为困难、高度可控且

可重现性强的网络模拟环境。

ns-3 是一个由多个库组合构建形成的系统，进行外部扩展也十分容易。例如，ns-3 可以与动画生成、数据分析和可视化工具等协同工作，与其他一些只为在 GUI 上进行操作的网络模拟器相比，它更加模块化。ns-3 还可以运行在 Linux、FreeBSD 和 Cygwin 上。

ns-3 可以通过 C++ 或者 Python 脚本文件实现，因此理论上可以组建任何网络。它自带了若干个示例脚本文件，建议大家在使用前先阅读一下这些范例脚本。本节将使用 `chapter4-base.cc` 脚本来进行拥塞控制算法的对比模拟，这个脚本在 ns-3 示例脚本 `tcp-variants-comparison.cc` 的基础上进行了一部分修正。此外，这里使用 Python 对输出的文件进行分析和可视化处理。

搭建 ns-3 环境

这里与 Wireshark 一样，我们首先通过 Virtualbox 和 Vagrant 在虚拟机上搭建 Ubuntu 16.04 的运行环境，然后在环境上运行 ns-3。为了完成这一步，需要搭建导言部分描述的 VirtualBox、Vagrant 和 X 客户端的环境。此外，本节同样将物理机上安装的操作系统称为宿主操作系统，将虚拟机上安装的操作系统称为客户操作系统。

当确认已经准备好 VirtualBox 和 Vagrant 的环境之后，请将本书的 Github 仓库[1] 克隆到任意目录中（❶）。然后，打开其中的 `ns3/vagrant` 目录（❷），运行 `vagrant up` 命令（❸）。如此一来，就完成了在虚拟机上安装 Ubuntu 16.04 并搭建 ns-3 的过程。

shell

```
$ git clone https://github.com/ituring/tcp-book.git ←❶
$ cd tcp-book/ns3/vagrant ←❷
$ vagrant up ←❸
```

另外，在笔者执笔时（2019 年 4 月 1 日），第 5 章和第 6 章所使用的 CUBIC 和 BBR 模块尚不支持 ns-3.28 以上版本，因此本书使用 ns-3.27 版

1 URL https://github.com/ituring/tcp-book

本。搭建 ns-3 环境相当花时间，还请大家耐心等待[①]。

接下来，通过 SSH 连接到客户操作系统上。

shell

```
$ vagrant ssh
> Welcome to Ubuntu 16.04.5 LTS (GNU/Linux 4.4.0-139-generic x86_64)
>
>  * Documentation:  部分省略
>  * Management:     部分省略
>  * Support:        部分省略
>
>   Get cloud support with Ubuntu Advantage Cloud Guest:
>   部分省略
>
> 13 packages can be updated.
> 6 updates are security updates.
>
> New release '18.04.1 LTS' available.
> Run 'do-release-upgrade' to upgrade to it.
>
>
vagrant@ubuntu-xenial:~$
```

如上所述，当提示 `vagrant@ubuntu-xenial:~$` 之后，SSH 就连接成功了。

基于 ns-3 的网络模拟的基础知识

在进行拥塞控制算法的比较模拟之前，我们先来学习一下 ns-3 的网络模拟基础知识。由于篇幅所限，所以本书只涉及方便理解模拟技术的最为基础的知识[②]。

在使用 SSH 连接到客户操作系统的状态下，打开 `ns3/ns-allinone-3.27/ns-3.27` 目录。此目录是后文中的 ns-3 的根目录，只要没有特别注明，所有的目录路径都指的是从这个根目录开始的相对路径。

