scenario_4.py这个脚本文件。Python的两个版本Python 2和Python 3无法兼容，这一点比较麻烦。2010年7月3日发布的Python 2.7是最后的Python 2版本。由于现在正在逐步往Python 3迁移，所以本书使用Python 3。

我们一起来看一下scenario_4.py的内容。如前文所述，本书是TCP的入门书，而非Python入门书。因此，下文在介绍Python时更注重怎样才能方便理解，而非语法的严密性。Python是如今世界上最为流行的编程语言之一，市面上有许多优秀的图书和网络资料可供参考和学习。感兴趣的读者请务必了解和学习一下。

当从命令行读取到Python脚本之后，if __name__ == __'main'__之后的代码会被运行。在scenario_4.py中，以下的main()函数会被调用。

scenario_4.py

```
def main():
    for algo in tqdm(algorithms, desc='Algotirhms'):
        execute_and_plot(algo=algo, duration=20)
```

Python使用def {函数名}({参数}):的方式来定义函数。因此，def main():代表着定义一个名为main且没有参数的函数。Python的特点之一便是通过缩进提高源代码的可读性。也就是说，从def main():之后的下一行即第2行开始，就是main()函数的具体内容。另外，虽然无论缩进有多少个空格，Python都可以正常运行，但是代码指南PEP 8中推荐使用4个空格作为缩进值。

第2行是一个按顺序读取algorithms数组中的所有元素，然后循环赋值给变量algo并进行处理的操作。也就是说，第1次是将TcpNewReno赋给algo，第2次则是将TcpHybla赋给algo，然后完成从第3行往后的运算操作。tqdm是表示for循环进行状态的函数，这里暂时忽略也没问题。

scenario_4.py

```
algorithms = [
    'TcpNewReno', 'TcpHybla', 'TcpHighSpeed', 'TcpHtcp',
    'TcpVegas', 'TcpScalable', 'TcpVeno', 'TcpBic', 'TcpYeah',
    'TcpIllinois', 'TcpWestwood']
```

接下来，更下一级缩进的第 3 行代表的是 for 循环的内部处理逻辑。这里的主要含义是，使用拥塞控制算法 algo 模拟运行 execute_and_plot() 函数，时间是 20 秒。execute_and_plot() 函数的主要作用是，在指定的模拟环境下进行 ns-3 模拟，并将运行结果数据保存到 ./data/chapter4/{ 算法名 }/，同时将绘图结果保存到 ./data/chapter4/04_{ 算法名 }.png。

execute_and_plot() 中可设置的参数如下所示。

- algo：可设置拥塞控制算法
- duration：设置文件传送最长运行时间。注意如果将 duration 的值设置得过大，会导致模拟时花费大量的时间
- 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。默认值为 0.01 ms
- 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
- 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
- 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

也就是说，开头的 python3 scenario_4.py 命令的含义是，设置模拟时间为 20 秒，并使用所有的拥塞控制算法运行 execute_and_plot() 函数。实际上，只要任意修改 execute_and_plot() 的函数参数，就可以自由地调整模拟环境，得到想要的运行结果。

确认运行情况 IPython

假设我们要观察一下提高数据包错误率之后的 NewReno 的运行情况。由于可以通过 error_p 设置错误率，所以我们运行下面的命令。

shell

```
vagrant@ubuntu-xenial:~/ns3/ns-allinone-3.27/ns-3.27$ ipython
> Python 3.5.2 (default, Nov 12 2018, 13:43:14)
> Type 'copyright', 'credits' or 'license' for more information
> IPython 7.2.0 -- An enhanced Interactive Python. Type '?' for help.
>
In [1]: import scenario_4
In [2]: scenario_4.execute_and_plot('TcpNewReno', 20, error_p=0.01)
```

ipython 是启动 IPython 的命令。IPython 是交互式运行 Python 程序的工具。虽然我们也可以在默认的 Python 解释器上运行 Python 程序，但是由于 IPython 拥有 Tab 补全等便利功能，所以这里比较推荐使用 IPython。

在 Python 中，如果想要使用外部的模块，就需要输入 import {模块名} 代码。已经 import 过的模块中的函数可以通过 {模块名}.{函数名} 使用。这里就将拥塞控制算法设置为 TcpNewReno，模拟时间设置为 20 秒，数据包错误率设置为 0.01，运行 execute_and_plot() 函数。