shell

```
vagrant@ubuntu-xenial:~$ cd ns3/ns-allinone-3.27/ns-3.27
```

① 在笔者的计算机环境下，花费的时间约为 1 个小时。

② 具体请参考官方手册“ns-3 Manual”。

主要目录组成

这里介绍一下根目录下最重要的一些目录和文件。

shell

```
vagrant@ubuntu-xenial:~/ns3/ns-allinone-3.27/ns-3.27$ tree -L 2
> .
> ├── scenario_4.py 进行数据处理和可视化处理的Python脚本
> ...
> ├── data 保存输出文件的目录
> ...
> ├── examples 保存ns-3自带示例脚本文件的目录
> │   ├── tcp 本次模拟使用的脚本文件的参考源文件，
>                 保存tcp-variants-comparison.cc的目录
> ...
> ├── requirements.txt 罗列了所有用于数据处理和可视化处理的Python库文件
> ...
> ├── scratch ns-3自带的，用于保存用户自己开发的脚本文件的目录
> │   ├── chapter4-base.cc 本次模拟所使用的脚本文件
> ...
> ├── src 保存ns-3源代码的目录。当需要加载独有的协议时，会涉及这个目录
> │   ├── internet TCP拥塞控制算法所安装的目录
> ...
> ├── waf 负责脚本文件的编译和运行的目录
> ...
> ├── wscript waf的设置文件
```

ns-3 的命令

ns-3 主要通过运行 ./waf –run {脚本名称} {命令行参数} 命令启动模拟过程。在默认设置下，可以指定的脚本文件只能在根目录（./）或者 scratch/ 目录下。如果想要增加可使用的目录，就必须修改 wscript 文件。例如，想要运行 scratch/chapter4-base.cc 脚本，就需要输入以下命令。注意，这里需要去掉扩展名（.cc）。

shell

```
vagrant@ubuntu-xenial:~/ns3/ns-allinone-3.27/ns-3.27$ ./waf --run chapter4-base
> Waf: Entering directory '/home/vagrant/ns3/ns-allinone-3.27/ns-3.27/build'
部分省略
> [1969/1980] Linking build/bindings/python/ns/spectrum.so
> Waf: Leaving directory '/home/vagrant/ns3/ns-allinone-3.27/ns-3.27/build'
> Build commands will be stored in build/compile_commands.json
> 'build' finished successfully (2m0.891s)
```

可以通过加入如下所示的命令行参数 `--PrintHelp` 查看所有可使用的命令行参数。请注意从脚本名称开始到命令行参数为止的部分需要加上双引号（""）。

shell

```
vagrant@ubuntu-xenial:~/ns3/ns-allinone-3.27/ns-3.27$ ./waf --run "chapter4-base --PrintHelp"
> Waf: Entering directory '/home/vagrant/ns3/ns-allinone-3.27/ns-3.27/build'
> Waf: Leaving directory '/home/vagrant/ns3/ns-allinone-3.27/ns-3.27/build'
> Build commands will be stored in build/compile_commands.json
> 'build' finished successfully (0.799s)
> chapter4-base [Program Arguments] [General Arguments]
>
> Program Arguments:
>     --transport_prot:    Transport protocol to use: TcpNewReno, TcpHybla, TcpHigh↵
Speed, TcpHtcp, TcpVegas, TcpScalable, TcpVeno, TcpBic, TcpYeah, TcpIllinois, TcpW↵
estwood, TcpWestwoodPlus [TcpWestwood]
>     --error_p:           Packet error rate [0]
>     --bandwidth:         Bottleneck bandwidth [2Mbps]
部分省略
>     --sack:              Enable or disable SACK option [true]
>
> General Arguments:
>     --PrintGlobals:              Print the list of globals.
>     --PrintGroups:               Print the list of groups.
>     --PrintGroup=[group]:        Print all TypeIds of group.
>     --PrintTypeIds:              Print all TypeIds.
>     --PrintAttributes=[typeid]:  Print all attributes of typeid.
>     --PrintHelp:                 Print this help message.
```

下文将详细介绍 `chapter4-base.cc` 的内容。

脚本文件 chapter4-base.cc

本次模拟所使用的 `chapter4-base.cc` 脚本文件，主要是基于 ns-3.27 的示例脚本文件之一的 `examples/tcp/tcp-variants-comparison.cc` 修改制作而成。具体来说就是，增加一部分代码逻辑，以获取 `examples/tcp/tcp-variants-comparison.cc` 的逻辑中无法获取的 ACK 和状态迁移情况。修改的详细内容均可在源代码的注释中找到。感兴趣的读者请务必参阅。