当显示了 In [3]: 之后，就表示运行已经完成。接下来，看一下宿主操作系统的 tcp-book/ns3/vagrant/shared/chapter4/04_tcpnewreno.png 的内容。如图 4.34 所示，算法的运行情况已与图 4.6 大为不同，丢包导致 *cwnd* 无法变得太大。此外，由于丢包是随机发生的，所以读者得到的结果图形可能与图 4.34 并非完全一致，这一点请注意。

shell

```
In [3]: scenario_4.execute_and_plot('TcpNewReno', 20, delay='1s', access_delay='1s')
```

此外，如何确认长距离通信中 NewReno 算法的表现呢？只要调整 delay 或者 access_delay 的值，便可以很简单地得到相应的数据。

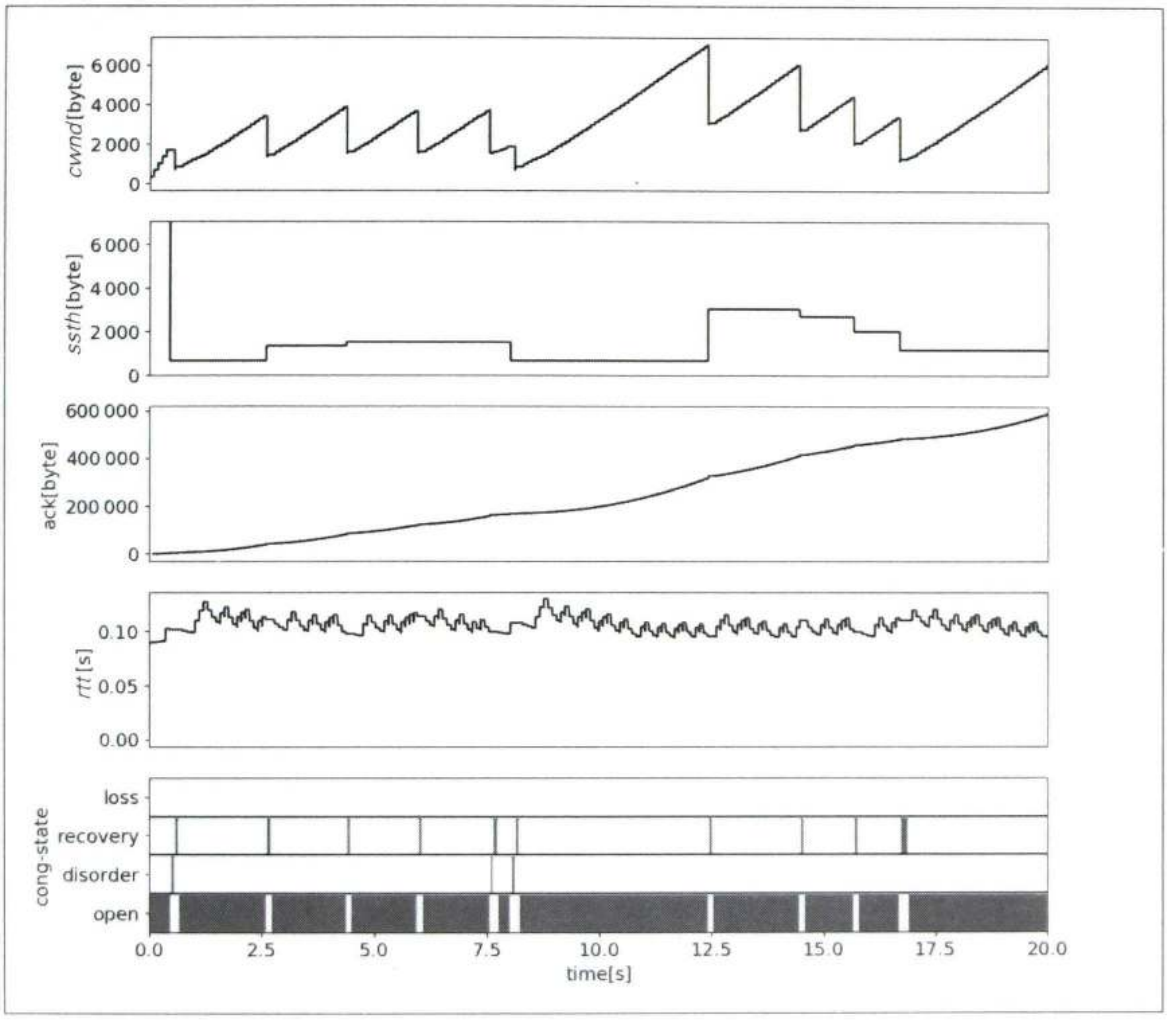

图 4.34 NewReno 在易发生数据包错误的环境下的表现

当显示了 `In [4]:` 之后，就表示运行已经完成。接下来，看一下宿主操作系统的 `tcp-book/ns3/vagrant/shared/chapter4/04_tcpnewreno.png` 的内容（图 4.35）。从图中可以看出，由于信号往返需要 4 秒左右的时间，所以 *cwnd* 在模拟结束之前基本上没变大。

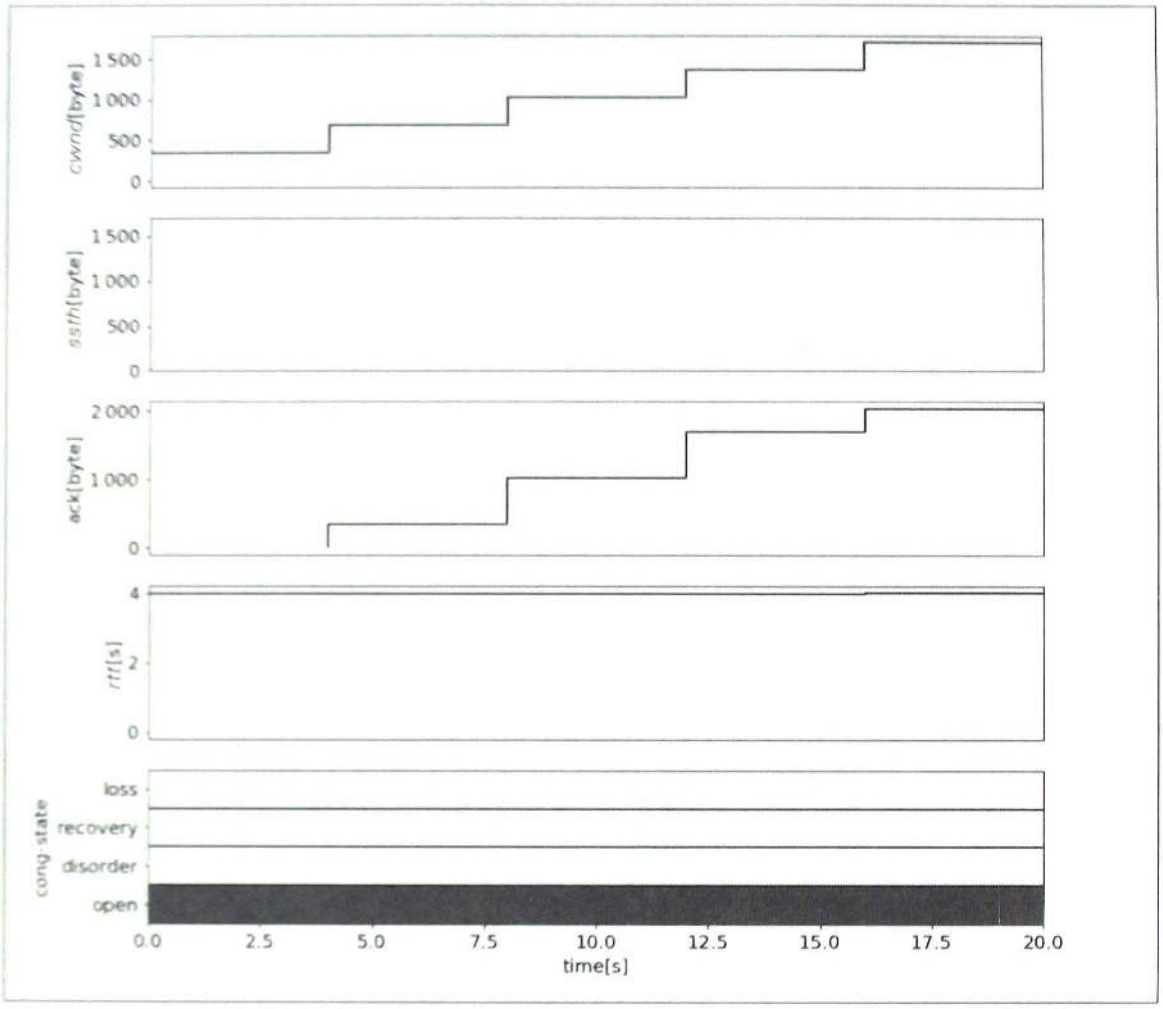

图 4.35 长距离通信下 NewReno 算法的表现

这里举的只是简单的例子。请务必尝试使用各种各样的模拟环境，加深对拥塞控制算法的理解。此外，只要输入 `exit`，就可以退出 IPython。

shell

```
In [4]: exit
vagrant@ubuntu-xenial:~/ns3/ns-allinone-3.27/ns-3.27$
```

本节介绍了使用离散事件驱动型网络模拟器 ns-3 观察 *cwnd*、*ssthresh* 等内部变量变化情况的方法，同时也提及了使用本书专用的 `scenario_4.py` 模块，就可以快捷简便地观察各种环境下拥塞控制算法的表现。

这里有一点值得注意，本节的专用模块只是 Python 脚本文件 `chapter4-base.cc` 的一个封装，所能设置的模拟环境有限。例如，无法模拟多个拥塞控制算法并存的环境。要想搭建这种复杂的网络环境，就必须亲自实

现 ns-3 脚本文件。而且，显而易见的是，ns-3 只能评估现有的拥塞控制算法。想要评估私有的拥塞控制算法，就必须修改 ns-3 的源代码。

ns-3 模拟器被世界各国的研究者所称道，可以说是一个相当深奥的模拟器。假如本书能让你对 ns-3 产生兴趣，那就再好不过了。

4.5 小结

本章概述了 TCP 的拥塞控制算法，并使用 Wireshark 和 ns-3 工具，通过模拟对拥塞控制算法的表现进行了观察。本章所介绍的概念是理解第 5 章和第 6 章内容所必不可少的。这里就再温习一下。