图 4.33 展示的是 `chapter4-base.cc` 所假想的网络构成。

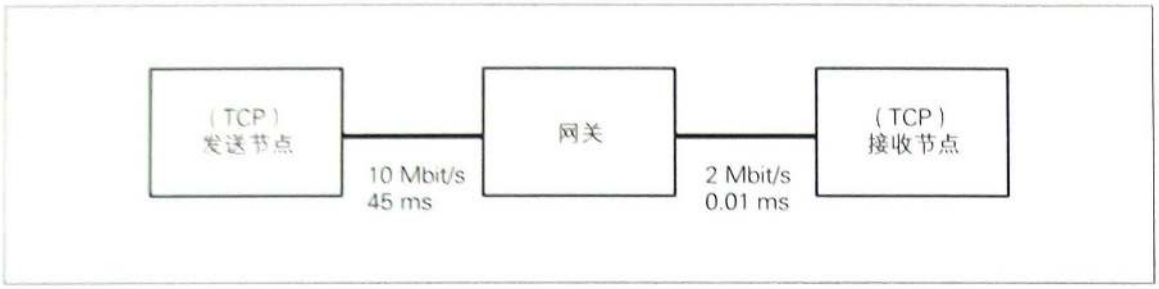

图 4.33 ns-3 中的网络构成

此脚本文件进行的是从发送节点到接收节点的文件传输，可以设置的命令行参数主要有以下这些。

- `transport_prot`：拥塞控制算法。默认值为 `Westwood`
- `error_p`：数据包错误率。默认值为 `0`
- `bandwidth`：网关与接收节点之间的带宽。默认值为 `2 Mbit/s`
- `delay`：网关与接收节点之间的链路传播时延（详见 5.1 节）。默认值为 `0.01 ms`（millisecond，毫秒）
- `access_bandwidth`：发送节点与网关之间的带宽。默认值为 `10 Mbit/s`
- `access_delay`：发送节点与网关之间的传播时延。默认值为 `45 ms`
- `tracing`：是否激活追踪功能。默认值为 `false`，但这样的话无法得到分析数据，因此需要设置为 `true`
- `prefix_name`：输出文件保存位置。默认值为 `TcpVariantsComparison`
- `data`：待发送文件的大小（单位：Mbit/s）。默认值为 `0`，代表无限大
- `mtu`：IP 数据包的大小（单位：byte）。默认值为 `400`
- `num_flows`：TCP 流个数。默认值为 `1`
- `duration`：文件的最长传输秒数。如果将 `duration` 的值设置得过大，那么会耗费大量模拟时间。默认值为 `100`
- `run`：用于生成随机数的索引值。默认值为 `0`
- `flow_monitor`：是否激活 Flow monitor 功能。默认值为 `false`
- `pcap_tracing`：是否激活 PCAP tracing 功能。默认值为 `false`
- `queue_disc_type`：网关所使用的队列类型。默认值为 `ns3::PfifoFastQueueDisc`

- sack：是否激活 SACK（Selective ACKnowledge，选择确认应答）。默认值为 true

本次模拟将多次调整上述参数中的 transport_prot，并与 4.2 节中出现的所有拥塞控制算法进行对比。例如，在将拥塞控制算法修改为 TcpNewReno 时，要运行下面的命令。

```shell
vagrant@ubuntu-xenial:~/ns3/ns-allinone-3.27/ns-3.27$ ./waf --run "chapter4-base
--transport_prot='TcpNewReno' --tracing=True --prefix_name='data/chapter4/TcpNew
Reno/'"
```

使用 Python 运行模拟器并进行分析和可视化

本次模拟将统一使用 Python 完成模拟器的运行、数据分析和可视化。首先构建虚拟环境，打开 ns-3 的根目录。

```shell
$ vagrant up
vagrant@ubuntu-xenial:~$ cd ns3/ns-allinone-3.27/ns-3.27
```