4.1 节主要介绍了与拥塞控制的目的、基本设计、状态迁移和拥塞控制算法相关的基础知识。你能说一说什么是 *cwnd* 吗？有关 *cwnd* 的内容在 4.1 节，请根据需要翻阅一下。

4.2 节概括介绍了具有代表性的拥塞控制算法，并从定性和定量两方面展示了不同算法的特点。请回顾一下图 4.3，重新复习一下各个拥塞控制算法之间的关联性。

4.3 节使用了数据包分析器 Wireshark 分析了虚拟机间传输文件时的 TCP 协议的机制。不单是 TCP，Wireshark 在各种协议的解析方面都算是一个强力的武器。

4.4 节使用离散事件驱动型网络模拟器 ns-3 观察了网络抓包工具无法获取的 TCP 内部变量的变化情况。TCP 的有趣之处正是在不断地修改其网络构成后所展现出来的各种令人惊异的表现。请随意调整模拟条件，并对比观测结果。相信读者一定会有新的发现。

笔者想通过本章内容传递的是设计拥塞控制算法的困难性。通信网络可以说是覆盖了全球的大型网络，拥塞发生时受到影响的终端设备个数也是常人无法想象的规模。

而 TCP 中的拥塞控制算法是通过 ACK 等有限数据推测整个网络的拥

堵情况的，这显然需要花费很多功夫。例如，使用有限状态机进行状态管理，各种反馈形式等都是在这个背景下所诞生的智慧结晶。近年来，也有人提出了尽力排除网络方面的前置知识，通过强化学习来进行拥塞控制的方案[1]。但是，也请大家务必记得集结了先人智慧结晶的、传统且优秀的基于模型的技术方案。

第 5 章和第 6 章中介绍的两个拥塞控制算法都是基于模型的算法中最为优秀的代表。请仔细阅读和学习这些算法，尝试去了解这些优秀的算法究竟是如何解决那些疑难问题的。

参考资料

- 《TCP 拥塞控制》（RFC 5681）.
- 《TCP 快速恢复算法的优化版 NewReno》（RFC 6582）.
- Peter L Dordal. An Introduction to Computer Networks [EB/OL].
- 《面向大型拥塞窗口的高速 TCP》（RFC 3649）.
- Carlo Caini, Rosario Firrincieli. TCP Hybla: a TCP enhancement for heterogeneous networks [J]. International journal of satellite communications and networking 22.5，pp.547-566，2004.
- Saverio Mascolo et al. TCP Westwood: Bandwidth Estimation for Enhanced Transport over Wireless Links [C]. MobiCom，2001.
- Lawrence S. Brakmo, Larry L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet [J]. IEEE Journal on selected Areas in Commnications 13.8(1995): 1465-1480.
- Tom Kelly. Scalable TCP: Improving Performance in Highspeed Wide Area Networks [J]. ACM SIGCOMM computer communication Review，vol.33，no.2，pp.83-91，2003.
- Cheng Peng Fu, S. C. Liew. TCP Veno: TCP Enhancement for Transmission Over Wireless Access Networks [J]. IEEE Journal on

[1] Wei Li, Fan Zhou, K.R. Chowdhury, et al. QTCP: Adaptive Congestion Control with Reinforcement Learning [J]. IEEE Transactions on Network Science and Engineering, 2018.

Selected Areas in Communications, vol.21, no.2, pp.216-228, 2003.

- Lisong Xu, Khaled Harfoush, Injong Rhee. Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks [C]. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM), pp. 2514-2524, 2004.
- Andrea Baiocchi, Angelo P.Castellani, Francesco Vacirca. YeAH-TCP:Yet Another Highspeed TCP [C]. Proceedings of PFLDnet, Vol.7, 2007.
- Shao Liu, Tamer Bas,ar, R.Srikant. TCP-Illinois: a loss and delay-based Congestion control algorithm for high-speed networks [J]. ACM, New York, Article 55, 2006.
- Douglas J.Leith, Robert Shorten. H-TCP: TCP for high-speed and long-distance networks [C]. Proceedings of PFLDnet, 2004.

第 5 章

CUBIC 算法

通过三次函数简单地解决问题

随着互联网的普及，TCP 也被广泛推广开来，在这段时期内，TCP 默认使用的拥塞控制算法便是 Reno 和 NewReno。然而随着近些年来互联网的高速化和云服务的普及，被称为长肥管道的宽带高时延环境逐渐流行，旧的拥塞控制算法暴露出扩展性不足和在 *RTT* 不同的网络流中吞吐量不公平等问题。

这些问题催生了 CUBIC（CUBIC-TCP），它已成为现在主流的拥塞控制算法之一。CUBIC 采用简单的算法，在解决了上述问题的同时，还具备稳定性强、与现有算法亲和性强等特点。

本章将在进行实际模拟的同时，详细介绍旧算法随着网络环境变化而暴露出的问题，以及 CUBIC 算法和其性能。

5.1 网络高速化与TCP拥塞控制

长肥管道带来的变化

随着互联网的普及而广泛推广开来的TCP拥塞控制算法是Reno和NewReno，然而随着近些年来网络的高速化，名为长肥管道的环境逐渐普及开来，这导致Reno和NewReno的低效率问题逐渐暴露出来。

Reno和NewReno 广泛使用的算法

作为标准的TCP拥塞控制算法，Reno自1990年出现以来，如前文所介绍的一样，广泛地用于各个领域。后来，为了解决Reno快速恢复算法的缺陷，NewReno算法被提了出来。具体来说，Reno的缺陷是，由于快速恢复阶段的快速重传算法是即使有1个数据包被废弃，也会进入重传模式，导致数据包发送停止，所以当发生连续丢包时，吞吐量会大幅下降。

针对这一问题，NewReno进行了改良，即在每次的重传模式下重传多个数据包。

Reno和NewReno出现之后被许多操作系统使用，并被安装在很多终端设备中。因此，可以说Reno和NewReno便是实质上的“标准TCP拥塞控制算法”。这些算法在当时的网络环境下十分高效，然而随着时间的推移，技术与网络环境不断发展，一些当时完全没有想象到的问题愈发显著。