4.2 节展示的所有图形都可以通过以下命令输出出来。输出位置是在客户操作系统的 ~/ns3/ns-allinone-3.27/ns-3.27/data/chapter4 目录之下。由于客户操作系统的 ~/ns3/ns-allinone-3.27/ns-3.27/data 目录与客户操作系统的 tcp-book/ns3/vagrant/shared/ 目录是同步的，所以也可以在宿主操作系统上看到里面的文件。此外，下文中的 tcp-book/ 指的是从 https://github.com/ituring/tcp-book 下载回来的目录。

```shell
vagrant@ubuntu-xenial:~/ns3/ns-allinone-3.27/ns-3.27$ python3 scenario_4.py
```

运行完上述命令之后，请确认一下宿主操作系统的 tcp-book/ns3/vagrant/shared/chapter4 目录下的内容。该目录下的文件构成应该如下文所述。

- 04_tcpbic.png：图 4.12
- 04_tcphighspeed.png：图 4.9
- 04_tcphtcp.png：图 4.13
- 04_tcphybla.png：图 4.14
- 04_tcpillinois.png：图 4.15
- 04_tcpnewreno.png：图 4.6
- 04_tcpscalable.png：图 4.10
- 04_tcpvegas.png：图 4.7
- 04_tcpveno.png：图 4.11
- 04_tcpwestwood.png：图 4.8
- 04_tcpyeah.png：图 4.16
- TcpBic：存储 BIC 相关输出数据的目录
- TcpHighSpeed：存储 HighSpeed 相关输出数据的目录
- TcpHtcp：存储 H-TCP 相关输出数据的目录
- TcpHybla：存储 Hybla 相关输出数据的目录
- TcpIllinois：存储 Illinois 相关输出数据的目录
- TcpNewReno：存储 NewReno 相关输出数据的目录
- TcpScalable：存储 Scalable 相关输出数据的目录
- TcpVegas：存储 Vegas 相关输出数据的目录
- TcpVeno：存储 Veno 相关输出数据的目录
- TcpWestwood：存储 Westwood 相关输出数据的目录
- TcpYeah：存储 Yeah 相关输出数据的目录

从 TcpBic 到 TcpYeah 的所有目录都包含以下文件。

- ack.data：接收的 ACK 序列号历史记录
- ascii：收发事件的日志
- cong-state.data：状态变化的历史记录
- cwnd.data：*cwnd* 的历史记录
- inflight.data：*swnd* 的历史记录
- next-rx.data：下一个要接收的 ACK 序列号的历史记录

- next-tx.data：下一个要发送的 ACK 序列号的历史记录
- rto.data：超时时间的历史记录
- rtt.data：*RTT* 历史记录
- ssth.data：ssthresh 的历史记录

由于本次模拟没有使用 ascii，所以不再介绍。其他的数据都是用 Tab 分隔的 2 列数据，其中第 1 列是经过的秒数，而第 2 列是对应的值。例如，在把 TcpNewReno 的 ack.data 用文本编辑的方式打开时，就能看到下列数据。

ack.data

```
0.0905768 1
0.18279 341
0.276537 1021
0.370283 1701
0.462454 2381
0.465606 3061
0.5562 3741
0.559352 4421
0.562504 5101
0.649946 5781
```

由于篇幅所限，这里只列了前 10 行数据。例如，第 1 行表示在模拟开始后的 0.090 576 8 秒收到了序列号为 1 的 ACK。第 2 行以后的数据也类似。直觉较为敏锐的读者可能已经发现，4.2 节的图正是用 cwnd.data、ssth.data、ack.data、rtt.data 和 cong-state.data 的数据绘制出来的图形。

scenario_4.py 的内容 Python 入门

如果任意改变模拟环境，再对得到的不同结果进行分析，就可以加深我们对拥塞控制算法的理解。真正的 ns-3 编程已经超出了本书所涉及的范围，因此接下来会介绍一下使用 scenario_4.py 简单地改变模拟环境的方法。这里虽然需要进行 Python 编程，但并非意味着必须掌握 Python 的前置知识。此外，这部分内容是逐步推进的，所以请放心阅读。

虽然可能有点跑题，不过还是需要首先介绍一下第 155 页出现过的命令 python3 scenario_4.py。这一句的含义是，使用 Python 3 运行