本章将详细介绍NewReno，它比Reno使用得更加广泛。下文将首先回顾NewReno算法的特点，并概述一下其近些年随着网络环境的变化而产生的问题。

快速恢复 NewReno的特点

首先，我们用图5.1来回顾一下NewReno的拥塞窗口大小（*cwnd*）的变化情况。在最初的慢启动阶段，拥塞窗口大小是缓慢增大的，这与以

前的 Tahoe 算法是一致的。然而在进入拥塞避免阶段后采用的是快速恢复，这是 NewReno 的独特之处。

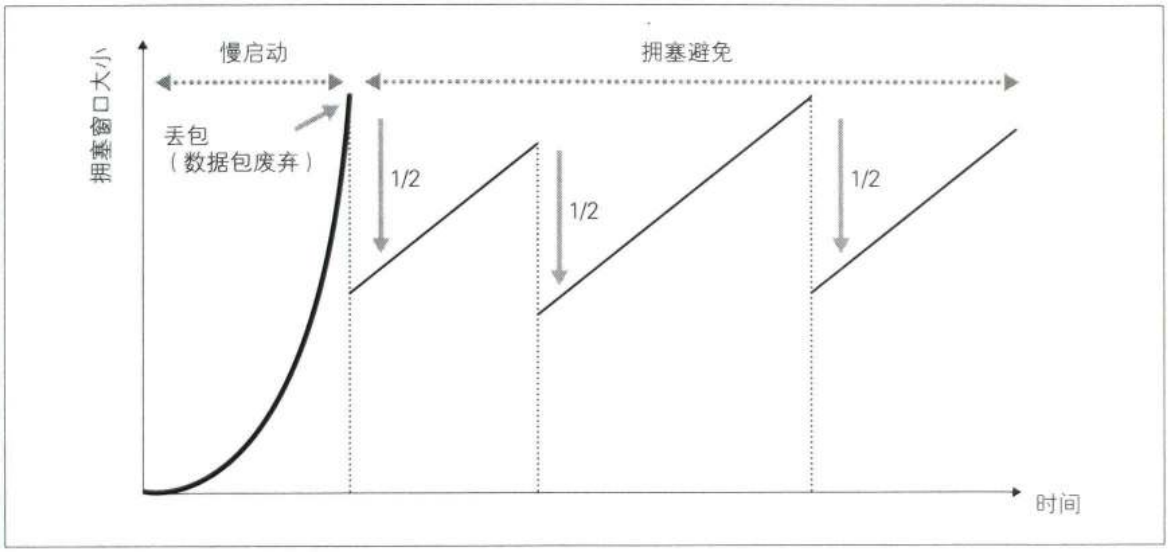

图 5.1 NewReno 的拥塞窗口大小的变化

快速恢复具体是指，在收到 3 次 ACK（重复的 ACK），也就是检测到丢包时，实际上就是检测到拥塞发生时，将拥塞窗口大小设置为拥塞发生时的 1/2，同时将 *ssthresh* 也设为同一个值，然后再缓慢地增大拥塞窗口大小。换句话说，开发 Reno 和 NewReno 的核心目的，就是防止在拥塞发生时拥塞窗口大小变得过小，进而避免出现吞吐量过低的情况。显而易见，与将拥塞窗口大小直接减小到 1 相比，使用 *ssthresh* 时吞吐量最终的恢复速度显然更快。而且在发生拥塞时，拥塞窗口大小越大，这个效果就越明显。

Reno 和 NewReno 使用了相对简单的算法，却能实现比 Tahoe 算法更高的吞吐量。这一大优点正是它们多年以来被广泛使用的关键。

网络的高速化与长肥管道 从通信环境变化的观点来看

接下来，我们来看一下 TCP 使用环境的变化。如第 2 章介绍的一样，TCP 自 20 世纪 90 年代普及以来，发生了各种各样的变化。

近些年来，网络高速化的步伐越来越快。通过 1 Gbit/s 或 10 Gbit/s 等

光纤连接的有线互联网逐渐普及，同时 5G 等能够在速度上逼近甚至超越有线网络的高速无线网络也开始推广开来。网络链路的速度不断提高，这就意味着单位时间内网络中可传输的数据量也在增加。因此，网络链路流量的传输率也有望提升。

此外，不仅仅是高速化，近些年来远程节点间的通信也变得频繁起来。例如，随着云服务的普及，用户手边的终端设备与远程数据中心中设置的云服务器之间的通信也越来越多。此外，在商业场景下，遍布各地的服务器间的数据传输、**数据中心互连**（data center interconnect，图 5.2）等应用场景也不断增多。

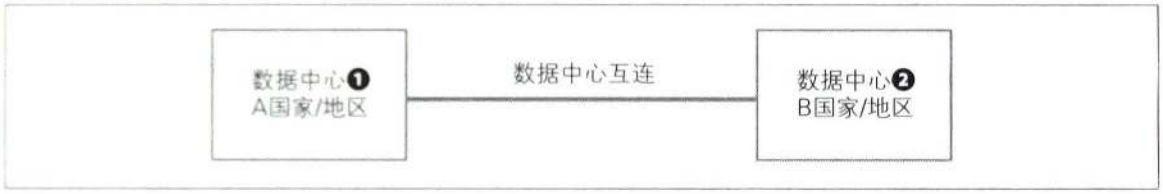

图 5.2 数据中心互连

端到端之间的三大时延 处理时延、队列时延和传播时延

从通信数据包发送到接收为止的**端到端的时延**，主要由以下三部分组成：因链路上的交换机和路由器进行数据包处理而产生的**处理时延**（processing delay）；因在链路上各个节点的队列中等待传输而产生的**队列时延**（queuing delay）；从信号发送节点到目的地节点所需要的物理上的**传播时延**（propagation delay）。

关于传播时延，我们通常认为它在无线信号下约为 3.33 μs/km（μs 是 microsecond 的缩写，表示“微秒”），而在光纤中约为 5 μs/km。在三大时延中，处理时延和队列时延可以通过提高通信节点的性能实现进一步的优化，而传播时延则只取决于传输距离。

因此，超远程节点间的通信往往无法忽视物理特性所导致的传播时延。例如，横跨太平洋的海底光缆，其传播时延超过 50 ms。

此类宽带、高时延的通信环境就被称为长肥管道（图 5.3），可以说是

近些年来最为典型的通信环境。

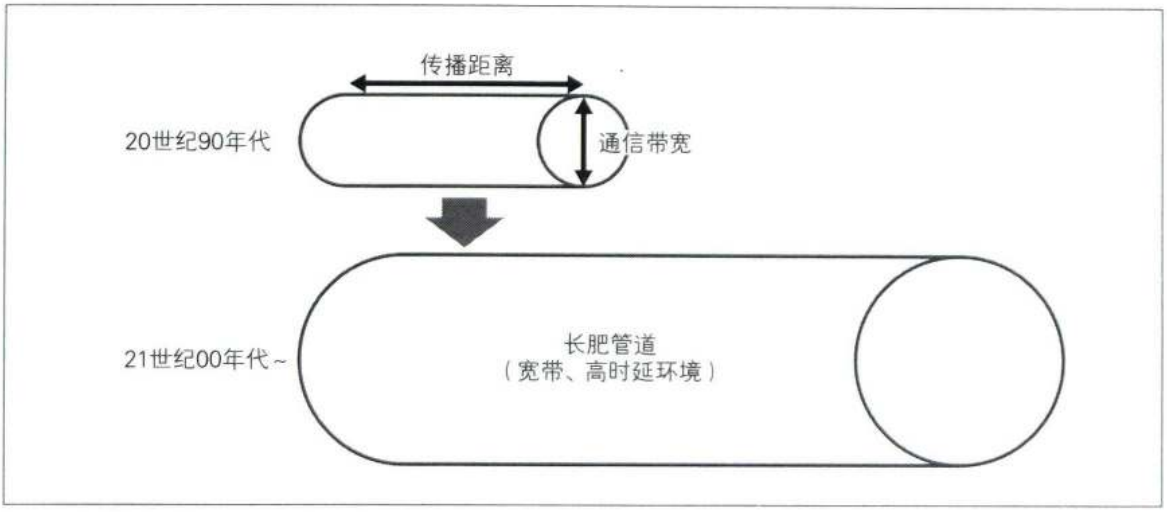

图 5.3 通信环境的变化与长肥管道

长肥管道下 NewReno 的新问题

随着长肥管道在 TCP 使用环境中的范围逐渐扩大，人们在以往默认采用的 NewReno 算法的一些问题也逐渐暴露出来。究其原因，主要是 NewReno 诞生于网络通信速度和可靠性都比较低的时代，所以它被开发出来也是为了适应这种环境，而对于现在的宽带高时延环境，它并不适合。

下文将简单介绍一下具体的问题。

——无法有效地利用宽带

首先，第一个问题是相比链路速度，拥塞窗口大小的增幅过小，无法有效地利用宽带的优势。

如图 5.4 所示，其原因主要是，在进入拥塞避免阶段之后的快速恢复状态时，链路速度越快，恢复吞吐量所需要的时间也就越长。由于 NewReno 会将拥塞窗口大小调整为 1/2，并在之后缓慢增大，所以与在窄带环境下相比，在宽带环境下拥塞窗口大小相对于线路传输率（wire rate，传输链路上的最大数据传输速度）进行恢复的话，恢复所需的时间即使不长，也

会被拉长，也就是说无法有效地利用带宽。

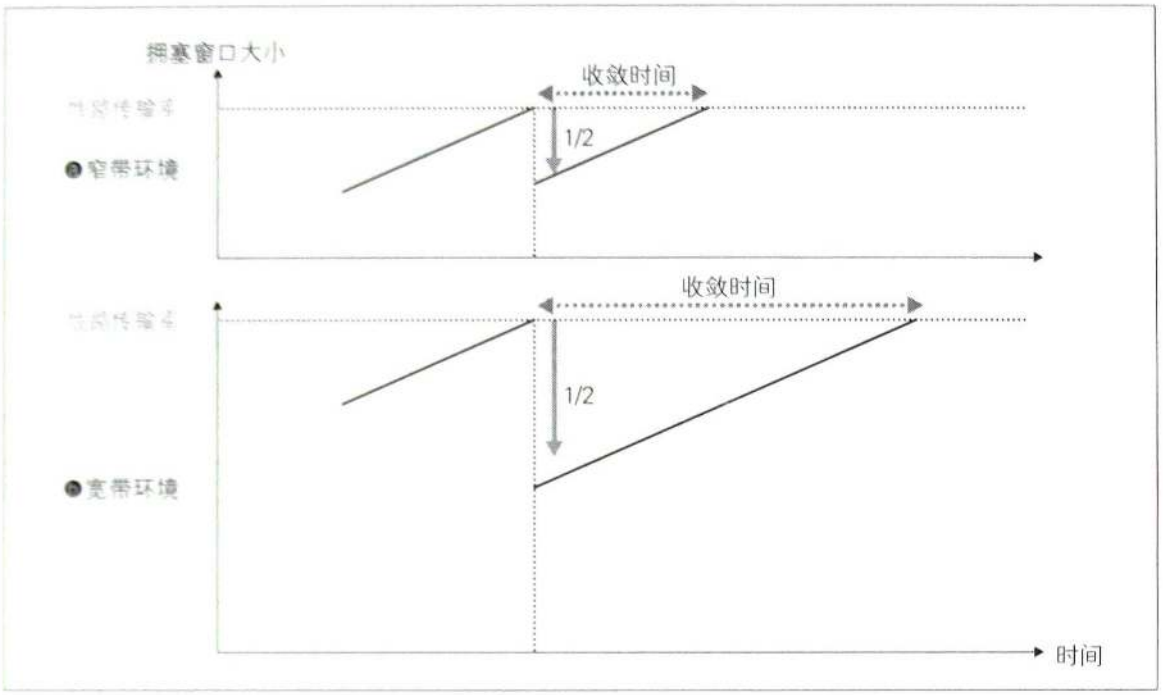

图 5.4 宽带化导致收敛时间增加

宽带环境下发送率不足

第二个问题就是 RFC 793 所规定的最大拥塞窗口大小是 65 535 字节，因此即使将拥塞窗口大小设置为最大，在宽带环境下发送率也不足。不过针对此问题，新标准制定了一个窗口扩大选项，使通信可以在 TCP 协商阶段使用此选项增大最大拥塞窗口大小。

时延越大，吞吐量越低

另外，如图 5.5 所示，发送方节点每次收到与发送的数据对应的 ACK 之后，都会计算 *RTT* 的值，随后更新拥塞窗口大小并发送数据。此时，随着传输距离的增大，*RTT* 也会增大，与之同时，ACK 的接收时间也会进一步延长。换句话说，数据收发的间隔，即图 5.5 中的等待时间会进一步延长。